



# Introdução

As tecnologias de hardware confiável estão em todo o lado e não subestimadas!

- Caixas regras para manter chaves criptográficas
- Âncoras de confiança

**Cartão SIM:** mecanismo para autenticação — simples; barato; fiável

**Cartão de Crédito/Débito:** criptografia no seu epicentro

**Hardware Security Module:** armazenamento de chaves numa caixa regra com acesso altamente escrutinado e protegido, não vulnerável a ataques forçados

**Java Cards:** excelente compatibilidade — escreve uma vez, corre em qualquer lado

Os adversários podem medir precisamente o tempo de execução, a temperatura e sinais elétricos, bem como quebrar o hardware, porque o transportam

O especialistas devem saber como usar e quando usar!

1. Intel SGX/1DX: execução isolada e atestação remota

2. ARM TrustZone: execução isolada e nó de aplicações verificadas

3. OpenTitan: RISC-V — open-source

**Ataques:** canais colaterais

1. análise de potência

2. injeção de falhas

3. execução especulativa

**Contro-Medidas:** implementações de tempo constante — tempo de execução constante, ortogonal a parâmetros privados (não influência)

Não é possível projetar um sistema totalmente seguro usando hardware confiável!

# Visão Geral

Criptografia Simétrica: mesma chave pré-partilhada conhecida pelos utilizadores

AES; MAC; AEAD

Criptografia de Chave Pública: chaves públicas diferentes partilhadas pelos utilizadores

RSA; Assinatura Digital; Diffie-Hellman

Base de Computação Confável: componentes que se assumem comportar como esperado

Smart Card: armazena e processa informação num micro-controlador embutido em plástico portátil, resistente a fraude e auto-contido

Java Card: portabilidade; segurança; independência de hardware; baixo custo

Applet: programa Java aderente a convenções executável no ambiente do cartão

APDU: pacote de dados para comunicação ao nível da aplicação - half-duplex

Trust Platform Module: microchip para funcionalidades de segurança que não assenta no SO para processamento e realiza operações criptográficas em isolamento - medição de integridade e inicialização segura

Ray de Confiança para Medição: valor predefinido para código crítico

Hardware Security Module: chaves não exchangeis e resistentes a fraude, com otimização criptográfica e alta disponibilidade

1. Rede

2. Interno

3. Offline

4. Cloud

Intel SGX: enclaves & atestação

Intel TDX

ARM TrustZone: mundo normal & mundo seguro

Ataques: canais colaterais; temporização; injeção de falhas; análise de potência; execução especulativa

# SmartCard

SmartCard: armazena e processa informação dentro de um micro-controlador embutido em plástico - computador portátil pequeno resistente a fraude

Auto-Controle: não necessita de depender de recursos externos potencialmente vulneráveis

SmartCard Magnético: não tem chip embutido

SmartCard: transporta mais informação e pode computar ou tomar decisões

A informação ou aplicação armazenada no chip é transferida através de um módulo eletrónico que se interconecta com um terminal ou leitor de cartões

ISO 7810/7816: circuito impresso e chip integrado embutidos no cartão

- Circuito Integrado
- |                 |  |
|-----------------|--|
| 1. CPU          | 346 sistema operativo e software básico<br>32 computação e resposta rápidas<br>1024 retem estado quando a alimentação é removida |
| 2. ROM          |  |
| 3. RAM          |  |
| 4. EEPROM/FLASH |  |
| 5. PROM         |  |
| 6. EEPROM       |  |

Input/Output: via porta de I/O única controlada pelo processador

Dispositivo de Interface: fornece alimentação e sinal de relógio  
 $\downarrow$  ABRE

Canal de Comunicação: half-duplex - dados armazenados em buffers na RAM

"Bit/Band Rate": rácio de amostragem da emissão pelo receptor para comunicação

- A autenticação com PIN funciona como segundo fator
- O cartão no leitor interage com a máquina local e com a rede
- A chave privada nunca sai do cartão!
- Todas as computações críticas ocorrem internamente

Ex:

1. Inserir o cartão no leitor — contém chaves criptográficas e dados biométricos
2. Introduzir PIN — desbloquear a representação digital da impressão digital
3. Colocar o dedo no leitor — comparar
4. Se corresponder, impressão digital + PIN → chave simétrica  $\xrightarrow{D}$  chave privada
5. Passar "nonce" para o cartão
6. Encriptar o "nonce" e devolver à aplicação
7. Verificar a desencriptação

Smart Cards:

1. Contacto: pinos no leitor — leitura e escrita
2. Contactless: antena embutida — leitura e escrita
3. Proximidade: antena embutida — leitura
4. Híbrido: dois chips (contacto e "contactless")
5. Combi: um chip de interface dual (contacto e "contactless")
6. Criptográfico: suporte adicional para operações criptográficas — inviolável
7. Multi-Aplicação: memória aloçada em secções independentes

Sistema Operativo do Chip: sequência de instruções permanentemente embutidas na ROM — determina as características funcionais e GERA TUDO

"Application Protocol Data Unit": contém comandos ou mensagem de resposta

- O smartcard espera pelo comando APDU do terminal — papel passivo
- O smartcard executa a ação e responde com uma APDU de resposta

## Transação:

1. Conexão: interação no ou aproximação ao leitor
2. Autenticação do Cartão: geração de uma mensagem e confirmação
3. Autenticação do Leitor: geração de uma mensagem e confirmação
4. Seleção de Aplicação: automática ou manual
5. Identificação de Requisitos de Segurança: definidos/forçados pelo cartão
6. Autenticação do Proprietário: PIN ou biometria
7. Transação: geração, verificação e autorização
8. Registo da Transação: geração pelo cartão e transmissão para o leitor
9. Cópia: registo em papel

## Ciclo de Vida:

1. Fábrico: adição de uma chave de fábrico para prevenir fraude/manipulação
2. Pré-Personalização: chave de fábrico substituída por chave de personalização
3. Personalização: escrita de dados aplicacionais
4. Utilização: activação da aplicação
5. Inutilização/Fim-de-Vida: desactivação das operações de escrita/actualização

PKCS #11: especificação do API Cryptoki

PKCS #15: padronização dos tokens criptográficos - interoperabilidade

Multi-Plataforma: aplicável a arquiteturas e sistemas operativos

Participação Aberta: aceita inputs e revisões

Interoperabilidade: interoperável com padrões e protocolos

Real/Funcional: aplicável a problemas reais

Experiência/Produtos: criado por pessoas experientes

Extensibilidade: facilita expansão a novas aplicações/capacidades

• Os cartões não sempre redefinem quando são inseridos num leitor!

Ataque Invasivo: aceder diretamente à superfície do chip

1. Desembalagem: remoção do chip
2. Reconstrução: reconstrução do layout
3. MicroAmostragem Manual: microscópio ótico especial
4. Leritura de Memória: registo dos valores em memória

Ataque Não-Invasivo: não danificam fisicamente o cartão

1. Softwares: exploração de vulnerabilidades de segurança
2. Geração de Falhas: uso de condições ambientais anormais
3. Glitch: geração deliberada de um mau funcionamento
4. Eavesdropping: aproveitamento das características analógicas e/ou radiação

Ataque Físico: engenharia reversa para determinar as chaves secretas

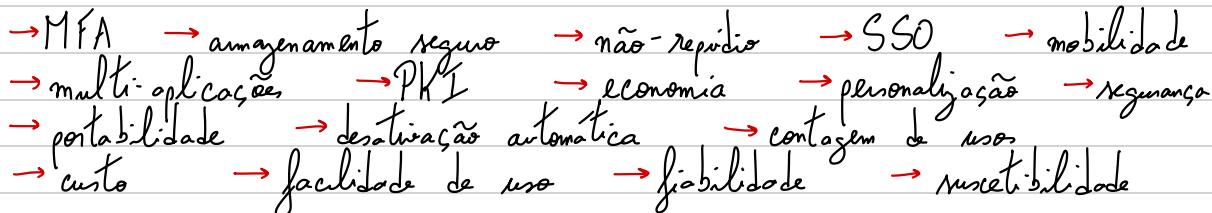
Ataque Lógico: ganhos de informação numérico examinam os bytes de e para o cartão extraem informações pelos tempos e números para a chave privada

