Week 2 - Group 3

1

Os *Java Cards* podem ser uma tecnologia adequada para um sistema de identificação em bibliotecas porque oferecem segurança, portabilidade, independência de *hardware* e baixo custo de produção.

- **Segurança:** Os *Java Cards* possuem mecanismos criptográficos que garantem que os dados armazenados, como informações pessoais e histórico de empréstimos, estejam protegidos contra acessos não autorizados. Além disso, permitem autenticação segura baseada em PKI, garantindo que apenas o utilizador legítimo possa aceder aos seus dados. São, também, resistentes a violações físicas.
- **Portabilidade:** Por serem cartões de pequenas dimensões, semelhantes a cartões bancários ou de identificação, os *Java Cards* são fáceis de transportar, permitindo que os utilizadores os tenham sempre consigo. Além disso, podem ser utilizados em sistemas de leitura contactless ou com chip.
- **Independência de *Hardware*:** Os *Java Cards* são compatíveis com diferentes dispositivos e leitores, uma vez que seguem um padrão universal baseado em *Java*. Isto significa que podem ser utilizados em qualquer biblioteca equipada com um sistema de leitura compatível, sem necessidade de *hardware* específico.
- **Baixo Custo:** Devido à sua independência de *hardware*, os *Java Cards* tornam-se uma alternativa de baixo custo porque não exigem um sistema operativo complexo para funcionar. Como dito anteriormente, podem ser utilizados em diversos dispositivos sem necessidade de *hardware* exclusivo, reduzindo os custos de implementação e manutenção.

  Outro fator que contribui para o baixo custo é a possibilidade de atualização do *software*, sem ser necessário substituir o *hardware*. Isto significa que novas funcionalidades podem ser adicionadas ao sistema da biblioteca sem custos adicionais com novos cartões ou dispositivos físicos.

Pode ser relevante mencionar que, em cada *Java Card* podem existir vários *applets*. Esta característica permite que os dados estejam isolados e protegidos caso um dos *applets* seja comprometido. Também permite que diferentes serviços possam ser disponibilizados no mesmo cartão, o que pode ser útil para a biblioteca.

2

O **Trusted Platform Module (TPM)** e o **Hardware Security Module (HSM)** são dispositivos de segurança baseados em *hardware* que oferecem proteção para operações criptográficas e armazenamento seguro de chaves, mas diferem no propósito, na implementação e na utilização. Ambos garantem a integridade dos dados e a resistência contra ataques. Criam e guardam chaves

de forma segura e realizam funções como criptografia, assinatura digital e autenticação.

No entanto, o **TPM** é um *microchip* embutido em computadores, servidores e dispositivos IoT, projetado para garantir a integridade e a autenticidade do sistema. Protegem o processo de inicialização e guardam chaves criptográficas locais. O *setup* do **TPM** deve ser confiável.

Já o **HSM** tem vários tipos como *Network* HSMs (LAN *connection*), *Internal* HSMs (PCI), *Offline* HSM (USB) e *Cloud* HSMs (PW *authentication*). É utilizado, principalmente, em ambientes empresariais e na *cloud* para proteger, gerir e armazenar chaves criptográficas e garantir segurança em transações financeiras. Exigem conformidade com requisitos de segurança rigorosos como FIPS 140-2. Realizam operações criptográficas de alto desempenho como otimizações criptográficas e têm alta disponibilidade e capacidade para suportar várias operações simultaneamente.

Assim, o **TPM** é ideal para a segurança em dispositivos individuais, enquanto o **HSM** é essencial para proteção de dados críticos e gestão de chaves em ambientes empresariais e na *cloud*.

3

Porque é que Intel SGX ou ARM TrustZone são mais adequados do que Hardware Security Modules (HSMs)?

O hospital precisa de armazenar e processar dados clínicos sensíveis, o que exige segurança tanto na execução quanto no armazenamento dessas informações.

