INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Mestrado em

Eng.ª de Segurança Informática

Segurança Informática em Arquiteturas ARM

(ARM TrustZone)

Geraldo Piçarra Oliveira

2024

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Mestrado Eng.ª de Segurança Informática

Segurança Informática em Arquiteturas ARM

(ARM TrustZone)

Elaborado por:

Geraldo Piçarra Oliveira

Orientado por:

Prof. Doutor José Jasnau Caeiro, IPBeja

Prof. Doutor João Carlos Martins, IPBeja

Dissertação de Mestrado
Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Beja

2024

Resumo

Segurança Informática em Arquiteturas ARM

(ARM TrustZone)

A quantidade de dados gerados cresce exponencialmente. A proliferação de dispositivos móveis e Internet das Coisas é um contributo importante para esse cenário. A segurança da informação apresenta, por isso, um papel preponderante no mundo actual. Muitas das soluções baseiam a sua segurança apenas na qualidade do *software* produzido. Os avanços tecnológicos, aliados a uma crescente criatividade, elevaram os níveis de sofisticação dos ataques aos sistemas de informação. A preocupação com a segurança deve estar presente desde as primeiras fases do desenvolvimento e no *core* do próprio sistema. A segurança do *hardware* torna-se por isso fundamental. A ARM, *main leader* na produção de processadores e SoC para dispositivos móveis, abraçou a segurança como factor crucial na concepção dos seus sistemas. A segurança deixa de estar apenas dependente da engenharia do *software* passando a sua implementação para os componentes físicos: o *hardware*. A tecnologia TrustZone permite o isolamento de dados ou processos cruciais numa zona de execução segura e isolada da restante aplicação e sistema operativo, inseguro por natureza. Foi desenvolvida uma aplicação, para uma plataforma IoT, como prova de conceito baseada na tecnologia TrustZone que permite a recolha de dados, seu processamento e arquivo utilizando processos seguros visando a obtenção de um elevado nível de segurança.

Palavras-chave: *Segurança, ARM TrustZone, IoT, OP-TEE, Raspberry Pi 3.*

Abstract

Computer Security in ARM Architectures

(ARM TrustZone)

The amount of data generated grows exponentially. The proliferation of mobile devices and the Internet of Things is an important contribution to this scenario. Therefore, information security plays a preponderant role in today's world. Many of the solutions base their security only on the quality of the software produced.

Technological advances, combined with growing creativity, have raised the levels of sophistication of attacks on information systems.

The concern with security must be present from the first stages of development and in the core of the system itself. Hardware security is therefore essential. ARM, the main leader in the production of processors and SoC for mobile devices, has embraced security as a crucial factor in the design of its systems. Security is no longer just dependent on software engineering, passing its implementation to the physical components: the hardware. TrustZone technology allows the isolation of crucial data or processes in a safe execution zone, isolated from the rest of the application and operating system, which is inherently insecure. An application was developed for an IoT platform, as a proof of concept, based on TrustZone technology, which allows data collection, processing and archiving using secure processes in order to obtain a high level of security.

Keywords: *Security, ARM TrustZone, IoT, OP-TEE, Raspberry Pi 3.*

Agradecimentos

Agradeço a todos os meus professores do curso de Engenharia Informática e do Mestrado de Segurança de Engenharia Informática. Aprendi algo com todos eles. Conseguiram moldar-me o ser cognitivo e emocional, ao ponto de me conseguirem fazer ver para além do que conseguia até então. Além das matérias passaram-me muito mais. Cada um, da sua forma particular, passou-me valores e perspectivas do mundo que não tinha. Como formador e docente que também sou muito evoluí convosco. Estou mais infeliz, mas mais consciente de mim mesmo e das fronteiras que tenho com o conhecimento.

Uma palavra de agradecimento a muitos dos meus colegas de curso que me fizeram perceber a importância de conseguir trabalhar em equipa para o bem comum. Como comunicar é importante no desempenho de qualquer tarefa e como ela pode ser preponderante no nosso sucesso.

A concretização desta dissertação de mestrado foi um grande desafio. Em termos pessoais e profissionais foram anos muito difíceis e conturbados. Um agradecimento muito especial a uma pessoa muito especial que me acompanhou do inicio ao fim do processo. Sempre com palavras de incentivo e resiliência, nunca duvidando da minha capacidade em conseguir alcançar o sucesso. Obrigado Elsa por tudo!

A chegada a bom porto penas foi possível graças às linhas orientadoras e apoio de duas pessoas muito importantes no processo e às quais gostaria de expressar o meu sincero agradecimento:

Ao professor Doutor João Carlos Martins, por desde a primeira hora se disponibilizar para me acompanhar neste projecto. Pela sua atitude pró-activa e palavras de incentivo. Por todos os conhecimentos que me transmitiu, pela proximidade que cultivou e amizade que permitiu construir.

Ao professor Doutor José Jasnau Caeiro, pois este projecto, em grande parte, lhe pertence. Foi ele que me resgatou quando o naufrágio estava eminente e as correntes puxavam para o fundo. A sua objectividade e profundos conhecimentos foram, constantemente, mantendo-me à tona. Regularmente tomou conta do leme na direcção correcta levando ao

sucesso deste projecto. Agradeço todo o tempo e dedicação que investiu comigo.