Ataque De Cavalo De Troia: aplicação aguarda pela submissão de um PIN válido por uma aplicação confiável para plciar os cartões para armazenar digitalmente dados com a chave privada sem o conhecimento do utilizador

↓ PREVENÇÕES

1. arquitetura de driver de acesso único - só uma aplicação de cada vez
2. um uso de chave privada por entrada de PIN

## FUNCIONALIDADES



# Java Card

Java Card: versão de Java para smartcards - compatibilidade  
↳ torna possível cartões multi-plataforma baseados numa plataforma comum  
↳ facilidade de desenvolvimento      ↳ segurança      ↳ independência de hardware

Read-Only Memory (ROM): armazena o programa fixo da cartão (sistema operativo e dados permanentes) Nem necessidade de alimentação, mas nem possibilidade de escrita - acessível um número ilimitado de vezes

Electrical Erasable Programmable Read-Only Memory (EEPROM): preserva dados quando a alimentação é desligada durante 10 anos - rápida de ler mas lenta (e limitada) de escrever

Random Access Memory (RAM): memória não-persistente usada como espaço de trabalho temporário para armazenar e modificar dados, mas não preservada quando a alimentação é removida - acessível um número ilimitado de vezes

Memória Flash: memória persistente e mutável como a EEPROM, mas mais eficiente em potência e espaço do que a EEPROM, podendo ser lido bit a bit, mas atualizada como um bloco - para armazenar programas adicionais

Sistema Operativo: conjunto de instruções no formato de APDUs

Sistema de Ficheiros:

1. Ficheiro Mestre: raiz do sistema de ficheiros - 1
2. Ficheiro Dedicado: ficheiro directorio que contém outros ficheiros
3. Ficheiro Elementar: ficheiro de dados que não pode conter outros ficheiros

Sistema Ampliáculo: no computador ligado ao leitor - sistema e aplicações

Sistema Cartão: no smartcard - segurança e gestão de chaves, memória e I/O

O caminho de comunicação entre o anfídio e o cartão é "half-duplex"!

Half-Duplex: os dados podem ser enviados num sentido ou no outro, mas não em ambos ao mesmo tempo

APDU de Comando {

<u>Cabeçalho</u>	<u>CLA</u> : classe da instrução
	<u>INS</u> : código da instrução
<u>Corpo</u>	<u>P1/P2</u> : parâmetros
	<u>Lc</u> : comprimento dos dados
	<u>Data</u> : dados para o cartão
	<u>Le</u> : bytes esperados na resposta do cartão

APDU de Resposta {

<u>Corpo/Dados</u>
<u>SW1</u>
<u>SW2</u>

ESTADO DE PROCESSAMENTO

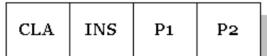
Transmission Protocol Data Unit (TPDU): estrutura de dados trocada pelo anfídio e o cartão pelo protocolo de transporte TCP/IP

Answer to Reset (ATR): mensagem enviada pelo cartão para o anfídio imediatamente após seu alimentado, contendo os parâmetros necessários pelo cartão para estabelecer um caminho de comunicação de dados — 33 bytes

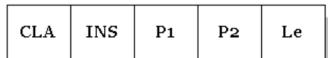
Java Card Virtual Machine (JCVM): define um subconjunto da linguagem de programação Java e da arquitetura da máquina virtual aplicável às aplicações SCVM = conversor (fora do cartão) + interpretador (dentro do cartão) — executa gera objetos

Java Card Runtime Environment (JCRC): descreve o comportamento em tempo de execução do Java Card, incluindo gestão de memória e de applets — JC OS

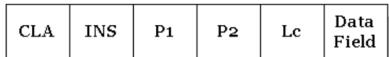
**Case 1:**  
No Command data,  
No Response required



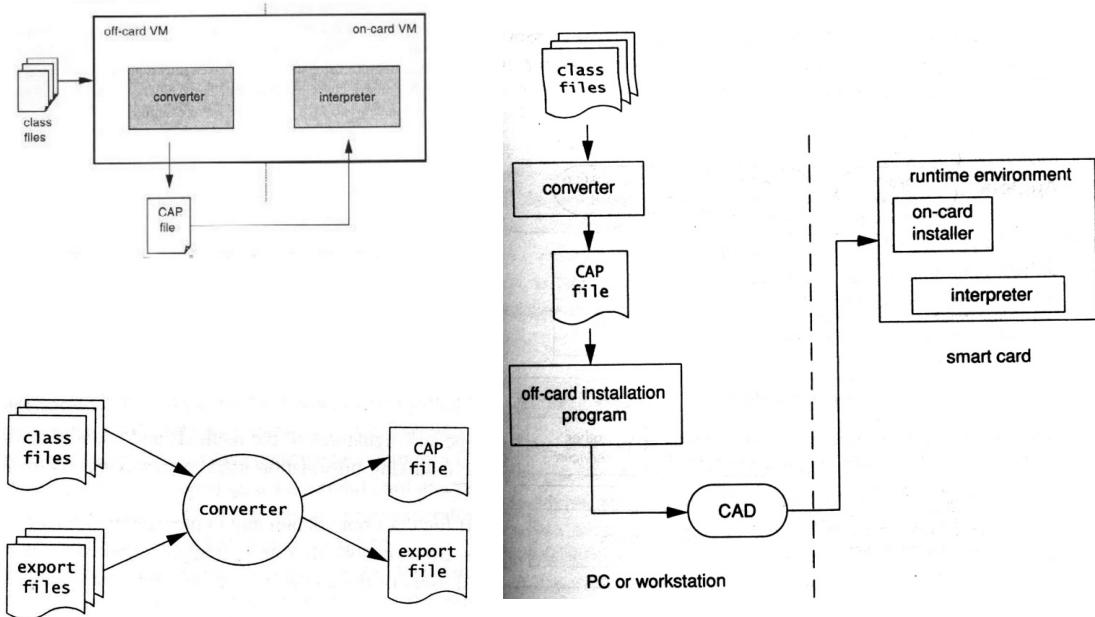
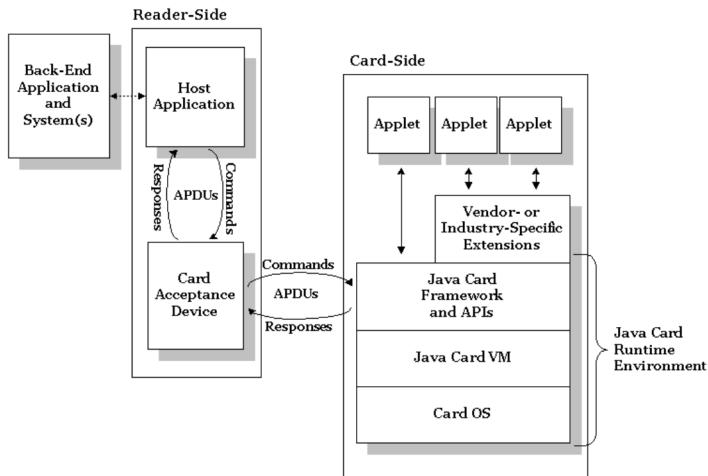
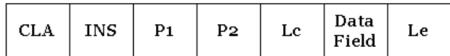
**Case 2:**  
No Command data,  
Yes Response required



**Case 3:**  
Yes Command data,  
No Response required



**Case 4:**  
Yes Command data,  
Yes Response required



• Tipos primitivos pequenos; arrays unidimensionais; pacotes; classes; interfaces; exceções  
• Tipos primitivos grandes; caracteres; strings; arrays multidimensionais; threads

Ficheiro CAP: ficheiro JAR otimizado que contém um conjunto de componentes descritivos e uma representação binária executável da classe no pacote

Ficheiro EXPORT: cabeçalho não carregado/usado que contém a informação da API pública e informações de ligação, definindo âmbito, armadura, ação, ...