Intel SGX e **ARM TrustZone** permitem o processamento seguro de dados dentro de regiões isoladas da memória, protegendo as informações até mesmo contra ataques internos. Como os dados do hospital precisam de ser acedidos e processados frequentemente pelos funcionários, estas tecnologias são ideais, pois possibilitam a computação segura diretamente no processador, sem necessidade de transferências externas para zonas menos seguras que poderiam comprometer a segurança dos dados.

Por outro lado, os **HSMs** são idealizados principalmente para operações criptográficas e armazenamento seguro de chaves, mas não são otimizados para o processamento frequente dos dados. Além disso, os **HSMs** são dispositivos externos, o que pode introduzir latência e dificultar o fluxo de trabalho hospitalar devido à necessidade de acesso contínuo.

Outra vantagem do **SGX** e do **TrustZone** em relação aos **HSMs** é que essas tecnologias já vêm implementadas nos processadores, o que torna a solução mais escalável e económica. O uso de **HSMs** exigiria a aquisição de *hardware* adicional, o que poderia aumentar os custos e a complexidade da implementação.

Qual escolher? Intel SGX ou ARM TrustZone?

A escolha entre **Intel SGX** e **ARM TrustZone** depende de vários fatores, incluindo compatibilidade, segurança e facilidade de implementação.

- **Compatibilidade:** Em termos de compatibilidade, o **Intel SGX** é mais adequado para servidores e computadores x86, que é a arquitetura mais utilizada, enquanto o **ARM TrustZone** é mais indicado para dispositivos móveis, *tablets* e equipamentos IoT. Considerando que a infraestrutura hospitalar provavelmente utiliza servidores e *desktops*, o **Intel SGX** tende a ser a melhor escolha.
- **Segurança:** Em relação à segurança, o **SGX** oferece um nível de isolamento mais rigoroso, garantindo que nem mesmo o sistema operativo consiga aceder aos dados sensíveis armazenados. O **TrustZone**, por outro lado, cria dois ambientes distintos, Seguro e Não Seguro, mas não oferece um isolamento tão bom quanto o **SGX**. Mais uma vez, para um hospital, onde a proteção de dados clínicos é essencial, o **Intel SGX** pode ser a opção mais segura.
- **Implementação:** No que diz respeito à implementação, o **SGX** possui ampla documentação, ferramentas de suporte e simuladores, tornando o seu desenvolvimento mais acessível.

  Já o **ARM TrustZone** pode ser mais complexo de implementar, pois exige familiaridade com a arquitetura ARM, que pode ser menos comum em ambientes hospitalares.

Concluindo, dado o contexto, onde a maioria dos sistemas hospitalares opera em servidores baseados em x86 e há a necessidade de um isolamento mais forte dos dados clínicos, o **Intel SGX** é provavelmente a melhor escolha. No entanto, caso haja necessidade de suporte para dispositivos móveis e IoT, o **ARM TrustZone** pode ser considerado como uma alternativa viável para essas plataformas específicas.

4

A execução especulativa é uma técnica utilizada em processadores modernos para melhorar o seu desempenho. Consiste em executar instruções antes que as mesmas sejam necessárias. Esta abordagem permite reduzir o tempo de espera devido a decisões condicionais (*if-statements*), por exemplo.

No entanto, se a abordagem do processador, por exemplo se este fizer uma previsão, estiver incorreta, as instruções executadas especulativamente são descartadas, e o estado do sistema é restaurado para um ponto anterior. Embora este mecanismo geralmente traga benefícios significativos de desempenho, em alguns casos, a execução especulativa pode resultar em reduções do mesmo, especialmente quando existem falhas de previsão ou necessidade de invalidação de cálculos especulativos.

Para mitigar estes riscos, algumas arquiteturas modernas implementam mecanismos de proteção, como restrições na execução especulativa ou melhorias nas previsões.

Execução *Eager*

Na execução *eager* o processador executa todas as possíveis ramificações de um *if-statement* simultaneamente, sem aguardar a satisfação da condição. Assim que a condição é avaliada, os caminhos desnecessários são descartados, e o programa segue apenas com o resultado correto.

Esta abordagem elimina penalizações causadas por previsões erradas, pois o caminho correto já está pronto para ser utilizado. No entanto, apresenta um alto desperdício de recursos, pois o processador executa instruções que podem nunca ser utilizadas.