Não poderia esquecer a instituição que foi minha casa durante largos anos, como discente e depois como docente: o Instituto Politécnico de Beja. À sua direcção, ao pessoal docente e não docente o meus mais sinceros agradecimentos.

Índice

Índice de Figuras

Abreviaturas e Siglas

TPM Trusted Platform Module

DRM Digital Rights Management

SRTM Static Root of Trust Measurement

TEE Trusted Execution Environment

REE Rich Execution Environment

TOS Trusted OS

IPB Instituto Politécnico de Beja

ARM The ARM Architecture

CISC Complex Instruction Set Computer

IDE Integrated Development Environment

IRQ Interrupt Request

ISA Instruction Set Architecture

IoT Internet das Coisas (*Internet of Things*)

RISC Reduced Instruction Set Computer

Secure world The execution environment when the core is in the Secure State

SoC System-on-Chip

Thumb An instruction set extension to ARM

Thumb-2 A technology extending the Thumb instruction set to support both 16-bit and 32-bit instructions

TrustZone The ARM security extension

FIQ Fast Interrupt

IRQ Interrupt Request (normally external interrupts)

GCC GNU Compiler Collection

NEON The ARM Advanced SIMD Extensions

QEMU A processor emulator

VFP Vector Floating-Point

PC *Personal Computer*

SO Sistema Operativo

SOs Sistemas Operativos

TCB Trusted Computing Base

CPU *Central Processing Unit*

SM Secure Monitor

SIMD Single Instruction Multiple Data

app Aplicação móvel

apps aplicações móveis

NS Non Secure

SCR Secure Configuration Register

SMC Secure Monitor Call

NDA non-disclosure agreements

TZASC TrustZone Address Space Controller

TZMA TrustZone Memory Adapter

DRAM Dynamic Random-Access Memory

SDRAM Synchronous Dynamic Random-Access Memory

MSR Move to Status Register

MRS System Coprocessor Register to ARM Register

VMM Virtual Machine Monitor

CPSR Current Program Status Register

SGX Software Guard Extensions

VPN Virtual Private Network

TA Trusted Application

SCA Side-Channel Attacks

OP-TEE Open Portable Trusted Execution Environment

NDA Non-Disclosure Agreements

STO Security Through Obscurity

PSP Platform Security Processor

OP-TEE Open Portable Trusted Execution Environment

IP Propriedade Intelectual

VM Virtual Machine

EL Exception Level

UUID Universally Unique Identifier

Capítulo 1

Introdução

> "The only thing for which I can perhaps claim sole credit is the name: three ungrammatical words that now label computing's future."
>
> *Kevin Ashton*

Decorria a primavera de 1999 quando Kevin Ashton, organizando uma apresentação para o seu trabalho na Procter and Gamble, emprega pela primeira vez o termo Internet das Coisas (*Internet of Things*) (IoT).

No mundo atual a ubiquidade tecnológica é exponencialmente crescente. A quantidade de dados gerados a cada instante é colossal. Além de outros desafios, garantir a segurança dessa grande quantidade de dados é uma tarefa incontornável e enorme. É notório o crescimento da consciencialização sobre as questões relacionadas com a segurança dos sistemas de informação. Contudo, na grande maioria das vezes, a mesma é descurada ou completamente esquecida. Por outro lado, o desenvolvimento de soluções de *hardware* e *software* sofreram uma mudança drástica nos conceitos envolvidos.

As soluções de segurança assentam, na sua grande maioria, em funções implementadas ao nível do *software*. Desde a encriptação de dados, controlo de autenticação, transacções, *etc*. É de extrema importância que os sistemas computacionais integrem ferramentas ao nível da própria lógica da arquitetura que permitam assegurar parte dessa segurança, nomeadamente: integridade, confidencialidade e autenticação.

Num mundo cada vez mais dependente da conetividade, a segurança não pode ser uma opção e tem de ser considerada desde a concepção do sistema. Existe um grande número de

consequências conhecidas, e outras tantas desconhecidas, derivadas da falta de segurança. O descuramento ao nível da segurança está na origem de inúmeros inconvenientes pessoais, na origem de fraudes financeiras, espionagem e sabotagem industrial, segurança nacional e até física [4].

As melhores práticas de segurança dos sistemas obrigam à realização das opções corretas ao nível do *design*, das características, implementação, teste, configuração e manutenção destes sistemas. Existem muitas outras considerações a ter em conta, incluindo protocolos de comunicação utilizados, tipos de cifragem, tecnologia, *software*, API's, plataformas, etc.

Desde 2004 que a ARM está atenta a estes assuntos e percebeu que a segurança deve iniciar logo no núcleo do sistema: *Central Processing Unit* (CPU). Iniciou o processo de integração de medidas de protecção e segurança no desenvolvimento dos seus núcleos de processamento e desenvolveu também soluções de *software* que o complementam [5]. A solução encontrada permite o isolamento da execução de código crítico bem como de todos os recursos envolvidos. Garantindo que esse processo é executado em recursos de *hardware* próprios e sem a possibilidade de acesso por terceiros.

