Sistema de Gestão de Bases de Dados para Aplicações Móveis apoiado por TrustZone

Pedro Ribeiro, Nuno Santos, e Nuno O. Duarte
{pedro.s.ribeiro, nuno.m.santos, nuno.duarte}@tecnico.ulisboa.pt

INESC-ID / Instituto Superior Técnico, Universidade de Lisboa

Resumo Nos últimos anos, a tecnologia de ARM TrustZone tem sido muito usada para melhorar a segurança de dispositivos móveis ao permitir a criação de Ambientes de Execução Segura (TEE). No entanto, soluções existentes de TEE tendem a não conseguir atingir um compromisso entre segurança e funcionalidade: expõem uma larga superfície de ataque em prol de carregamento dinâmico de código dentro da TEE, ou dependem da implementação estática de serviços seguros dentro do TEE, o que é mais pesado e sujeito a falhas de manutenção. Neste artigo apresentamos o DBStore, um sistema de gestão de bases de dados apoiado por TrustZone que expõe uma interface reduzida e pretende servir uma larga gama de aplicações. Através do DBStore, as aplicações podem criar bases de dados dentro do TEE, o que garante confidencialidade e proteção de integridade das mesmas ainda que o SO do dispositivo móvel tenha sido comprometido. Apresentamos um caso de estudo do DBStore para bilhética móvel segura baseada em Host Card Emulation (HCE).

1 Introdução

Ambientes de Execução Segura (TEE) têm tido um papel cada vez mais importante em segurança móvel. Um TEE é um ambiente de execução isolado que fornece confidencialidade e proteção de integridade a código e dados, mesmo na presença de um atacante com a capacidade de comprometer o SO (p.e., um rootkit). No mundo móvel, um TEE tende a ser criado por hardware especial disponível em processadores baseados em ARM: ARM TrustZone. Exemplos de sistemas comerciais implementados usando a tecnologia de TrustZone para proteção de dados e código dentro do TEE incluem o Android Keystore [5] e Samsung Knox [14], ambos apoiados por hardware.

Tanto a nível académico como da indústria tem havido um esforço para o desenvolvimento de soluções TEE que possam ser funcionalmente ricas e seguras. Atingir estes dois objetivos é, no entanto, desafiante. Uma linha de trabalho pretende desenvolver serviços seguros que residam dentro do TEE [9,16]. No entanto, esta abordagem requer alterações ao firmware para qualquer novo serviço implementado, o que desincentiva uma adoção mais geral por parte de fabricantes de dispositivos móveis. No outro espetro, encontramos sistemas de runtime TEE que permitem o carregamento de código aplicacional para que seja executado dentro do TEE [14, 15]. No entanto, mesmo que o código aplicacional seja

colocado dentro de sandboxes independentes residentes no TEE, a superfície de ataque do sistema aumenta.

O nosso objetivo é encontrar uma solução que permita às aplicações móveis tirarem proveito dos benefícios dos TEEs sem (i) permitir carregamento dinâmico de código e execução de código de aplicações de terceiros dentro do TEE, e (ii) requerer alterações frequentes ao firmware dos dispositivos móveis.

Este artigo persegue este objetivo e apresenta a conceção e implementação do DBStore, um sistema de gestão de bases de dados, apoiado por TrustZone, para aplicações móveis. O DBStore fornece uma interface SQL para a criação de bases de dados num TEE seguro. Uma API simples mascara os protocolos de segurança desenhados para defender contra ataques de retrocesso e de repetição, e proporcionar proteção de confidencialidade e integridade para tanto bases de dados como comandos SQL, incluindo inputs e outputs. No nosso protótipo atual, adotamos o SQLite como nosso motor SQL a correr dentro do TEE. O DBStore foi cuidadosamente concebido para garantir acesso um remoto seguro às bases de dados. Ao tirar partido do TrustZone para efeitos de isolamento, o DBStore pode proteger as bases de dados das aplicações mesmo na presença de um poderoso atacante que consegue controlar o sistema operativo local. Com a nossa solução, embora aplicações móveis não possam executar código arbitrário dentro do TEE, têm a possibilidade de gerir os seus dados de forma flexível e segura mediante a emissão de consultas sofisticadas às bases de dados.

Para demonstrar o nosso sistema, apresentamos um caso de estudo no qual usamos o DBStore na segurança de aplicações de bilhética móvel emergentes. Apresentamos o desenho de uma aplicação de bilhética móvel existente baseada em Host Card Emulation (HCE) e que usa o Android Keystore, e analisamos as suas vulnerabilidades. Depois, demonstramos como o DBStore pode ser utilizado para mitigar tais ataques com simples transações DBStore SQL. Embora nos tenhamos focado num só caso de uso, prevemos o DBStore a ser utilizado por um largo número de aplicações, p.e., e-health, pagamentos móveis, etc.