Conversor: realiza tarefas em tempo de carregamento das classes

Interpretador: apoia a execução do modelo de linguagem Java independentemente  
→ não carrega o próprio ficheiro CAP, mas descarrega-o e instala-o

As classes do sistema gerem transações, comunicações e applets

JAVA CARD  $\xrightarrow{\text{INICIALIZA}}$  JCPE  $\xrightarrow{\text{INICIALIZA}}$  JCVM  $\xrightarrow{\text{GERE}}$  APPLET  $\xrightarrow{\text{CPMA}}$  OBJETO

Sessão CAD: período do cartão no dispositivo que o aceita e alimenta

O JCPE espera por comandos em ciclo!

O applet toma o controlo, executa o comando, envia resposta e devolve o controlo

JCPE {  
1. objetos persistentes e transitórios (RAM)  
2. operações e transações atómicas  
3. firewall de applets e mecanismo de partilha — isolamento de applets

API {  
1. java.lang — objetos e exceções  
2. javacard.framework — classes para as funcionalidades nucleares  
3. javacard.security — funções criptográficas  
+ javacard.crypto — extensão para o processador criptográfico

Applet: programa Java que adere a um conjunto de convenções, pode executar no ambiente de execução Java Card e ser dinamicamente descarregado no cartão JCRE

Podem coexistir múltiplos applets no mesmo cartão — identificados unicamente  $(\text{AID}_{\text{RID+PIX}})$

1. escrita de classes, compilação e execução
2. execução, teste e debug em ambiente simulado
3. conversão para CAP
4. carregamento e execução dos CAP

"Masking": escrita das componentes permanentes na memória imutável do chip

Applet ROM: instanciado na EEPROM pelo JCRE durante a sua inicialização → conteúdo controlado pelos fabricantes e permitido declarar métodos nativos

Applet Pré-Emissão: descarregado antes ou depois da emissão, mas tratado como ROM

Applet Pós-Emissão: descarregado antes ou depois da emissão, mas nem permitido para declarar métodos nativos por razões de segurança

Instalação de Applet: processo de (1) carregar as classes applet num ficheiro CAP, (2) combiná-las com o estado de execução do JCRE e (3) criar uma instância do applet para trazer o applet a um estado selecionável e de execução

Ficheiro CAP: unidade carregável e instalável que consiste em classes e num applet mínimo que é um pacote Java com uma única classe derivada de Applet

CAP  $\xrightarrow{\text{OFF}} \text{comandos APDU}$   $\xrightarrow{\text{ON}} \text{CMD}$  escreve o CAP em memória e liga as classes

install(): cria uma instância do applet e regista-a no JCRE — ponto de entada que tem de ser implementado para chamar os construtores para criar e inicializar

- O processo de instalação é transacional!
1. applets em execução no cartão só se podem referir a classes já no cartão
  2. a ordem de carregamento deve garantir (1)

**Objeto:** instância de classe/array — um objeto não referenciado é inacessível

**Objeto Persistente** <sup>NEW!</sup>: armazena estado e valores entre sessões CAD, atualizado atomicamente — espaço irreversível se o JCSE não tiver um coletor de lixo

**Objeto Transitório**: os conteúdos dos campos têm uma natureza temporária, mas o espaço é reservado e não pode ser recuperado até ser limpo pelo coletor de lixo  
→ não armazena estado entre sessões → não é atualizado atomicamente

Um applet deve criar um objeto transitório uma só vez no seu ciclo de vida e guardá-lo num campo persistente para usar sempre a mesma referência

**CLEAR\_ON\_RESET**: objeto transitório com dados limpos entre resets do cartão

**CLEAR\_ON\_DESELECT**: objeto transitório com dados limpos entre seleções e resets

**Atomicidade**: qualquer atualização a um campo único de um objeto persistente ou de uma classe ou completa com sucesso ou é restaurada ao valor original

Num array transitório, um erro define os valores predefinidos

1. arrayCopy: garante atomicidade através de passos escrita na EEPROM
2. arrayCopyNonAtomic: não usa as capacidades transacionais
3. arrayFillNonAtomic: preenche todos os elementos de forma não atomica

**Transação**: requisiçãoatómica de operações — begin(); commit(); abort()

As transações não podem ser aninhadas — só uma de cada vez

Commit Buffer: armazena os conteúdos originais dos campos a atualizar até que a transação seja "committed" — capacidade variável get Max / get Unused

TransactionException: IN\_PROGRESS; NOT\_IN\_PROGRESS; BUFFER\_FULL; INTERNAL\_FAILURE

As atualizações a objetos temporários e variáveis locais nunca são desfeitas!

Exceção: evento que disrompe o fluxo normal de instruções durante a execução de um programa — forma elegante de lidar com erros na plataforma  
THROW CATCH TRY FINALLY

Throwable → Exception → RuntimeException

Exceções Verificadas: lançadas ou apanhadas — indicam um erro de programação

Exceções Não Verificadas: não necessariamente lançadas/apanhadas — erro de execução  
throwIt(reason)

As classes CardException e CardRuntimeException permitem um mecanismo de poupança de recursos para que uma exceção possa ser reutilizada várias vezes

Se um applet criar um objeto de cada vez que uma exceção é lançada, vai acumular muitas instâncias de exceções anteriores não utilizadas na EEPROM  
↓ OTIMIZAR MEMÓRIA

Todas as exceções devem ser pré-criadas em tempo de inicialização e as suas referências guardadas permanentemente para preencher com o código e reutilizá-las

NÃO VERIFICADA

ISOException: encapsula a palavra de estado da resposta no código da rayão

O JCSE apanha a exceção e retorna o código como a palavra de estado

Um applet é um objeto persistente que vive durante todo o tempo de vida do cartão

## Passo de Instalação de Applet:

1. Carregar no smartcard
2. Ligar com outros pacotes
3. Criar e registrar com JCPE

- JCPE é um ambiente de thread único — só um applet em execução de cada vez
- Quando um applet é instalado, está inativo
- O applet torna-se ativo quando é explicitamente selecionado pelo utilizador
- Os applets não aplicações reativas
- Um applet selecionado espera por um comando enviado pela aplicação
- O diálogo comando-resposta continua até o applet ser desselecionado

**SELECT (AID):** JCPE procura applets e seleciona —  $0x00 A4 04 00$

CLA INS P1 P2

→ install: cria instância → select: inicializa → process: entrega APDU → deselect: limpa