Os projetistas de sistemas têm na tecnologia ARM TrustZone uma ferramenta de peso para lidar com estas questões de segurança. A tecnologia TrustZone é implementada em System-On-Chip ou processadores e está disponível nos atuais processadores gerais «*Application processors*» e também nas novas gerações dos microcontroladores ARM. Esta tecnologia tem sido ignorada desde 2004, mas nos últimos anos, devido a inúmeras iniciativas, tem vindo a amadurecer.

Os sistemas operativos são complexos pela sua natureza. São constituídos por dezenas de milhares de linhas de código razão pela qual é relativamente fácil encontrar vulnerabilidades. A TrustZone garante que o sistema operativo não é corrompido e permite o isolamento na execução de processos críticos.

1.1 Objetivos

O trabalho tem como principais objetivos:

- Estudar os diferentes mecanismos de segurança incorporados nas arquiteturas dos processadores e microcontroladores atuais, nomeadamente na arquitetura ARM.

- Conhecer as principais vantagens e desvantagens bem como as vulnerabilidades identificadas nestes mecanismos.

- Estudar e aprofundar conhecimentos sobre a tecnologia TrustZone da ARM. Apresentar casos de estudo que comprovam a necessidade e aplicabilidade da tecnologia TrustZone. Apresentar tecnologias concorrentes idênticas.

- Documentar as fases de instalação, configuração, compilação e produção de uma aplicação baseada em tecnologia TrustZone. Documentar dificuldades encontradas nas diversas fases e sua resolução.

- Desenvolver um projeto prático como prova de conceito de aplicação assente na tecnologia TrustZone.

1.2 Motivação

Os dispositivos móveis, com forte foco nos *smartphones* e *wearables*, tornaram-se ferramentas indispensáveis na realização das mais comuns tarefas diárias. Recolhem e processam grandes quantidades de dados, muitos dos quais sem o nosso conhecimento ou consentimento, alimentando a *cloud*. Regularmente efetuamos múltiplas transacções envolvendo os nossos dados pessoais ou informações críticas. A segurança destas transacções dependem da integridade das aplicações (*software*) bem como do sistema (*hardware*) onde estão a ser executadas.

Devido à sua complexidade os sistemas operativos modernos não conseguem garantir às aplicações que executam uma eficaz proteção contra vulnerabilidades. O espetro dos ataques podem ser de várias ordens e complexidade, podendo tratar-se de um simples *malware* até um poderoso *ransomware*. Assim, a confidencialidade, integridade e disponibilidade dos dados ficam comprometidas levando à falha na segurança dos sistemas.

A evolução tecnológica, conjugada com necessidades da sociedade moderna, contribuíram para uma mudança de paradigma nos sistemas computacionais. Um mundo onde biliões de objetos interligados podem sentir, comunicar e partilhar informação. Interligados por redes IP públicas ou privadas estes objetos recolhem dados de forma regular, que analisam e utilizam para iniciar determinada ação oferecendo capacidade de inteligência para planear, gerir e tomar decisões [6]. Este é o mundo da IoT.

Com a proliferação de sistemas IoT emergiram novos desafios para a segurança derivadas das suas características, nomeadamente:

- Menor capacidade de processamento.

- Necessidade de baixo consumo energético.

- Menor capacidade de memória.
- Miniaturização do *chipset*.
- Elevado número de dispositivos que obriga a reduzido custo de produção.
- Proliferação de protocolos adaptados aos menores recursos disponíveis apresentando arquiteturas mais simples e sem certificação de segurança.

Parte substancial da segurança dos dados assenta em processos criptográficos conhecidos por consumirem vastos recursos de processamento e memória dos sistemas. Pelas razões explanadas é pertinente alterar substancialmente a forma como se olha para as questões da segurança nesta nova geração de dispositivos. Aliar a qualidade nos processos de desenvolvimento do *software* com a segurança nos mais diversos *layers* do sistema é fundamental.

É importante que as aplicações possuam mecanismos de segurança que lhes permitam garantir o isolamento de processos e dados críticos. Os avanços técnicos na engenharia do *software* têm contribuído para a produção de *software* mais seguro. No entanto pode não ser suficiente face à sofisticação e criatividade dos ataques atuais.

Os fabricantes de *hardware*, nomeadamente de processadores e System-on-Chip (SoC), atentos à problemática da segurança têm incorporado nas últimas décadas nos seus *cores* mecanismos de segurança que permitem às aplicações isolarem dados e processos críticos. É o caso dos processos criptográficos, processos de autenticação, protocolos de comunicação, direitos de autor, etc. A engenharia de *software* deverá adaptar as suas técnicas de desenvolvimento de modo a tirar partido destas novas capacidades adaptando as aplicações a esta nova dimensão no processo de desenvolvimento.

Tendo em vista o colmatar destas falhas os fabricantes de *hardware* desenvolveram ferramentas que permitem um acréscimo da segurança no *core* das suas arquiteturas. É objetivo desta dissertação apresentar algumas dessas tecnologias centrando atenção especial na TrustZone desenvolvida para SoC ARM.

1.3 Contribuições

Os mecanismos de segurança TrustZone são relativamente complexos. Um dos objetivos do presente trabalho é demonstrar o processo de preparação da *framework* que permitirá criar o ambiente de desenvolvimento seguro, conduzindo posteriormente ao desenvolvimento de uma aplicação segura - Trusted Application (TA). Devido a vários fatores, explicados

mais adiante, a informação técnica disponível sobre o tema é parca, tornando-se pertinente reunir mais documentação que permita, de alguma forma, auxiliar o desenvolvimento de trabalhos sobre a tecnologia TrustZone.