2 Motivação

Nesta secção, apresentamos alguma informação contextual bem como a apresentação dos objetivos e modelo de ameaças do sistema.

2.1 Contexto sobre ARM TrustZone e TEE

ARM TrustZone é uma extensão de segurança presente em processadores ARM que fornece isolamento ao nível de hardware entre dois domínios de execução: o *mundo não-seguro* (NW) e o *mundo seguro* (SW). Comparativamente ao mundo não-seguro, o mundo seguro tem mais privilégios, podendo aceder à memória, registos do CPU e periféricos do mundo não-seguro, mas não o inverso. O isolamento é aplicado ao verificar o estado do CPU através de um bit NS, e ao particionar o espaço de endereçamento de memória em regiões seguras e não-seguras. Interações entre mundos requerem uma operação de mudança de mundo ativada

por uma instrução privilegiada chamada SMC. O binário do software do SW deve fazer parte do firmware do dispositivo móvel e assinado pelo seu fabricante. Quando o dispositivo é ligado, o processador entra no SW, salta para o processo de inicialização do software do SW, cuja assinatura é validada, e depois salta para o NW e entrega o controlo total ao bootloader do SO.

TrustZone é usado por milhões de dispositivos móveis baseados em ARM para a criação de Ambientes de Execução Seguros (TEE). Um TEE consiste num ambiente de isolamento no qual aplicações seguras podem ser executadas sem a interferência do SO local (inseguro). Serviços TEE expõem apenas uma interface de chamada de função ao mundo não-seguro e implementam a aplicação específica. Apesar dos benefícios de segurança da arquitetura de Serviço TEE, pare se poder implementar uma nova aplicação segura dentro do TEE, os dispositivos deverão ser atualizados com um novo firmware contendo o binário da aplicação. No entanto, estas atualizações tendem a ser propícias a erros e trabalhosas para os utilizadores. Uma alternativa seria utilizar um Kernel TEE [2, 3] que permite o carregamento dinâmico de código, mas resulta numa superfície de ataque mais alargada. Pretendemos assim investigar se existe um serviço seguro de uso geral que possa ser usado por uma ampla gama de aplicações sem necessitar de executar código específico da aplicação no mundo seguro.

2.2 Objetivos e Modelo de Ameaças

Para responder a esta questão, propomos um novo serviço seguro – DBStore – que tem como objetivo manter e proteger bases de dados de aplicações dentro de um TEE apoiado por TrustZone. Considerando que bases de dados são comummente utilizadas, diversas aplicações podem beneficiar do DBStore.

Mais especificamente, pretendemos servir uma gama particular de aplicações nas quais um cliente remoto pode necessitar de aceder e manipular dados aplicacionais guardados localmente e, ao mesmo tempo, requerer um alto nível de proteção, tais como aplicações de bilhética móvel segura, pagamentos móveis e e-health. Consideramos aplicações distribuídas que consistem em duas partes: um *lado cliente* que é mantido sob controle do operador da aplicação, e um *lado móvel* que corre nos dispositivos dos utilizadores. A aplicação móvel pode requisitar espaço de armazenamento no DBStore para guardar bases de dados e deve interagir com as mesmas usando comandos SQL tradicionais.

O DBStore pretende dar os operadores de aplicações fortes garantias de segurança sobre a integridade e confidencialidade das bases de dados mantidas nos dispositivos dos utilizadores. Deve evitar que um adversário consiga ler ou alterar não só os conteúdos dos registos das bases de dados, mas também os inputs/outputs das transações SQL emitidas pelos clientes. O DBStore deve defender contra ataques de repetição que permitam a reexecução de comandos SQL legítimos mas anteriormente executados, e proteger contra ataques de retrocessos, que consistem em reverter o estado da base de dados para um anterior.