• Todos os objetos devem ser alocaos no construtor

• O objeto APDU encapsula a mensagem num array de bytes interno — buffer

**Comando Bem Formado:** os bytes do cabeçalho estão corretamente codificados

**Comando Executável:** o comando é suportado pelo applet e satisfaz as condições

get Incoming And Receive(): lê dados do buffer APDU

get Outgoing And Send(): escreve dados no buffer APDU

**Contexto:** espaço de objetos protegido separado particionado pelo firewall

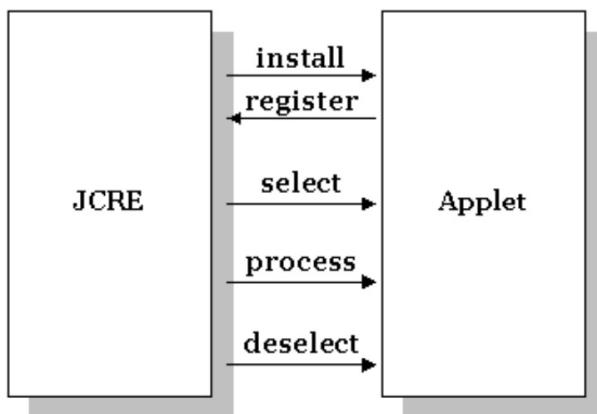
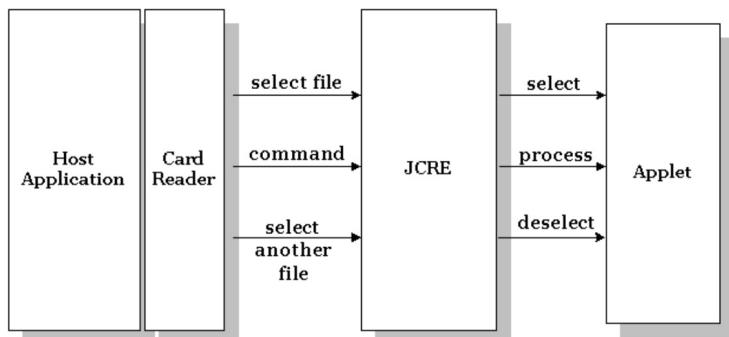
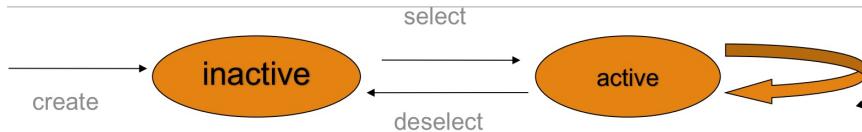
• Não há firewall entre duas instâncias do mesmo contexto de grupo

• Só há um contexto ativo de cada vez: JCPE ou grupo de applet

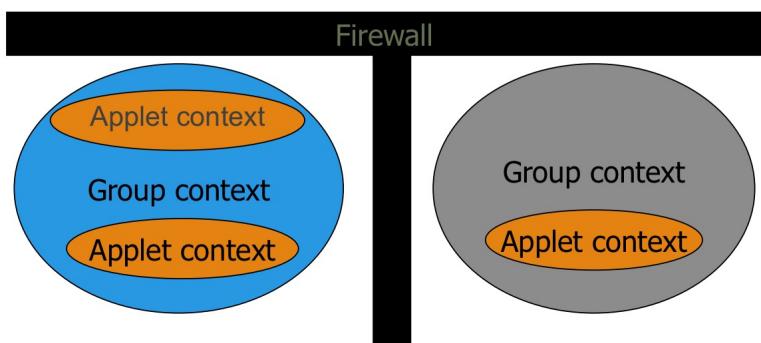
• Só os objetos são propriedade do contexto; classes não são

**Troca de Contexto:** JCPE applet

Process a command



JCRE context



## Criptografia:

- (1) garantir comunicação segura de dados entre aplicação e anfitrião
- (2) funcionar como token seguro e proteger chaves de encriptação

Arquitetura de Criptografia em Java: métodos "fábrica" - interface/implementação

(1) convenção de nomenclatura  
(2) parâmetros de seleção } EXTENSÃO FLEXÍVEL

O suporte a novos algoritmos faz-se ao adicionar parâmetros de seleção

javacard.security: interfaces para chaves, hashes, armaduras e aleatórios

javacardx.crypto: classe Cipher e interface Key Encryption - controlo de exportação

— //

Personal Computer/Smart Card (PC/SC): especificação para integração de smart-cards em ambientes computacionais - interoperabilidade e API de alto-nível

Um Cartão de Circuito Integrado (ICC) deve cumprir com a ISO 7816

Dispositivo de Interface (JFD): dispositivo terminal através do qual o cartão comunica com o PC

JFD Handler (JFDH): módulo de software de baixo-nível que lida com I/O - drivers

Gestor de Recursos: lida com os recursos, gera o envio de APDUs e gera canais

Fornecedor de Serviço: fornece uma API de alto-nível - API → APDU

GlobalPlatform: especificação padrão para a gestão de infraestrutura de um smart-card que define um conjunto de componentes lógicos que reúnem garantia, segurança, portabilidade e interoperabilidade dos smart-cards - fornecedor como autoridade central

1. OP-READY: domínio instalado e chaves iniciais armazenadas — lido com APDUs
2. INITIALIZED: o cartão ainda não está pronto para ser entregue ao proprietário
3. SECURED: chaves de segurança do domínio e funcionalidades de segurança
4. CARD-LOCKED: fornecedor bloqueou acesso a domínios de segurança e terceiros
5. TERMINATED: funcionalidades desativadas e só recebe dados

- a. INSTALLED: aplicação adequadamente instalada e ligada aos componentes
- b. SELECTABLE: aplicação capaz de receber comandos de entidades externas
- c. LOCKED: aplicação não pode ser selecionada e só o fornecedor pode desbloquear

**Domínio de Segurança**: aplicação privilegiada que representa no cartão uma entidade fora do cartão — capacidade de iniciar canais seguros e lidar com funções de gestão de conteúdo **ÁREA PROTÉGIDA**

1. Fornecedor: obrigatório
2. Suplementar: fornecedor de aplicação
3. Autoridade Controladora: força as políticas de segurança

**Gestor de Cartões**: lida com a gestão de aplicações e determina estados do cartão

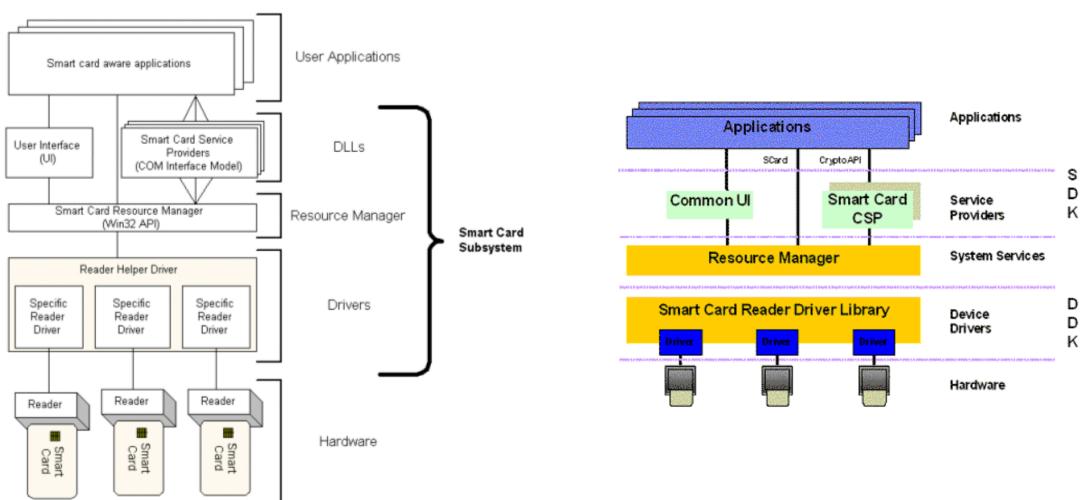
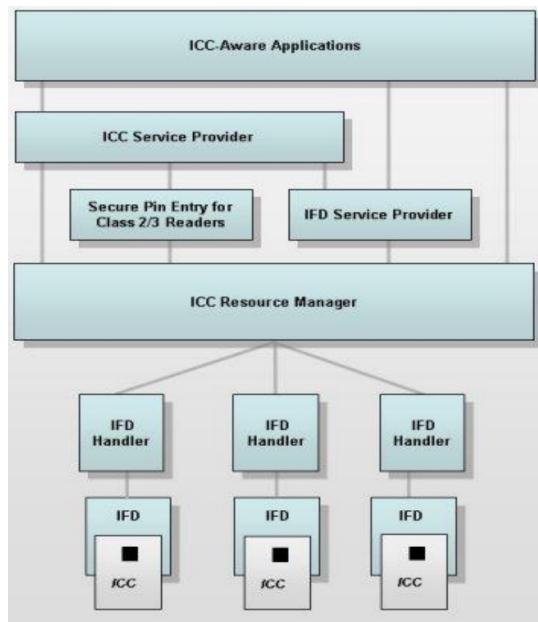
— // —

**PKCS**: grupo de padrões de criptografia de chave pública

**PKCS #11**: API para interagir com tokens criptográficos

**PKCS #12**: formato de ficheiros de arquivo para armazenar objetos criptográficos

**PKCS #15**: estrutura de ficheiros para guardar, proteger e usar objetos criptográficos  
neutralidade de plataformas, fornecedores e aplicações



# Trusted Platform Module

**Trusted Platform Module (TPM):** microchip para funcionalidades de segurança que não pertence ao SO para processamento e realiza operações criptográficas em isolamento

**Semelhanças:** pegada pequena; baixo custo; circuito integrado resistente a fraude  
**Diferenças:** bloco de construção para computação confiável com um token fixo vinculado a uma plataforma específica — controlado pelo utilizador

Um sistema seguro deve ter componentes fisicamente seguros que podem ser usados como fundação a partir da qual a confiança no sistema pode ser construída!