No decorrer da instalação da Trusted Execution Environment (TEE) Open Portable Trusted Execution Environment (OP-TEE) foram encontradas diversas adversidades. Muitas requerem do instalador conhecimentos profundos do sistema, o que nem sempre acontece. Este trabalho documentou os processos de forma mais detalhada de modo a auxiliar futuros programadores. Se a utilização e desenvolvimento do *software* apresentou os seus obstáculos, a utilização do *hardware* não é diferente. Nem todas as *frameworks* apresentam *porting* para todos os dispositivos disponíveis. Além da dificuldade de escolha das plataformas de *hardware*, existe ainda o obstáculo da constante evolução, tanto a nível das TEE bem como do *hardware*. Tornando a configuração do sistema mais complexa e instável [7].

Como forma de superar as dificuldades e validar a tecnologia Trust Zone está programada a integração do dispositivo de hardware desenvolvido, bem como a aplicação, num projeto experimental IoT intitulado IoT Lysimeter System with Enhanced Data Security [8]. Projeto esse desenvolvido no âmbito de outra dissertação do Mestrado de Internet das Coisas. É objetivo utilizar a presente solução como forma de implementar a segurança na recolha e tratamento dos dados adquiridos pelo referido sistema.

A arquitetura do sistema, a implementação do hardware e o módulo de aquisição de dados desenvolvido utilizando a OP-TEE resultou num artigo científico que foi apresentado e publicado na atas da conferência *APCA International Conference on Automatic Control and Soft Computing (CONTROLO 2022)* [9].

1.4 Estrutura do Trabalho

A dissertação está organizada em capítulos, com as respetivas secções e subsecções.

O capítulo 2 - Literatura e Estado da Arte - relata o trabalho de análise realizado para suporte ao tema desta dissertação. É realizada uma introdução sobre os assuntos a desenvolver no capítulo 2.1. Na secção 2.2 é apresentada alguma da evolução histórica no *hardware* dos equipamentos computacionais e a sua relação com a segurança. Na secção 2.3 é analisada a arquitetura ARM TrustZone, a sua aplicabilidade prática (subsecção 2.3.1), uma breve análise às vulnerabilidades documentadas (subsecção 2.3.2) e as desvantagens encontradas no uso da TEE TrustZone (subsecção 2.3.3). Na secção 2.4 são identificadas algumas das tecnologias concorrentes à ARM TrustZone bem como as suas principais características. No final do capítulo são apresentadas algumas conclusões sobre o mesmo na

(secção 2.5).

No capítulo 3 - Realização Experimental - é apresentada a arquitetura da aplicação desenvolvida. Para auxiliar em futuros trabalhos foram resumidos os pontos mais importantes do processo de instalação (subsecção 3.3.1), a configuração do ambiente de desenvolvimento (subsecção 3.3.2) e o processo de compilação da *Trusted Aplication* (subsecção 3.3.3). Finalmente é apresentada a aplicação desenvolvida, com algum pormenor, na subsecção 3.3.4. No final do capítulo são apresentadas algumas conclusões na (secção 3.4)

Por último, no capítulo 4 - Conclusões - é realizada uma síntese de toda a pesquisa realizada e da realização prática.

Capítulo 2

Literatura e Estado da Arte

2.1 Introdução

O computador, na sua forma digital, nasceu com a evolução dos circuitos eléctricos por volta do ano de 1940. A forma mais comum de guardar números (dados) era o *flip-flop*, o relé eletromecânico e as válvulas de vácuo. Estes dispositivos eletrónicos, que possuíam dois estados, conduziram à representação binária dos números. O resto é história [10].

A arquitetura x86 foi introduzida, pela primeira vez, no processador Intel 8086 em 1978. Tornou-se a base das gerações de computadores durante décadas [11]. Inicialmente com uma velocidade de relógio de 5Mhz e composto apenas por 29,000 transístores. A rápida evolução da tecnologia associada à eletrónica e ao fabrico de sistemas eletrónicos integrados tornou a escalada no desempenho computacional uma constante.

Ao processador 8086 seguiram-se as arquiteturas Intel 286 (1982), Intel 386 (1985) e Intel 486 (1989). Surge então a AMD que veio introduzir novas abordagens no desenvolvimento das arquiteturas conduzindo à evolução que conhecemos hoje.

Poucos previam tamanha transformação nas últimas duas décadas: a era do *Personal Computer* (PC) terminou com a chegada dos computadores portáteis, com um formato ideal e duradouro. As potencialidades do computador já saíam da secretária para locais onde antes eram inacessíveis. A capacidade dos computadores portáteis, alavancada pelo desenvolvimento das comunicações móveis, deram início à ubiquidade que a sociedade experimenta hoje. As telecomunicações acompanharam este rápido desenvolvimento tecnológico e os telemóveis rapidamente ganham acesso à Internet, conduzindo ao surgimento de novos formatos: *smartphone*, *tablet*, híbridos, etc.