Consideramos que o adversário pode intercetar, espiar e modificar mensagens trocadas entre cliente e DBStore, quer seja na rede, ou por controlar a aplicação móvel ou o SO. Tal controlo pode ser ganho, p.e., através de malware, ou root dos

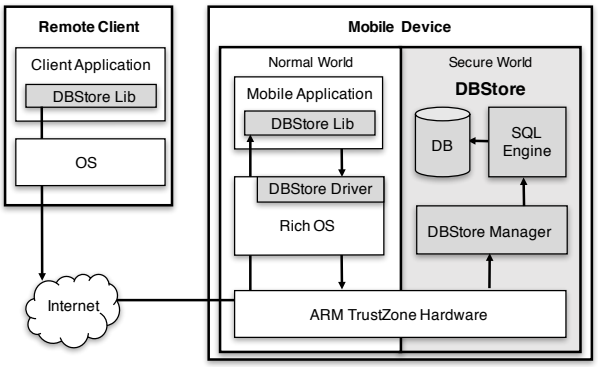

Figura 1: Arquitectura do DBStore: setas representam o fluxo de invocação de uma SRPC. O DBStore é composto por todas as subcomponentes no SW.

dispositivos. Assumimos, no entanto, que um adversário não pode fazer ataques físicos à plataforma móvel, que usa tecnologia de ARM TrustZone. Não tomamos em consideração ataques colaterais, como por exemplo baseados em caches.

3 Arquitetura do Sistema

A Figura 1 apresenta a arquitetura do DBStore. O mundo seguro é usado para implementar um TEE onde o DBStore reside; SO e aplicações móveis correm no mundo não-seguro. Como resultado aplicações móveis e SO não têm acesso direto ao estado do DBStore. A aplicação cliente interage com a aplicação cliente ao emitir comandos SQL às bases de dados hospedadas dentro do DBStore. As bases de dados são mantidas de modo seguro pelo DBStore, que tem acesso exclusivo ao estado das mesmas e é responsável pela execução dos comandos SQL submetidos pelos clientes. Corre dentro de um TEE em isolamento tanto do SO local como das aplicações móveis. O DBStore garante que os comandos SQL emitidos pelos clientes são seguramente encaminhados e executados pela base de dados pretendida e que os resultados são devolvidos ao cliente respetivo.

Comandos SQL são emitidos para uma base de dados residindo num DBStore remoto ao invocar *Chamadas de Procedimento Remoto SQL* (SRPC). Para este fim, uma API é oferecida pela biblioteca do DBStore. Uma SRPC consiste num ou mais comandos SQL, tais como "CREATE TABLE", "INSERT INTO", e "SELECT", e são executados como uma única transação. Um SRPC pode ser parametrizado pelos inputs fornecidos pelo aplicação cliente, e pode opcionalmente retornar um output à aplicação cliente. A biblioteca do DBStore e o gestor do DBStore implementam protocolos de segurança que envolvem a trocar de informação na forma de Unidades de Dados de Protocolo (PDUs) específicas. Uma SRPC é traduzida em PDUs que são enviadas pela rede até à aplicação mobile que os remete para o DBStore através do driver. O driver efetua a mudança de contexto para o SW. O DBStore, então, trata as PDUs, envia os comandos SRPC ao motor de SQL, e retorna os resultados até ao cliente. É de realçar que

Protocolo de Inicialização	
(M_1)	C $\rightarrow$ S: $\langle AppId, n_1 \rangle_{AK^-}$
(M_2)	S $\rightarrow$ C: $\langle n_1, \{SK\}_{AK^+} \rangle_{TK^-}$, C_{TK}
Protocolo de Invocação de SRPC	
(M_3)	C $\rightarrow$ S: $m = \{n_2, SrpcReq\}_{K_i}$, hmac($m$,$K_i$)
($M_{4.a}$)	S $\rightarrow$ C: $m = \{Ok, \{n_2, SrpcRes\}_{K_i}\}$, hmac($m$,$K_i$)
($M_{4.b}$)	S $\rightarrow$ C: n_2, *Fail*
Protocolo de Resincronização	
(M_5)	C $\rightarrow$ S: n_3, *Resync*, hmac(SK)
(M_6)	S $\rightarrow$ C: n_3, $\{i, salt\}_{SK}$, hmac(SK)

Tabela 1: Protocolos de Segurança DBStore: C = cliente remoto, S = DBStore.

o DBStore restringe o acesso das aplicações às suas bases de dados, pelo que uma aplicação maliciosa não poderá alavancar o sistema para extrair dados de outras aplicações. De seguida, apresentamos mais detalhes do sistema.

3.1 Protocolo de Inicialização

Antes de colocar bases de dados num DBStore remoto, um cliente tem que primeiro verificar que o par remoto corre uma versão legítima do DBStore dentro de um dispositivo com TrustZone disponível. Analogamente, o DBStore deve ser capaz de autenticar o cliente e implementar uma política de autorização que impede outras aplicações de acederem às bases de dados do cliente remoto. Para obter estas garantias, o cliente e o DBStore remoto devem primeiro realizar um protocolo de inicialização ao trocar duas PDUs: M_1 e M_2 (ver Tabela 1).