**Plataforma Confiável:** plataforma computacional que tem um componente confiável que é usado para criar uma fundação de segurança para processos de software

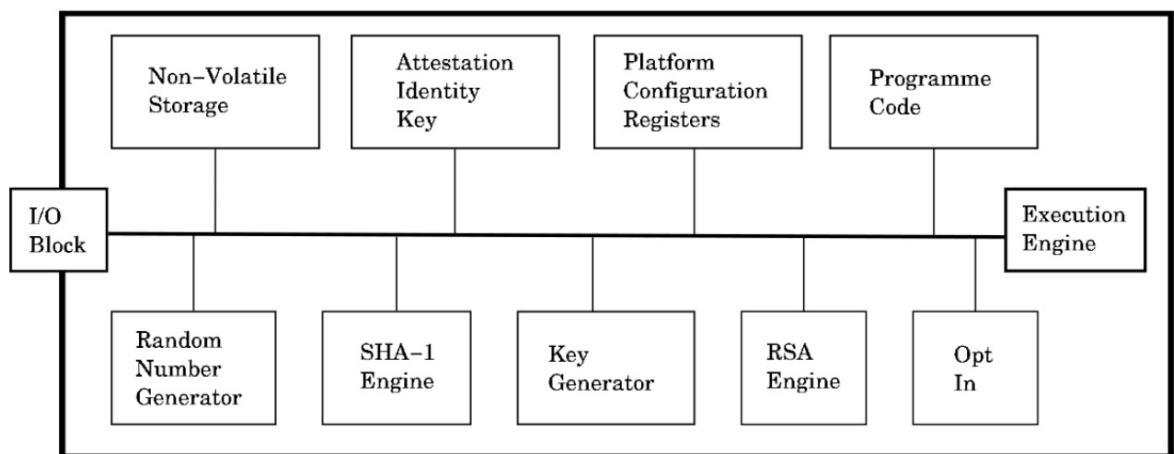
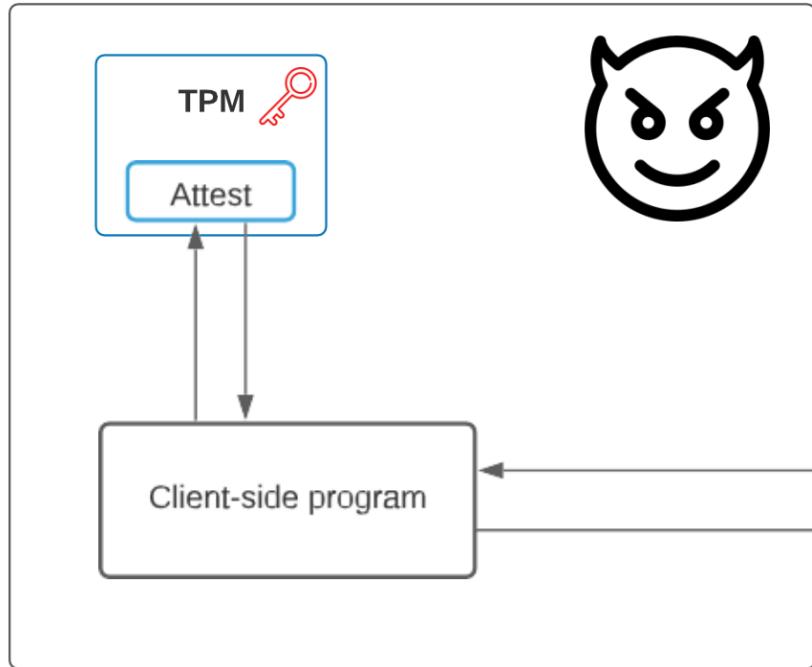
1. Capacidades Protegidas: localizações escondidas para operar sobre dados sensíveis  
a. proteção e reporte de medições de integridade  
b. geração interno seguro de chaves e processamento criptográfico  
c. armazenamento e gestão de chaves criptográficas

2. Medição e Reporte de Integridade: garantir configurações seguras da plataforma  
a. análise integridade do sistema — valores de hash de processos críticos de inicialização escondidos de fraude externa

b. Raiz de Confiança para Medição (RTM)  
c. Reporte de integridade — tem uma chave secreta de assinatura que pode assinar dados de forma segura com um certificado para verificar

**Chave de Atestação de Identidade:** única ao TPM para identificar e atestar

**Chave Raiz de Armazenamento:** para armazenar dados do TPM fora do ambiente



# Hardware Security Module

Hardware Security Module (HSM): caixa-preta para operações de segurança com armazenamento seguro de chaves não exchangeáveis, activação de M em N, autenticações multi-fator e resistente a fraude - necessário para aplicações de alto risco

Semelhança: escudo de dados e operações - âncora de confiança inutíl contra perpétuos  
Diferença: não vinculado a uma plataforma - múltiplos utilizadores e desempenho

- altamente especializado; libera recursos; altamente disponível
- investimento; manutenção; recursos adicionais de IT; conhecimento especializado

1. Rede: partilhado entre servidores e aplicações na mesma conexão LAN ~~é~~ VM
2. Interno: instalado no slot PCI, não partilhado ~~é~~ não compatível com VM
3. Offline: conectado via USB, usado para CA rai ~~é~~ não compatível com VM
4. Cloud: centro de dados na nuvem, usado por aplicações para gestão de chaves

HSM Genuíno: confiável para desempenhar operações criptográficas

Amfitrião Genuíno: confiável para verificar consistência do estado do domínio e para colocar pedidos para os HSMs no domínio

Operador Genuíno: só atesta entidades genuínas

Operador Desonesto: pode atestar afirmações falsas

Confiança Honesta: (1) todas as entidades são genuínas, (2) todo o quórum de operadores contém um operador honesto e (3) o seu antecessor é honesto PERMANECE HONESTA

Gestão de Confiança: para modificar uma confiança existente, o HSM verifica que (1) a confiança é consistente (verificada pelo HSM na confiança existente), (2) o próprio está na confiança existente e (3) toda a entidade da nova confiança ou está na confiança antiga ou é atestada por um quórum de operadores da confiança existente

A segurança das confianças está predicada no hardware confiável

# Intel SGX

**Computação Segura "Outsource":** o utilizador C tem um programa que quer que seja executado remotamente no Núcleo S, mas estando protegido contra actividades maliciosas no Núcleo — processar dados sem os ler

**Tecnologias de Reforço da Privacidade:** tecnologias que preservam a segurança e permitem a computação — criptografia

**Intel SGX:** permite que aplicações corram com confidencialidade e integridade no ambiente nativo do sistema operativo, sem requerer hardware adicional

**Servidor:** armazena uma chave HTTPS; corre código privado na cloud

**Cliente:** esconde armaduras dos antivírus

**Enclave:** container isolado para execução de código, cujos conteúdos não podem mudar depois da inicialização — mensagens produzidas autenticadas e vinculadas

Os outputs devem ser produzidos por software genuíno!