Nas primeiras décadas deste século os equipamentos tecnológicos ficaram ainda mais acessíveis em termos económicos. Os processadores aumentam a sua capacidade de proces-

samento, numa proporção inversa às suas dimensões e custo, até que surgem, em massa, os microcontroladores e com eles a IoT. Atualmente podemos colocar capacidade de processamento onde dantes não era possível, e dotar todo e qualquer dispositivo com um processador e ligação à Internet.

Não foi diferente com o *software*, existindo um processo evolutivo constante. Na década de 60 iniciou-se a consciencialização da importância e do impacto dos sistemas de *software* em muitas das atividades da sociedade humana. A produção de *software* enfrentava inúmeras dificuldades: Aos processos de desenvolvimento faltavam fundamentos teóricos sólidos, baseados nas ramificações dos ramos da engenharia. É realizada então a primeira conferência sobre o tema da Engenharia de *Software* em 1968 [12]. No entanto, é nos anos 90 que o termo «evolução» começou por ter aceitação na indústria do *software*. Mais concretamente, as preocupações com o desenvolvimento de *software* seguro começam por volta de 2001. Os programadores, arquitetos de sistemas e a comunidade científica verificaram que a dependência das boas práticas de uso do *software* não era suficiente e concluem que um dos aspetos críticos e centrais, por detrás da segurança de um sistema computacional é um problema de *software* [13].

Um dos objetivos desta dissertação é mostrar que, a afirmação anterior, não corresponde à verdade completa. O surgimento constante de novas abordagens, linguagens de programação e modelos de produção de *software* conduziram ao desenvolvimento de *software* cada vez mais fiável e robusto. A ideia de *software* livre de erros é a meta a atingir, no entanto é preciso ter consciência que não passa de uma utopia [14]. Perante toda esta evolução estamos a assistir a uma profunda transformação nos sistemas computacionais. Os dados deixaram de estar centralizados no *core* dos sistemas de informação. São recolhidos por pequenos dispositivos dispersos onde a segurança é difícil de manter.

2.2 Segurança e o *hardware*

A complexidade dos sistemas computacionais atuais é extremamente elevada. A indústria dos semicondutores aplica uma combinação de técnicas diferentes que visam garantir a segurança do projeto dos seus sistemas de computação integrados (SoC). A crescente sofisticação dos ataques informáticos estão a explorar defeitos que ultrapassam diversas camadas de *software* e *hardware*, e a ataques que conduzem a subtis interações entre o *hardware* e o *software*. Prova disto são os mais recentes *exploits*[1] que afetam os principais

[1]Ataque que se aproveita de vulnerabilidades em aplicativos, redes, sistemas operativos ou *hardware*. *Exploits* geralmente são um *software* ou código que tem como objetivo assumir o controle de computadores ou roubar dados de rede.

atores do mercado. A segurança torna-se assim, obviamente, um problema transversal a todo o sistema. É a conjugação de esforços entre a qualidade do *software* e a implementação de mecanismos de segurança no próprio *hardware* que parece ser a direção a seguir [15].

Um dos métodos usados com vista a garantir a segurança passa pela criação de uma Trusted Computing Base (TCB). Uma TCB define-se pela quantidade mínima de código, *hardware*, utilizadores, processos, etc, em que se pode confiar com o objetivo de atingir os requisitos de segurança [16].

Os processos críticos necessitam de maior proteção do próprio sistema operativo, inseguro por natureza. É necessário garantir o isolamento dos processos. Uma possível solução é a criação de uma TEE. Trata-se de uma área no processador de um dispositivo separada do sistema operativo principal. Garantindo que os dados são guardados, processados e protegidos num ambiente seguro [17]. A TEE disponibiliza aos programadores métodos e ferramentas que garantem que os processos críticos são isolados, mantendo dados sensíveis, chaves de cifragem, processos de autenticação, *etc*, em zonas de memória protegidos do próprio sistema operativo e de aplicações que podem estar ou serem corrompidas. Este método visa permitir uma separação total de recursos. O acesso a registos, *caches*, memória e periféricos ficam assim isolados do sistema operativo nativo, potencialmente inseguro. O próprio código pode executar com total separação no *core*, evitando assim o acesso a qualquer informação crítica do processo, que se quer seguro.

Os arquitetos e fabricantes de processadores e microcontroladores, atentos a estas questões de segurança, têm vindo a incorporar diversas extensões nos *cores* dos seus SoCs e CPUs. Extensões essas que vieram permitir diferentes abordagens na implementação de segurança, nomeadamente as TEE.

2.3 ARM TrustZone

Atenta às questões da segurança, a ARM implementa nas suas arquiteturas Cortex-A e Cortex-M a tecnologia TrustZone. A tecnologia TrustZone consiste num conjunto de extensões de segurança baseada em *hardware* [18]. Embora apenas nos últimos anos o seu potencial tenha vindo a ser explorado, a sua introdução data de 2004 [5].

Esta dissertação tem como um dos objetivos dissecar a arquitetura ARM Coxtex-A, também conhecida por ARMv8-A (*Application profile*) [19], que, como a designação indica, tem como foco principal a sua utilização na execução de aplicações em computadores de uso genérico. A ARM disponibiliza mais duas vertentes da mesma arquitetura, ambas com diferentes áreas de aplicação, a ARMv8-R (*Real-time profile*) que disponibiliza proces-

sadores de alto desempenho para ambientes críticos e seguros (servidores) [20]. ARMv8-M (*Microcontroler profile*) que inclui microcontroladores de baixa latência para desenvolvimento de sistemas embutidos e sistemas IoT [21].