Para provar que o serviço DBStore é autêntico e que reside num ambiente TrustZone válido, a primeira mensagem (M_1) contém um desafio (nonce n_1) que o DBStore deve assinar com uma chave de atestação (TK) válida. O nonce é gerado pelo cliente. A chave de atestação é um par criptográfico assimétrico incorporado no firmware do dispositivo pelo fabricante. A chave privada TK^- é acessível apenas no SW e sob a condição do boot seguro ser bem sucedido. Se o DBStore é capaz de assinar n_1 com tal chave, significa que o DBStore não foi alterado e que corre dentro do SW. O DBStore deve então assinar n_1 com TK^- e enviar a assinatura juntamente com um certificado (C_TK) que contém a chave pública (TK^+) necessária para verificar a assinatura. Este certificado é emitido pelo fabricante do dispositivo. Assim, ao verificar a assinatura através de um chave de atestação devidamente certificada, o cliente pode validar a autenticidade do DBStore remoto.

Para autenticar o cliente remoto e restringir acesso às bases de dados locais por aplicações desconhecidas, M_1 contém o nonce n_1, um ID de aplicação e ambos são assinados com a parte privada AK^- do *par de chaves da aplicação* (AK). Este par de chaves deve ser gerado pelo operador da aplicação e a parte privada deve ser mantida secreta pelo lado cliente. A parte pública AK^+ está incluída no pacote da aplicação móvel. Após receber M_1, o DBStore pode validar a assinatura ao obter a chave pública AK^+ da aplicação móvel e assim

autenticando o cliente. Para assegurar que futuras interações se mantém mutuamente autenticadas, uma *chave de sessão* (SK) é trocada como um símbolo de autenticação mútua bem sucedida do cliente e atestação do serviço remoto. Isto é refletido pela mensagem M_2 que inclui a chave de sessão SK criada pelo DBStore e cifrada com AK^+. Este resultado é incluído na assinatura por TK^-. Deste modo, o DBStore consegue garantir a autenticidade do cliente, pois apenas um aplicação cliente legítima tem a chave privada AK^- que permitirá a decifra da chave de sessão. Por outro lado, ao verificar a assinatura de M_2, o cliente pode validar a autenticidade daquela chave de sessão e a frescura da chave de sessão. A chave de sessão será fundamental para proteger as SRPCs.

3.2 Chamadas de Procedimento Remoto SQL

Após o protocolo de inicialização concluir com sucesso, o cliente pode então invocar Chamadas de Procedimento Remoto SQL (SRPC) no DBStore. Como mostrado em baixo, cada SRPC gera uma mensagem para o DBStore (M_3), que irá então retornar uma de duas possíveis mensagens; $M_4.a$ se a mensagem de pedido foi bem formulada, ou $M_4.b$ caso contrário (ver Tabela 1). A mensagem de pedido SRPC M_3 inclui o campo *SrpcReq* que contém os comandos SQL e inputs para serem executados no DBStore remoto. O campo *SrpcRes* é incluído na mensagem $M_4.a$ e contém o output de execução para ser enviado para o cliente. Para proteção de confidencialidade, *SrpcReq* e *SrpcRes* são cifrados com a chave simétrica partilhada por cliente e DBStore. Para proteção da integridade da mensagem, a mesma chave é usada para gerar códigos HMAC para toda a mensagem. O nonce n_2 é gerado pelo cliente para o ajudar a detetar ataques de repetição de mensagens M_4 antigas.

Embora a chave simétrica partilhada entre ambos os pontos seja a chave simétrica SK, esta chave não é usada nas mensagens M_3 e $M_4.a$. Ao invés, o cliente cria uma nova chave simétrica K_i para cada SRPC invocada. K_i é gerada ao calcular o hash de SK concatenado com o valor do contador i, que é inicializado a zero e incrementado por cada pedido SRPC emitido pelo cliente. Para determinar K_i, o DBStore deve também manter um registo do valor local de i. Para calcular a chave de uma mensagem SRPC recebida, o DBStore apenas tem de juntar i a SK, e computar o hash do resultado, gerando K_i, que é então usado para decifrar a SRPC $\#i$ e cifrar a resposta SRPC correspondente. Após submeter a resposta ao cliente, o DBStore incrementa o contador i local.

Este mecanismo serve dois propósitos. Primeiro, impede ataques de repetição de pedidos SRPC antigos. (O contador i do DBStore é implementado usando um contador monotónico do TrustZone, impedindo assim que um adversário consiga decrementar i no dispositivo móvel.) Uma segunda vantagem deste mecanismo é evitar a reutilização da chave de sessão SK.