**Intel TDX:** permite a utilização de hardware confiável a partir de máquinas virtuais

**Memória Reservada para o Processador (PRM):** o CPU é responsável por proteger a PRM de ataques externos para a memória reservada para o enclave

**ECREATE:** estabelece o endereço de memória para o enclave

**EADD:** copia páginas de memória para o enclave

**EEXTEND:** computa o hash de conteúdos do enclave

**EINIT:** verifica se o conteúdo hashed está corretamente armazenado → inicializa

**EENTER:** chama uma função dentro do enclave

**EEXIT:** retorna a partir do enclave

Segurança Inteira-Plataforma: o microprocessador mantém uma chave criptográfica para cada enclave — mecanismo de autenticação

1. A chama EREPORT para m para B
2. SGX busca a chave do enclave B
3. SGX produz um MAC de m e o código de A
4. SGX retorna o MAC a A
5. A envia o MAC para B, juntamente com m
6. B pede a chave via EGETKEY e valida a mensagem

Segurança Inter-Plataforma: o microprocessador tem uma chave de armazenada da Intel, gerida pelo "quotting enclave"

1. O enclave gera um MAC para o "quotting enclave"
2. O enclave envia a sua informação com o MAC para o "quotting enclave"
3. O "quotting enclave" verifica e produz uma quote

A "quote" contém uma assinatura digital produzida por uma chave não acessível através do quotting enclave

Parte Não Confiável: gera a aplicação

Parte Confiável: realiza operações críticas

- A geração de alatórios deve seguir mecanismos específicos
- As bibliotecas de alocação de memória devem ser compatíveis com SGX
- I/O está desativado nos enclaves

• edl: define a API — funções chamadas pelo ambiente não confiável

→ O Intel SGX fornece computação segura de propósito-geral

Atestação Inteira-Plataforma:

Atestação Inter-Plataforma:

# ARM TrustZone

Objetivo: eficiência; abrangência do sistema; inicialização segura  
isolamento forçado pelo hardware construído no CPU

Funcionalidades: garantias de confidencialidade e integridade num ambiente programável

Mundo Seguro: hospeda memória segura e contentores  
Mundo Normal: hospeda a pilha de software padrão

Os contentores seguros para execução de código confiável não necessários para as garantias de hardware confiável, que só podem correr aplicações confiáveis!

ASSINADAS

ARM TrustZone: um ambiente para aplicações confiáveis que não é funcionalmente especializado, mas o mundo Seguro não pode correr aplicações arbitrárias

O núcleo do CPU TrustZone está equipado com duas tabelas de páginas de registos-base que fornecem uma tradução de endereços separada entre mundos

O endereço inclui um bit adicional de segurança que estabelece se os conteúdos pertencem ao mundo Seguro ou normal

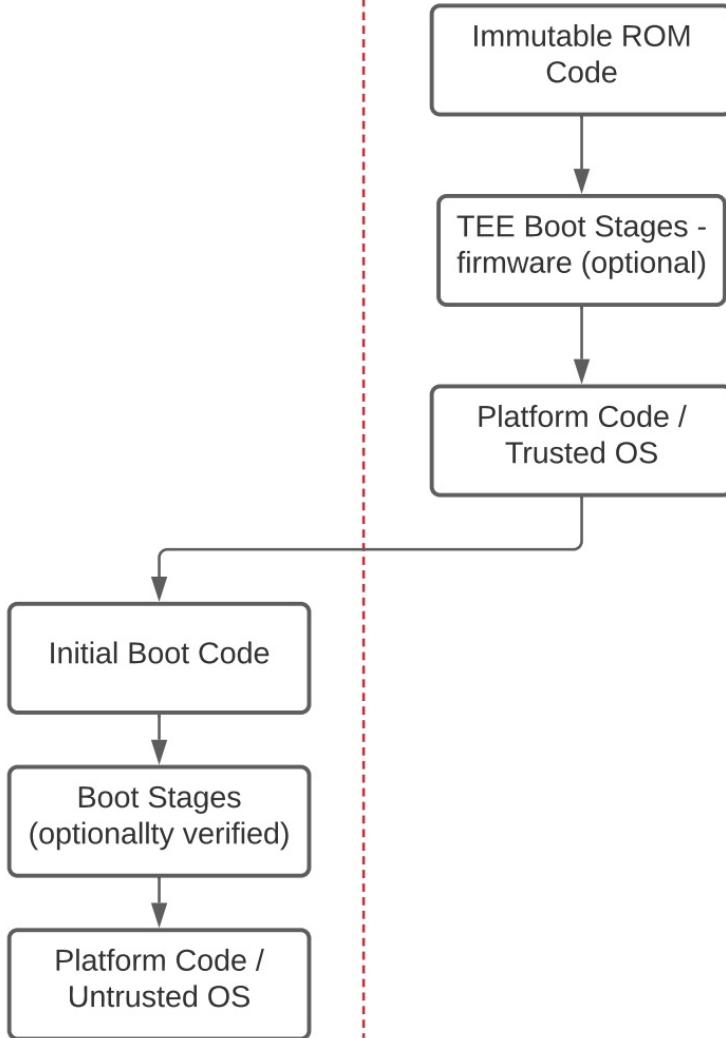
Os programadores de contentores seguros devem implementar monitores para realizar mudanças de contexto de volta para o mundo normal, garantindo segurança

As garantias de segurança funcionam sobre a armazém de que uma raiz de confiança verificou a integridade do sistema antes da execução - a inicialização segura

O SO confiável é inicializado antes do Kernel do mundo normal!

## Normal World

## Secure World



Bootloader & OP-TEE

OP-TEE

loads

returns

FIT Image

Device Tree

ROM

verifies



Bootloader

verifies



Kernel

verifies



Filesystem

As fases de verificação formam uma cadeia de confiança em que cada ligação corresponde a uma etapa, que se baseia na anterior para garantir integridade

VALORES IMUTÁVEIS → FIRMWARE → SO CONFIÁVEL → SO NORMAL

O SO confiável está sub-especificado, mas tem de ser inicializado

**OP-TEE:** projetado para correr em paralelo com um sistema operativo não seguro, interagindo com aplicações confiáveis via RPCs definidas

Objetivos: 1. Isolamento de Aplicações Confiáveis 2. Pequena Pegada 3. Portabilidade

Semelhanças a Intel SGX: execução; personalização; isolamento

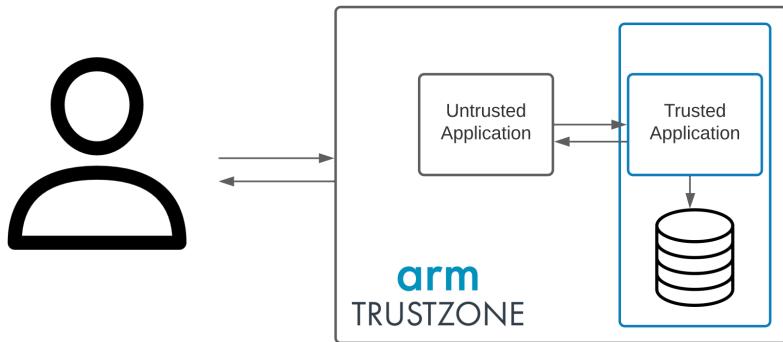
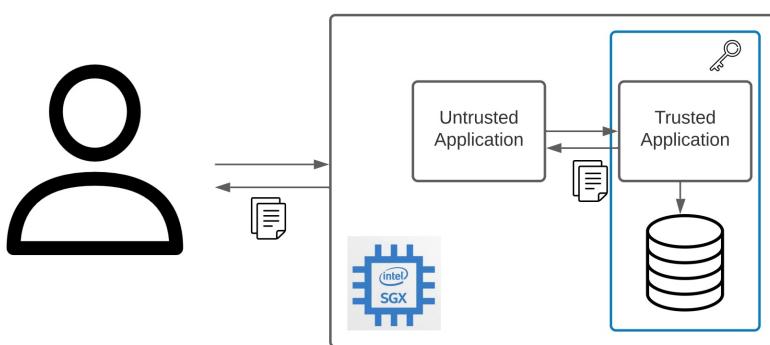
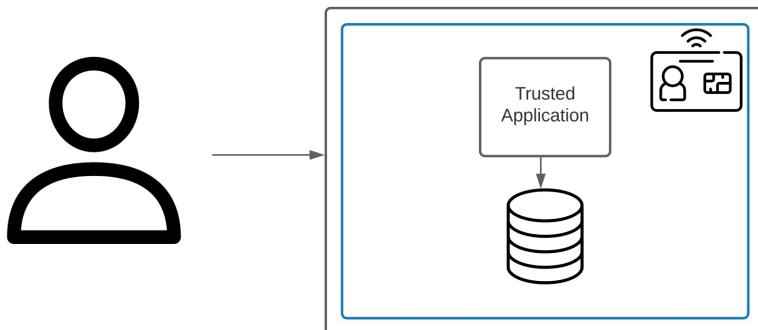
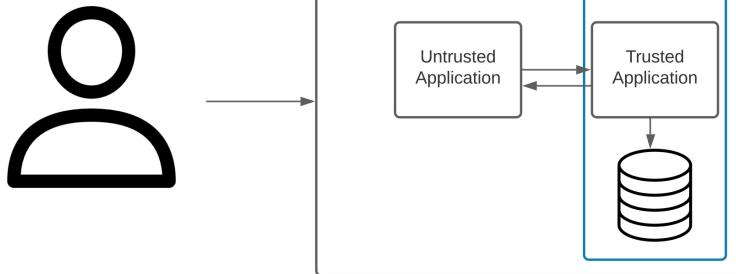
Diferenças de Intel SGX: arquitetura; a testação remota; mundos