A tecnologia TrustZone também foi adaptada à nova geração dos microcontroladores Cortex-M. Apresenta um modelo semelhante mas com aplicação prática distinta [22]. Os processadores da família Cortex-R não disponibilizam as extensões TrustZone. A sua segurança é realizada através de extensões de virtualização com segurança reforçada e proteção de memória.

Os processadores ARM são do tipo Reduced Instruction Set Computer (RISC). Implementam as características típicas deste tipo de arquiteturas. A essas características adicionam ainda:

- Instruções que combinam operações de deslocamento (*shift*) com operações aritméticas e lógicas.
- Auto-incremento e auto-decremento de modos de endereçamento para a optimização de ciclos.
- Instruções de leitura e escrita na memória (*Load* e *Store Multiple*) para a maximização das taxas de transferência de dados.
- Execução condicional de várias instruções para a maximização das taxas de execução.

Estas melhorias face às características básicas da arquitetura RISC fazem com que os processadores *ARM* atinjam um bom compromisso entre uma elevada *performance*, dimensão dos programas, consumo energético e área de silício [23].

A tecnologia TrustZone assenta o seu conceito na criação de dois domínios de segurança designados por: mundo normal (*normal world*) e mundo seguro (*secure world*) como representado na fig. 2.1. O domínio relativo ao mundo normal não pode aceder aos recursos e memória alocada para o domínio do mundo seguro. Dessa forma, é garantida a confidencialidade e integridade dos dados críticos.

O objetivo da tecnologia ARM TrustZone é permitir usufruir de um sistema operativo aberto e repleto de funcionalidades (RichOS [2]) oferecendo um ambiente Rich Execution Environment (REE), que é executado no mundo normal e que sendo mais complexo, está sujeito a mais vulnerabilidades. O código sensível é executado num sistema operativo mais pequeno e seguro, denominado por Trusted OS (TOS) e isolado do RichOS e cujo

[2]Um sistema operativo que disponibiliza aos utilizadores acesso às funcionalidades gerais de um dispositivo com muitas funcionalidades. Exemplos típicos são o Linux, Android, iOS, Windows.

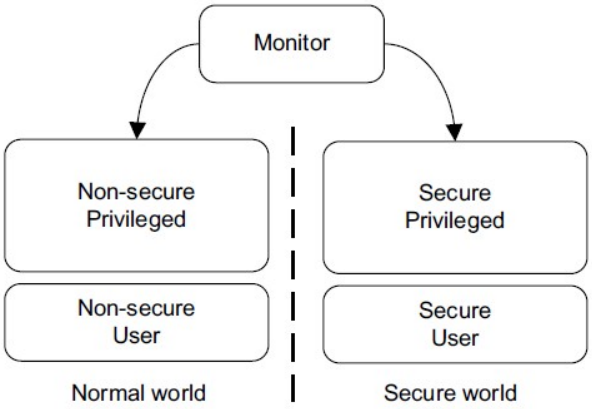

Figura 2.1: Mundo normal e mundo seguro [1]

acesso é controlado por uma *gateway* denominado Secure Monitor (SM). A função do SM é controlar as transições entre os dois sistemas ou mundos. Esta arquitetura permite assim criar uma TEE. Uma arquitetura de *hardware* bem projectada do ponto de vista da segurança conjugada com o design do *software*, permitindo que as rotinas e dados sensíveis permaneçam encapsulados, ainda que sejam utilizados *RichSO's*, menos seguros [24].

No mundo normal executa-se um REE, por exemplo, Linux ou Android. O subsistema seguro a executar é, por exemplo, OP-TEE ou uma versão modificada do Linux. Existe uma preocupação com a sequência de *boot* do sistema e a *framework* que permitirá a sua comunicação. A garantia de uma *chain of trust* para ambas as imagens dos sistemas operativos é obtida por verificação criptográfica. Após o fim do processo de arranque, os dois sistemas operativos podem comunicar entre eles através do modo *kernel monitor*. O *software* do mundo normal pode aceder a este modo através de *hardware interrupt*, um sinal externo de *abort* ou utilizando a instrução Secure Monitor Call (SMC). O mundo seguro pode usar esses métodos ou escrever directamente no Current Program Status Register (CPSR).

Uma característica importante da arquitetura ARMv8-A é a sua retro-compatibilidade com as versões anteriores. Suporta também modos de execução de 32 *bits* (AArch32) e 64 *bits* (AArch64) [25]. Ambos os modos suportam instruções do tipo Single Instruction Multiple Data (SIMD).