3.3 Proteção das Bases de Dados do DBStore

Para assegurar a segurança das bases de dados do DBStore durante inatividade, beneficiamos o armazenamento seguro alocado para o SW. Usamos mecanismos

TrustZone adicionais para prevenir ataques de retrocesso e proteger a confidencialidade dos dados. Em particular, usamos contadores monotónicos seguros para manter registo da versão mais atualizada das bases de dados do DBStore. Sempre que uma base de dados é atualizada, o seu número de versão é incrementado e escrito na base de dados, e o contador monotónico seguro é também incrementado. Se o dispositivo é reiniciado e a base de dados é substituída por uma versão antiga, o número de versão da base de dados recuperada e o valor atual do contador serão inconsistentes, fazendo com que o DBStore lance uma exceção. Para preservar confidencialidade, as bases de dados do DBStore (e números de versão associados) são cifrados com a chave de sessão que é então selada, i.e., cifrada com a chave acessível se o boot seguro é bem sucedido.

4 Implementação do Protótipo

Implementámos um protótipo inicial do DBStore para a Freescale NXP i.MX53 Quick Start Board. O DBstore corre dentro do SW e consiste em várias componentes. Primeiro, inclui um pequeno kernel responsável por gestão da memória, execução de threads, e operações de mudança de mundo. Para este fim, adotamos o kernel personalizado *base-hw* do Genode Framework [4], que consiste numa base de código de 20 KLOC. A lógica responsável pela gestão do DBStore é implementada em C. É também responsável pela gestão das chaves criptográficas simétricas, e fornece as funções criptográficas. Para manter uma base de código reduzida, usamos a implementação de AES-128 da biblioteca *mbed TLS* [1] como algoritmos de cifra simétrica. Também adaptamos o SHA-256 do *mbed TLS* para calcular hashes criptográficos.

Para a implementação do motor SQL, portámos o SQLite. Para persistência das bases de dados, implementámos um serviço de armazenamento seguro que media o acesso ao sistema de ficheiros residente no mundo seguro, através de uma API semelhante à usada por programas ao nível utilizador e bibliotecas tais como o SQLite. Mais especificamente, configurámos uma partição dedicada como segura no gestor de partições do Genode (*part_blk*). De modo a gerir ficheiros nesta partição, usamos o port de um rump kernel, *dde_rump*, do Genode, especialmente desenvolvido com um driver de sistema de ficheiros.

Para o NW, tirámos proveito do Monitor de Máquina Virtual (VMM) do Genode para correr um SO Android paravirtualizado, que potencializa um kernel Linux personalizado para o Genodo. No que diz respeito à interface entre NW e SW, criámos um módulo para o kernel Linux que oferece uma API que ativa o mecanismo de mudança de mundo necessário através de invocações de SMC.

5 Caso de Estudo: Bilhética Móvel Segura

Uma tendência em emergência em bilhética móvel é a transição de Elementos Seguros para HCE no SO Android, que permite que uma aplicação possa manter cartões contactless emulados por software e comunique com leitores NFC. Neste secção, estudamos uma aplicação existente de bilhética móvel HCE para

transportes públicos com o propósito de ilustrar como o DBStore pode prevenir algumas das suas falhas de segurança mais críticas.

5.1 Bilhética Móvel baseada em HCE para Transportes Públicos

A aplicação de bilhética móvel HCE em estudo, destina-se a transportes públicos, tais como comboios e autocarros. Numa utilização normal, um passageiro instala a aplicação, cria uma conta com nome de utilizador e palavra passe, e compra um cartão virtual com saldo para um certo número de viagens. Esta informação é descarregada e mantida dentro do cartão HCE emulado pela aplicação móvel. Para fazer uma viagem, o passageiro deve validar o cartão ao deslizar o dispositivo móvel à frente de um leitor NFC. O leitor comunica por NFC com a aplicação, que verifica se o cartão possui saldo suficiente. Se sim, decrementa o saldo disponível e informa o leitor que pode autorizar a viagem.

De um ponto de vista de segurança, a maior preocupação para o programador é prevenir viagens fraudulentas de passageiros sem saldo. A segurança da solução deve tomar em conta que os dispositivos móveis dos passageiros podem não ter uma ligação à rede disponível no momento da validação.

Compra de Cartões: Numa aplicação de bilhética móvel HCE existente, para comprar um cartão virtual, a aplicação deve abrir uma ligação HTTPS autenticada com um servidor de bilhética. Quando um bilhete é comprado, o servidor gera um número de série secreto para o novo cartão virtual, que é depois assinado criptograficamente e enviado para a aplicação móvel juntamente com o saldo do número de viagens associado. A assinatura é verificada usando a chave pública do servidor de bilhética, e os dados do cartão virtual (número de série e viagens) são escritos num ficheiro privado mantido pela aplicação. Este ficheiro é cifrado com uma chave K, mantida de modo seguro pelo Android Keystore.