**Ambiente de Execução Isolado:** ambiente isolado para a execução de código, capaz de executar código arbitrário protegido de leituras externas (confidencialidade) e de fraude externa (integridade)

Operações { 1. Compilar: construir aplicação para correr em hardware  
2. Carregar: carregar aplicação  
3. Executar: executar aplicação  
4. Verificar: verificar a veracidade do "output" produzido

Assunção: o hardware fornece as garantias de segurança descritas

Ataques { 1. Canais Colaterais  
2. Execução Especulativa  
3. Desenrolamento  
4. Implementação



# Ataques

Modelo Tradicional: ataque no canal de comunicação entre as partes

Dispositivo Criptográfico: dispositivo eletrônico que implementa um algoritmo criptográfico, armazena uma chave criptográfica e é capaz de realizar operações criptográficas usando essa chave — o adversário pode ser um utilizador legítimo com total acesso físico ao dispositivo num ambiente hostil

Um sistema é tão forte quanto o seu elo mais fraco!

Ataque | Ativo: manipular e concluir

Ativo: observar e inferir

Ataque | Invasivo: abre o pacote e contactar com o chip

Semi-Invasivo: abre o pacote, mas não contactar com o chip

Não Invasivo: Nm modificações

Criptoanálise: conta a matemática do algoritmo

Ataque Físico: não conta a matemática do algoritmo

Fugas por Canais Colaterais: libertam resultados intermitentes fracos

1. Passivo:

a. temporização

b. radiação eletrromagnética

c. som, temperatura, ...

2. Ativo: manipulação física da tecnologia

```
FUNCTION check (USER_PIN, CORRECT_PIN)
FOR i=1 TO PIN_LENGTH
  IF USER_PIN[i] != CORRECT_PIN[i]
    RETURN -1
  ENDFOR
RETURN 0
```

## MAIN FUNCTION

```

    ...
    IF check(...) == -1
        COUNTER++
    ELSE
        COUNTER = 0
    ...

```

## Ataques Invasivos:

- Passivo - Micro-Amostragem: amotrar o BUS com uma agulha e ler
- Ativo - Modificação do Circuito: conectar ou desconectar mecanismo de segurança

## Proteções

- |                 |   |
|-----------------|---|
| <u>Ativas</u>   | <ol style="list-style-type: none"> <li>1. Difícultar - proteger partes do chip</li> <li>2. Detectar - rastros, verificações, redundância e alarmes</li> <li>3. Reagir - interromper a computação</li> </ol> |
| <u>Passivas</u> | <ol style="list-style-type: none"> <li>1. Eliminar canais colaterais</li> <li>2. Esconder canais colaterais</li> <li>3. Utilizar a informação em fuga</li> </ol>  |

Cache de Memória: armazena cópias de memória recentemente acedida

Execução Especulativa: adivinhar caminhos prováveis do programa

↓

O processador executa instruções que não era suposto - as instruções incorretamente processadas não são descartadas **MAS** podem ter efeitos colaterais visíveis

Ataque do Ramo Condicional: combina cache de memória e execução especulativa

↳ restaurar todo o estado da cache quando a especulação falha?

↳ a execução especulativa pode ter efeitos colaterais para além da cache!

↳ instrução LFENCE que para a execução especulativa?

↳ como o fazer? manualmente? avisado! sempre? desempenho! **FALHA**

```

FUNCTION check (USER_PIN, CORRECT_PIN)
FOR i=1 TO PIN_LENGTH
    IF USER_PIN[i] != CORRECT_PIN[i]
        RETURN -1
    ENDIF
RETURN 0

```

## MAIN FUNCTION

```

    ...
    IF check(...) == -1
        COUNTER++
    ELSE
        COUNTER = 0
    ...

```

# Assunções e "Big Picture"

Âmbito de Confiança: ambiente onde a computação é segura - dispositivo criptográfico que desempenha um papel central em assegurar o comportamento seguro e confiável de uma aplicação **HARDWARE CONFIÁVEL**

A aplicação não pode estar inteiramente no hardware  
**MAS**

A aplicação aproveita as vantagens fornecidas pelo hardware

Hardware Confável: minimalistas

Aplicação: complexa e dependente do desempenho

1. O que estou a proteger contra? Modelo de Adversário
2. Que garantias o hardware dá? Assunções
3. A utilização é adequada? Poder do Adversário

Domínio: estrutura crítica em termos de segurança para garantir controlo de acessos  
Inclui, entidades confiáveis; regras; chaves secretas

Domínio | Confiança: estrutura para controlar quem pode gerir o domínio e pedir operações  
| Chaves: encriptadas para o HSM no domínio  
| Assinatura: por um HSM do domínio para garantir integridade

Confiança | Identificadores  
| Chaves Públicas  
| Regras de Quórum: conjunto de operadores que pode gerir a confiança  
| Hash dos Antecessores  
| Assinatura

## **Grupo 1 - Smart Cards**

### **1.1. Definições.**

- a) Uma das distinções chave entre os smart cards modernos e os cartões de fita magnética é a capacidade de armazenamento segura. Dê um exemplo de ataque que acontece trivialmente em cartões de fita magnética mas não em smart cards modernos, devido à ausência desta garantia.
- b) Dê dois exemplos de características essenciais associadas a smart cards, para além da capacidade para armazenamento seguro.

### **1.2. Autenticação.**

A realização de operações seguras em smart cards é habitualmente precedida por uma sequência de validações do cartão perante o leitor, e vice-versa.

- a) Explique a importância de autenticar o leitor de cartões (i.e. demonstrar que o leitor é genuíno), no contexto de realizar operações em smart cards.
- b) Dê um exemplo de uma aplicação vulnerável a ataques, caso o cartão não seja validado perante um leitor de cartões (i.e. demonstrar que o cartão é genuíno).

### **1.3. Comunicação e APDUs.**

- a) Explique o que significa descrever a comunicação entre smart cards e leitores como sendo *half-duplex*.
- b) Após inicialização e seleção do applet, suponha que envio o seguinte APDU para o cartão: 0x14, 0x0A, 0x01, 0x02, 0X01, 0X00. O seguinte APDU é recebido como resposta: 0x90, 0x00. Enumere as diferentes componentes desta comunicação – o que representa cada um destes bytes?
- c) Esta troca de APDUs está correta? Justifique.

### **1.4. Memória e Armazenamento Confiável.**

- a) Estudamos quatro tipos diferentes de memória presentes em smart cards: RAM, ROM, EEPROM e Flash. Associe as seguintes características a estes diferentes tipos de memória (responda na folha de exame).