A transposição entre os dois ambientes (ambiente normal e ambiente seguro) é controlada por um 33º bit denominado Non Secure (NS) *bit* que está no registo. O valor deste é lido pelo Secure Configuration Register (SCR), e também é propagado pelo sistema de memória e *bus* de comunicação com periféricos. A TrustZone adiciona mais um modo de

funcionamento do processador responsável pelo controlo das transições entre os dois ambientes, independentemente do valor do NS *bit*. Terá, por isso, uma função de *gateway* e recebeu o nome de *monitor mode* (MM). Desta forma, foi adicionada a instrução SMC, que vai permitir as transições de estado por parte do *software* de ambos os ambientes. Para além do *bit* NS *bit* e SMC é possível a comutação entre os dois estados recorrendo a excepções, Interrupt Request (normally external interrupts) (IRQ) e Fast Interrupt (FIQ). Os FIQ diferenciam-se dos IRQ por usufruírem de um nível de prioridade mais alto. Existem registos protegidos no sistema, cujo acesso é condicionado e protegido pelo *software* que controla o ambiente seguro.

A TrustZone implementa também o TrustZone Address Space Controller (TZASC), que permite mapear a memória em zonas seguras ou não seguras e é controlado pelo ambiente seguro. Possibilita o particionamento da memória em diversas zonas. Caso o fabricante do SoC não implemente TZASC a arquitetura não será segura pois não existe separação no acesso à memória entre os dois ambientes.

Com uma função semelhante ao TZASC existe o TrustZone Memory Adapter (TZMA). Enquanto o TZASC permite particionar a memória Dynamic Random-Access Memory (DRAM), o TZMA gere a memória externa Synchronous Dynamic Random-Access Memory (SDRAM). Importa referir que estes componentes são ambos opcionais na implementação de cada SoC pelos fabricantes [18].

Com o intuito de reforçar o isolamento dos dois mundos, na arquitetura ArmV7-A, o processador tem ao seu dispor alguns registos especiais, e outros do sistema que são apenas acedidos pelo coprocessador. Já na arquitetura ArmV8-A são utilizadas instruções Move to Status Register (MSR) e System Coprocessor Register to ARM Register (MRS) para mover o conteúdo destes registos.

2.3.1 Utilização

Muitos dos equipamentos móveis atuais utilizam processadores com arquitetura ARM [26]. Estes equipamentos, na sua grande maioria *smartphones*, são a plataforma de eleição de diversos fabricantes, que adquirem a Propriedade Intelectual (IP) à *ARM*, e a incorporam nas suas arquiteturas.

Muitos dos serviços oferecidos pelas empresas assentam em aplicações móveis (apps). Grande parte dessas apps são executadas pelo sistema operativo em Android que é um dos sistema operativos com maior expansão nos dias que correm. É nos *smartphones* que transportamos muitos dos nossos dados pessoais e executamos transacções importantes diariamente. Contudo, a popularidade do Android e a natureza aberta como são dispo-

nibilizadas as apps, fazem dele um alvo apetecível para ataques informáticos, colocando assim em perigo os nossos dados, que podem ser roubados ou adulterados.

Devido à crescente quantidade de dados críticos guardados nos *smartphones*, as questões de segurança têm merecido a atenção da indústria e comunidade académica. Resultado disso, são muitos os mecanismos de segurança e as ferramentas desenvolvidas. Grande parte dessas ferramentas são *anti-malware* e propõem-se mitigar os riscos de segurança destes equipamentos. Contudo, essas ferramentas e mecanismos executam em equipamentos que podem estar já infetados, tanto a nível do sistema operativo como do próprio *hardware*.

Tendo em vista a mitigação deste problema foi desenvolvido o *Mobile Device Security with ARM TrustZone*[3]. Este trabalho utiliza as extensões de segurança ARM TrustZone com vista a criar um ambiente isolado e seguro assistido por *hardware*. Esse ambiente, que será aprofundado no Capítulo 3, não é mais do que uma TEE [27]. O trabalho desenvolvido apresenta duas vertentes. Na primeira são desenvolvidos serviços de segurança com o intuito de detetar intrusões no dispositivo móvel e na segunda vertente desenvolver forma de garantir a autenticidade e integridade de uma *app* que seja executada no sistema operativo Android.

São várias as *apps* das quais dependemos diariamente e cuja segurança é crítica tais como acesso a *homebanking*, transacções comerciais, pagamentos, acesso a dados críticos no local de trabalho (conceito de *Bring Your Own Device* (BYOD)), *etc.* Pensando nestas questões foi desenvolvido o TrustICE [28], consiste numa *framework* que permite criar um ambiente computacional isolado em dispositivos móveis e que utiliza as extensões TrustZone para executar apps de forma isolada do RichOS.

Várias empresas associaram-se ao *ARM TrustZone Program* permitindo-lhes desenvolver produtos e aplicações seguras [29]. Um exemplo é a Verimatrix [30], que desenvolve soluções de *hardware* e *software* em torno das extensões TrustZone da ARM. Conta com soluções em *Digital Rights Management (DRM)* [31], autenticação por *hardware* em soluções Virtual Private Network (VPN) para dispositivos móveis e *tablets* [32]. Muitas outras soluções podem ser consultadas no site do Verimatrix [33].

Existem projetos com grande aplicabilidade prática da TrustZone em domínios diversos, tais como indústria, sector automóvel e aeroespacial. Disponíveis sob diferentes politicas de licenciamento, desde o *open-source* a projetos proprietários.

[3]Tese de doutoramento realizada sobre o assunto.