Validações de Cartões: A validação de um cartão é iniciada por um *handshake* entre leitor de cartão e aplicação móvel. Neste *handshake*, o leitor de cartões é autenticado, uma chave de sessão é trocada, e o leitor de cartões envia um pedido de validação ao dispositivo. A aplicação recebe esta mensagem através do serviço HCE do Android, decifra o cartão virtual com a chave K (libertada pelo Android Keystore), e verifica se existe saldo suficiente. Se sim, envia ao leitor o número de série do cartão virtual cifrado com a chave de sessão e a aplicação atualiza o novo saldo no ficheiro, utilizando uma nova chave K para recifrar o seu conteúdo.

Recarregamento de Cartões: O recarregamento de um cartão virtual com saldo adicional combina mecanismos das operações anteriores. Primeiro, a aplicação móvel liga-se ao servidor de bilhética através de HTTPS. Quando o pagamento é concluído, a aplicação móvel descobre o saldo que foi adquirido e actualiza o ficheiro local de modo seguro, tal como no caso da validação.

5.2 Vulnerabilidades de Segurança Existentes

Apesar dos mecanismos de segurança da aplicação de bilhética móvel baseada em HCE descrita acima, existem vários ataques que podem ser efetuados por um

adversário que consiga controlar o sistema operativo do dispositivo móvel do passageiro. Descrevemos três ataques que estudámos considerando implementações do Android Keystore apoiadas por hardware.

Ataque 1 - Retrocesso dos ficheiros de Keystore e dados do cartão: Num sistema de bilhética móvel HCE, os dados do cartão são cifrados com uma chave K, que é cifrada com a parte pública de um par de chaves guardado num ficheiro do Android Keystore. Os ficheiros de par de chaves são cifrados com uma chave mestre que é mantida dentro do TEE e não pode ser retirada facilmente. Assim, um adversário com privilégios root consegue copiar todos estes ficheiros, e substitui-los com cópias antigas, de modo a trazer um cartão de volta a um estado anterior onde o saldo era positivo, o que permite que um atacante viaje com um único cartão desde que o seu número de série não seja bloqueado.

Ataque 2 - Roubo do par de chaves através de um aplicação maliciosa: Este ataque tira partido do facto do Android Keystore preservar os pares de chaves de uma dada aplicação em dois ficheiros cujo caminho inclui o ID do utilizador da aplicação. O dono do par de chaves é identificado pelo mesmo ID. Primeiro, o adversário cria uma aplicação "RoubaChaves" com o objetivo de retirar o par de chaves da aplicação móvel do ficheiro do Keystore e escrevê-lo noutro lugar, decifrado; com esse par de chaves, K pode ser obtido e, desse modo, os dados em bruto dos cartões. Desse modo, basta alterar o ID de Utilizador dos ficheiros gerados para a aplicação HCE de bilhética móvel para corresponderem ao ID do "RoubaChaves". O Android Keystore vai ser enganado e pensar que a aplicação "RoubaChaves" é dona do par de chaves e irá decifra-lo usando a chave mestre.

Ataque 3 - Engenharia reversa na aplicação móvel: Um atacante pode modificar a aplicação móvel para impedir decremento de saldo em cada validação. Tendo em conta que os leitores de cartões não têm modo de autenticar a aplicação móvel, não conseguiriam distinguir a aplicação correta duma modificada.

5.3 Ultrapassar as Vulnerabilidades com o DBStore

Pada defender contra os ataques expostos em cima, estado dos cartões virtuais será alocado dentro de uma base de dados do DBStore, existente nos dispositivos dos passageiros. Neste caso, existem dois clientes DBStore: o servidor de bilhética e os leitores de validações. Um par de chaves (ver Secção 3.1) deve ser gerado: a parte pública está incluída no pacote da aplicação móvel e a privada é mantida secretamente por servidor de bilhética e leitores. Assim, a versão revista da aplicação móvel de bilhética funciona do seguinte modo.

Após um novo utilizador criar uma conta, o servidor de bilhética realiza um protocolo de inicialização com o dispositivo móvel. De seguida, o servidor de bilhética envia a primeira SRPC com o objetivo de inicializar o cartão virtual na base de dados dentro do DBStore. A base de dados contém um registo dos cartões. Cada entrada tem um tipo de cartão, um número de série e saldo atual. Para compras e recarregamentos, o servidor de bilhética apenas precisa de emitir chamadas SRPC para inserir e atualizar a base de dados, respetivamente.