Nota: Algumas características são comuns a várias memórias. Nesse caso, indique apenas uma.

1. Persistente e mutável
2. Tempo de vida (nº de acessos) limitado
3. Utilizada para armazenamento de programas fixos
4. Informação não é preservada sem energia
5. Pode ser acedida um número ilimitado de vezes

## Grupo 2 - HSMs e TPMs

### 2.1. Inicialização Confiável.

- a) Os protocolos estudados para inicialização confiável na TPM pressupõem a existência de um valor sucinto (uma hash) para validar integridade do código de inicialização da plataforma. Porque é que conseguimos fazer apenas através de uma hash, em alternativa a ter o código todo dentro da TPM?
- b) Um componente essencial do TPM é o seu gerador de números/valores aleatórios. Que ataques podem surgir se um TPM tiver que obter a sua aleatoriedade através do sistema operativo?

### 2.2. Hardware Security Modules.

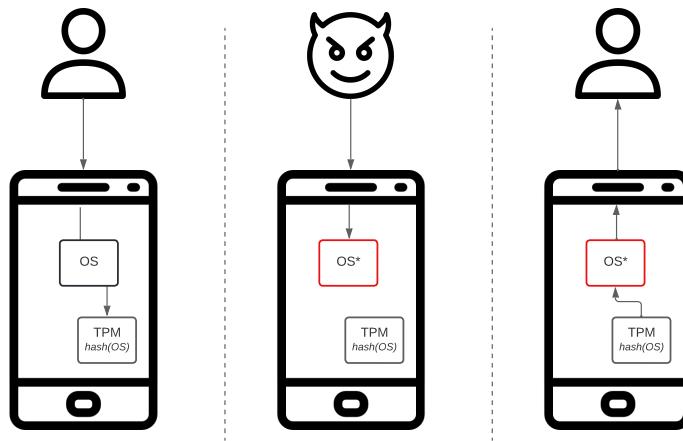
- a) Dê dois exemplos de benefícios de utilizar APIs como a Cryptoki para aceder a HSMs, ao invés de permitirmos que cada tecnologia de HSM defina a sua própria API.
- b) Indique se a seguinte frase está correta, justificando.

Tendo em conta que HSMs dão garantias elevadas de segurança para operações criptográficas, estas devem também ser consideradas para outro tipo de aplicações. Por exemplo, eu posso usar HSMs para armazenar e fazer cálculos sobre a informação bancária da minha empresa.

### 2.3. Assunções

Considere um sistema desenhado para proteger a integridade de um sistema operativo para um dispositivo móvel. A sua utilização divide-se em 3 etapas (na figura, da esquerda para a direita):

- Numa primeira etapa, existe um setup confiável, onde o utilizador inicializa um sistema operativo confiável ( $OS$ ), e armazena o seu measurement no TPM ( $\text{hash}(OS)$ ).
- Numa segunda etapa, o sistema fica vulnerável a ataques da parte do adversário. Este pode corromper o sistema operativo, alterando o seu código ( $OS^*$ ). Porém, este não consegue alterar os valores internos da TPM.
- Finalmente, o utilizador volta a ter controlo do seu sistema. Pode chamar a TPM para obter o measurement, de forma a verificar a integridade do sistema operativo.



- a) Que garantias é que o TPM está a dar ao utilizador, no que toca à integridade do seu sistema operativo?.
- b) Considere, agora, que o adversário também consegue corromper a memória do TPM, podendo alterar os valores armazenados. Dê um exemplo de ataque que subverte o sistema para a instalação de um sistema operativo corrupto.
- c) Se quisermos adaptar este sistema para uma arquitetura x86, conseguimos ter as mesmas garantias se a nossa solução de hardware seguro for Intel SGX? Justifique.

## Grupo 3 - Intel SGX e ARM TrustZone

### 3.1. Ambientes de Execução Isolados

a) Migrar aplicações para serem executadas remotamente é uma abordagem apelativa do ponto de vista de preço, performance e escalabilidade. Desta forma, não temos que nos preocupar com adquirir e gerir a infraestrutura física, e podemos pagar apenas a computação que as nossas aplicações precisam. Indique **um problema de segurança** associado a este modelo, onde os dados são preservados remotamente, e descreva **como este pode ser resolvido** utilizando hardware confiável.

b) Indique se as seguintes frases são verdadeiras ou falsas (responda na folha de exame).

1. Intel SGX e ARM TrustZone são importantes, pois não existe forma de fazer computação sobre dados protegidos usando apenas software.
2. É possível programar as funcionalidades criptográficas de um HSM num enclave de Intel SGX.
3. Código a correr no mundo seguro do ARM TrustZone é resistente a ataques de timing.
4. Executar programas em ambientes isolados é uma medida que me protege contra potenciais bugs dos mesmos programas.
5. É possível converter qualquer programa desenhado para ARM TrustZone para ser executado usando uma TPM.

### 3.2. Intel SGX

Considere a seguinte imagem, que descreve um passo do mecanismo de atestação intra-plataforma do Intel SGX. O enclave A deseja produzir uma atestação de uma mensagem  $m$  para o enclave B. Como resposta a este pedido, o CPU devolve um MAC da mensagem  $m$  e do código do enclave A, criado utilizando a chave associada ao enclave B.



- a) Podemos redesenhar o Intel SGX para devolver a chave do enclave B ao enclave A, de modo a que possa ser o próprio enclave a produzir o MAC em questão? Justifique.
- b) Porque é que não podemos simplesmente utilizar o mecanismo existente de atestação intra-plataforma para atestar valores para um cliente que esteja a comunicar externamente com a plataforma (inter-plataforma)?

### 3.3. ARM TrustZone.

- a) No mundo seguro do ARM TrustZone, só pode ser executado código assinado por uma entidade de confiança. Indique uma desvantagem associada a esta restrição.
- b) Qual é a vantagem de equipar sistemas de ARM TrustZone com dois sistemas operativos: um sistema operativo na zona de memória confiável (Trusted OS), e um sistema operativo na memória normal (Untrusted OS)?

## Grupo 4 - Ataques e Contramedidas

### 4.1. Modelo Embebido

- a) Porque é que o adversário contra hardware confiável tende a ser consideravelmente mais poderoso que um adversário contra um sistema clássico?
- b) Explique o que são ataques físicos, e de que forma se comportam de forma diferente da criptanálise.

### 4.2. Vulnerabilidades e Contramedidas

Considere uma função para deteção de vulnerabilidades, que se comporta da seguinte forma:

- Recebe uma lista de especificações de sistema (em formato de array)
  - Num loop, itera sobre as várias entradas de uma base de dados de vulnerabilidades (BD):
    - Se detetar que uma (ou mais) especificações recebidas estão associados a uma vulnerabilidade na BD, termina e retorna uma mensagem de "vulnerabilidade detetada".
    - Caso contrário, segue para a próxima entrada na base de dados.
  - Se o ciclo terminar, nenhuma vulnerabilidade foi associada, e o programa termina com uma mensagem de "sucesso".
- a) Assuma agora que o tamanho da base de dados de vulnerabilidades é público. Explique como é possível, através de um *timing attack*, distinguir (com algum nível de confiança) se o resultado desta operação foi "vulnerabilidade detetada", ou "sucesso".
  - b) Explique como ataques de manipulação de energia podem ser usados para atacar um sistema de hardware confiável.
  - c) Desativar a execução especulativa é uma forma natural e eficaz de evitar ataques através deste mecanismo. Porque é que esta medida não é tomada em todos os sistemas para os proteger contra esta fragilidade?
  - d) Uma disciplina conhecida para prevenir ataques de timing é a de constant-time. Descreva que garantias temos de um programa que corre em tempo constante.
  - e) Descreva os benefícios principais de exigir certificação de hardware, quando queremos desenvolver uma aplicação que visa utilizar hardware confiável como âncora de confiança.