Vantagem: Evita penalizações por previsões erradas.

Desvantagem: Gasta muitos recursos computacionais e energia ao executar instruções desnecessárias.

Execução Preditiva

Ao contrário da execução *eager*, a execução preditiva consiste na previsão do caminho mais provável e na execução desse caminho, antecipadamente.

Se a previsão estiver correta, a execução ocorre de forma eficiente, sem desperdício de recursos. No entanto, se a previsão estiver errada, o processador descarta os resultados incorretos e executa o caminho correto, o que gera uma redução do desempenho devido à necessidade de reexecução.

Vantagem: Mais eficiente que a execução *eager*, pois utiliza menos recursos.

Desvantagem: Se a previsão estiver errada, ocorre uma redução do desempenho, pois o processador precisa de descartar a execução errada e refazer a correta.

5

a)

O ataque **Lucky13** afeta sistemas que utilizam os protolos **TLS** (Transport Layer Security) e **DTLS** com o modo de operação **CBC** (Cipher Block Chaining). Também pode ser considerado um *man-in-the-middle attack*.

Embora o ataque esteja associado a vulnerabilidade no **TLS** 1.2 e versões anteriores, o problema reside na utilização do na cifra de blocos CBC. Implementações que utilizam essa cifra podem estar vulneráveis ao ataque *Lucky13*, independetemente da versão.

Para mitigar a vulnerabilidade, é necessário deixar de usar cifras de blocos CBC e adotar cifras mais seguras, como as que utilizam **AEAD** (Authenticated

Encryption with Associated Data), disponíveis na versão 1.2 e obrigatórias na versão 1.3 do **TLS**, que já tem a proteção contra este ataque.

As implementações vulneráveis podem estar em servidores web (o tempo de resposta da verificação do *padding* pode dar informação sobre os dados), VPNs (obter comunicações parciais entre as partes envolvidas), clientes TLS (um servidor malicioso pode manipular as respostas), Smart Cards e HSMs (muitos são projetados para operarem em tempo real, o que torna o ataque mais preciso).

b)

O **Lucky13** é um *timing attack* que explora diferenças mínimas no tempo de processamento de mensagens cifradas em **TLS** com **CBC**.

A vulnerabilidade surge devido ao uso do esquema *MAC-then-Encrypt* no **TLS** com **CBC**, no qual o código de autenticação da mensagem (MAC) é calculado antes da encriptação. Durante a desencriptação, o servidor segue os seguintes passos:

1. Decifra a mensagem usando o modo **CBC**.
2. Verifica e remove o *padding* adicionado para alinhar o tamanho do bloco.
3. Calcula e verifica o MAC para garantir a integridade da mensagem.

Isto é um problema, pois diferentes tipos de *padding* e variações na verificação do MAC resultam em tempos de processamento ligeiramente diferentes. Estas variações permitem que um atacante envie mensagens manipuladas e meça o tempo de resposta do servidor, inferindo informações sobre o conteúdo cifrado.

c)

O ataque **Lucky13** é especialmente perigoso para dispositivos de *hardware* criptográfico, como *smart cards*, pelas seguintes razões:

- Dispositivos de *hardware*, como *smart cards*, possuem execuções altamente determinísticas, ou seja, realizam operações criptográficas com variações no tempo de resposta. Isto facilita ataques baseados em tempo de resposta, pois o atacante pode medir diferenças temporais de forma precisa e inferir informações sobre a mensagem cifrada.
- Os *smart cards* seguem padrões antigos e não implementam certas contramedidas como execução em tempo constante ou adição de aleatoriedade no tempo de resposta.
- Operam localmente, executando um pedido de cada vez e possuem interação direta com o atacante o que aumenta a precisão da medição dos tempos de resposta. Por exemplo, um terminal de leitura de *smart card* pode ser manipulado para enviar pedidos controlados e medir diretamente os tempos de resposta sem interferências da rede. Isso torna os ataques como o *Lucky13* ainda mais eficazes, já que elimina variáveis imprevisíveis presentes nos servidores ou na rede.