Para a operação de validação, o leitor apenas necessita de efetuar uma SRPC. Esta transação simplesmente emite um comando SQL para decrementar o saldo

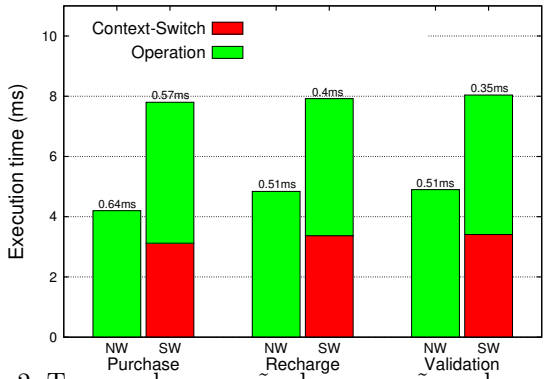

Figura 2: Tempos de execução das operações sobre o cartão.

atual do cartão virtual se for positivo. Caso contrário, o valor é colocado a -1, indicando que o saldo foi gasto. O segundo comando SQL lê o crédito atual juntamente com o número de série para que o cliente – i.e., o leitor – possa determinar se a passagem pode ser autorizada ou não: -1 significa que não pode. Visto que o DBStore protege contra ataques de retrocesso, o Ataque 1 é mitigado. O Ataque 2 falha pois o DBStore mantém o material criptográfico relevante guardado dentro de um TEE apoiado por TrustZone. Finalmente, o Ataque 3 não é possível pois, através do protocolo de inicialização, todas as SRPC são executadas dentro do DBStore, fora do alcance do atacante.

6 Avaliação de Desempenho

Para estudar o desempenho do DBStore, implementámos um protótipo da aplicação original de bilhética móvel HCE para a nossa plataforma, e medimos o tempo de execução das três principais operações. Realizámos medidas comparativas em ambos os mundos. Os números reportados excluem custos de comunicação pela rede. A plataforma de testes consiste numa i.MX53 Quick Start Board, com 1 GHz ARM Cortex-A8 e 1 GB de memória DDR3 RAM. Nesta placa específica, o boot seguro não foi implementado. Para cada experiência, fizemos 50 testes; reportamos a média e o desvio padrão.

A Figura 2 mostra os tempos de execução tanto no NW como no SW para as operações testadas: compra, recarregamento e validação. Os desvios padrão estão colocados em cima de cada barra. O tempo de execução é dividido em duas partes: "operação" refere-se à lógica base dos três casos, que é comum no mundo não-seguro e seguro; "mudança de contexto" refere-se à lógica associada com a mudança de contexto do mundo e existe apenas no SW.

Os resultados mostram que compras são mais rápidas visto que consistem apenas numa inserção na tabela de bilhetes. O recarregamento é mais lento pois envolve uma operação de escrita na tabela para atualizar o saldo. Finalmente, a validação é a mais lenta, pois necessita primeiro de ler a tabela para obter o saldo

e validar a operação, e só depois pode escrever o novo saldo atualizado. O SW tem uma penalidade consistente de cerca de 3ms associado com a mudança de contexto. De qualquer modo, tendo em consideração os benefícios de segurança que esta solução traz, argumentamos que esta penalidade causa pouco impacto na usabilidade de bilhética móvel.

7 Trabalho Relacionado

Existe trabalho significativo realizado na área da proteção de dados de utilizadores em dispositivos móveis. Aquifer [11] aplica controlo do workflow das aplicações para prevenir divulgação acidental de informação. TaintDroid [6] aproveita o controlo do fluxo de informação para rastrear os movimentos de dados de utilizador no sistema operativo. Sistemas para Gestão de Direitos Digitais (DRM) [12] também protegem dados no SO. Enquanto estes sistemas fornecem mecanismos eficazes para proteger dados em dispositivos móveis, a sua grande desvantagem é a dependência na integridade do sistema operativo. Algum trabalho existente foca-se na segurança do Android Keystore. Sabt et al. [13] provou formalmente a falha do esquema de cifra do keystore em fornecer garantias de integridade, e explora esta falha para definir um ataque de falsificação para quebrar a sua segurança. Dada a importância do Android Keystore em proteger o material criptográfico de aplicações, nós propomos um serviço análogo baseado em TrustZone para a proteção de bases de dados sensíveis.

No que diz respeito ao ARM TrustZone, investigação tem sido prolífica, o que resultou em várias contribuições para a segurança móvel baseada em TEEs. Alguns sistemas implementam serviços de segurança específicos que residem no SW, tais como autenticação segura [9], senhas de uso único [16], e canais de I/O seguros [8]. Outra classe de sistemas fornece suporte em runtime para a execução de código aplicacional instanciado dentro de sandboxes hospedadas pelo SW. Exemplos de tais sistemas incluem o Trusted Language Runtime (TLR) [15]. Em contraste, o DBStore não permite a execução de código arbitrário no SW, reduzindo assim a superfície de ataque, enquanto que, ao mesmo tempo, oferece uma interface SQL rica para armazenamento seguro de bases de dados.

Relacionado com bilhética móvel, o use de TEE apoiado por TrustZone para proteção dos dados de bilhetes surgiu em trabalhos anteriores. Num caso, a ideia foi sugerida num artigo de opinião de alto nível [7] sem nenhuma implementação. Noutros casos, a componente de software da aplicação de bilhética móvel deve correr dentro de uma sandbox TEE, a qual é suportada por um pequeno kernel seguro [17] ou por um ambiente HCE emulado por TEE [10, 18]. No nosso trabalho, demonstramos uma abordagem alternativa para implementar bilhética móvel segura baseada em comandos SQL apoiados por TEE.

8 Conclusões

Este artigo apresenta DBStore, um sistema que permite que aplicações móveis criem e operem sobre bases de dados residentes dentro de um TEE apoiado

por TrustZone, de modo a preservar a confidencialidade e integridade dos dados contra um atacante que consiga controlar o SO. Para facilitar portabilidade, as aplicações interagem com o DBStore através de uma interface SQL padrão. Mostramos que o DBStore pode ser facilmente adotado para assegurar o bom funcionamento de aplicações HCE de bilhética móvel para transportes públicos.

Agradecimentos Agradecemos aos revisores anónimos pelos seus comentários. Este trabalho foi parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) enquadrado nos projectos UID/CEC/50021/2013 e SFRH/BSAB/135236/2017, e por COMPETE 2020 / Portugal 2020 / União Europeia pelo projecto Mobile Security Ticketing (#11388), apresentado pela Link Consulting Tecnologias de Informação SA.

Referências

1. mbed TLS, `https://tls.mbed.org/`
2. OP-TEE, `https://www.op-tee.org`
3. SierraTEE, `https://www.sierraware.com/open-source-ARM-TrustZone.html`
4. The Genode OS Framework, `http://genode.org`
5. Android: Android Key Store (2018), `https://developer.android.com/training/articles/keystore.html`
6. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In: Proc. of OSDI (2010)
7. Hussin, W.H.W., Coulton, P., Edwards, R.: Mobile ticketing system employing trustzone technology. In: Proc. of ICMB (2005)
8. Li, W., Ma, M., Han, J., Xia, Y., Zang, B., Chu, C.K., Li, T.: Building Trusted Path on Untrusted Device Drivers for Mobile Devices. In: Proc. of APSys (2014)
9. Liu, D., Cox, L.P.: VeriUI: Attested Login for Mobile Devices. In: Proc. of HotMobile (2014)
10. Merlo, A., Lorrai, L., Verderame, L.: Efficient Trusted Host-based Card Emulation on TEE-enabled Android Devices. In: Proc. of HPCS (2016)
11. Nadkarni, A., Enck, W.: Preventing accidental data disclosure in modern operating systems. In: Proc. of SIGSAC (2013)
12. OMA: Enabler Release Definition for DRM V2.0.1 (2008)
13. Sabt, M., Traoré, J.: Breaking Into the KeyStore: A Practical Forgery Attack Against Android KeyStore. In: Proc. of ESORICS (2016)
14. Samsung: White Paper: An Overview of Samsung KNOX (April 2013), `http://www.samsung.com/es/business-images/resource/white-paper/2014/02/Samsung_KNOX_whitepaper-0.pdf`
15. Santos, N., Raj, H., Saroiu, S., Wolman, A.: Using ARM Trustzone to Build a Trusted Language Runtime for Mobile Applications. In: Proc. of ASPLOS (2014)
16. Sun, H., Sun, K., Wang, Y., Jing, J.: TrustOTP: Transforming Smartphones into Secure One-Time Password Tokens. In: Proc. of CCS (2015)
17. Tamrakar, S., Ekberg, J.E.: Tapping and Tripping with NFC. In: Proc. of TRUST (2013)
18. Umar, A., Mayes, K.: Trusted Execution Environment and Host Card Emulation. In: Smart Cards, Tokens, Security and Applications, pp. 497–519. Springer (2017)