2.3.2 Vulnerabilidades documentadas

A segurança de centenas de milhões de dispositivos móveis está confiada a TEEs que assentam nas extensões TrustZone, com vista à protecção de aplicações em que a segurança é crítica. Embora seja assumido que as TEEs são altamente seguras, nos últimos anos têm sido muitos os ataques bem sucedidos, muitos deles com elevado impacto em diversas plataformas [34], levantando dúvidas sobre a real fiabilidade que uma TEE pode garantir. Estudos apontam que erros críticos de implementação das TEE estão entre os mais comuns. Muitos *bugs* têm sido encontrados em TA (aplicações TEE) [35] [36]. Muitos desses erros são os clássicos *buffer overflows* [37], que conduzem a falhas de segurança no *kernel* ou nas TA de dispositivos de marcas como Samsung, Google e Huawei [38] [39] [40]. Têm sido identificadas inúmeras deficiências na arquitetura das TEE assistidas pela TrustZone [41], muitas delas relacionadas com os mecanismos de proteção de memória [42] e as chamadas de sistema por parte da TA.

Outra frente de ataque é o *hardware*. A manipulação de parâmetros de funcionamento, como a tensão eléctrica e a frequência de relógio, podem ser utilizadas para quebrar a barreira de segurança da TrustZone. Para manipular estas variáveis nem sempre é necessário recorrer a alterações no *hardware* ou adicionar novos circuitos [43].

Podemos considerar que as memórias são, normalmente, partilhadas entre a vítima e o atacante. O atacante, ao monitorizar a memória, pode ter acesso a dados sensíveis. O comportamento, alterações e velocidades de acesso podem permitir corromper a segurança, permitindo uma quebra nos mecanismos de segurança [44] [45].

Os ataques denominados por Side-Channel Attacks (SCA) exploram as dimensões físicas de um sistema computacional. Medindo e estudando o consumo de corrente, tensão, campos eletromagnéticos gerados, *etc*, e recorrendo a diversas áreas da análise de sinais, com o conhecimento do funcionamento da arquitetura é possível aceder a recursos do sistema que, em situação normal, não seria possível [45] [46].

2.3.3 Desvantagens da *TEE TrustZone*

Embora as vantagens da aplicabilidade da tecnologia TrustZone sejam evidentes existem aspetos a serem melhorados, nomeadamente: **Apenas permite a execução um processo seguro simultâneamente.** Nas arquiteturas modernas o multicore veio permitir o aumento do número de tarefas executadas por unidade de tempo (throughput). Desta forma consegue-se mais capacidade de processamento sem a necessidade de aumentar a velocidade dos sinais de relógio. No entanto a TrustZone apenas permite uma chamada de

processo seguro em simultâneo. Com o domínio de sistema operativos multi-tarefa existe a necessidade de executar vários processos seguros em simultâneo. Na arquitetura TrustZone a mudança entre ambientes (ambiente normal e ambiente seguro) poderá representar uma perda de desempenho substancial com a impossibilidade de executar dois processos seguros em simultâneo e a paragem do processamento do mundo normal.

Falta de documentação. Embora os fabricantes de *hardware* forneçam documentação, esta nem sempre é completa e suficiente. Para programadores com pouca experiência anterior, encontrar respostas na documentação é um processo difícil e nem sempre eficaz. Alguns fabricantes oferecem fóruns comunitários onde os programadores podem colocar questões, mas que muitas vezes não são respondidas ou as respostas não são totalmente esclarecedoras. Os programadores são assim obrigados a continuar a procura da solução, tendo de investir muito mais tempo e num processo de tentativa e erro [7].

Dificuldades com o ambiente de execução. Existem algumas *frameworks* de desenvolvimento que permitem a exploração da TrustZone, como é exemplo a OP-TEE explorada neste projeto. Estas *frameworks* necessitam ser instaladas e configuradas, mas mais uma vez, os programadores com pouca ou nenhuma experiência têm muitas dificuldades no processo. Dificuldades acrescidas pela falta de conhecimentos sobre o funcionamento dos sistemas. As implementações de cada fabricante podem variar ou não estarem completamente implementadas, conduzindo a um enorme investimento em torno das documentações do *hardware* e das *frameworks*, que nem sempre dão resposta a todas as dificuldades técnicas que podem surgir.

Dificuldades com as placas de desenvolvimento. Todas as placas têm o seu próprio *software* (*TEE*), desenvolvido para funcionar apenas nessa placa específica. Embora o problema não esteja directamente relacionado com a TrustZone, vai levantar muitas questões aquando da escolha do *hardware* a utilizar. Cada placa aplica a mesma função de forma diferenciada, levando a que os mesmos passos de configuração sejam substancialmente diferentes entre elas.

Evolução do hardware. A evolução no *hardware* das placas é constante, e com isso novas funcionalidades são adicionadas. Muitas delas com uma nova forma de implementação e utilização. Isto conduz a significativas alterações na implementação, levando a que seja necessário rever todo o trabalho desenvolvido para as versões anteriores.

Assinatura de Non-Disclosure Agreements (NDA). Outra dificuldade prende-se com o facto dos fabricantes, muitas vezes, exigirem a assinatura de NDA aos programadores antes de libertarem qualquer informação sobre uma determinada implementação [47]. Os trabalhos de pesquisa sobre a TrustZone são em menor número devido à falta de plataformas de desenvolvimento onde as funcionalidades estejam totalmente implementadas ou

desbloqueadas [7].

2.4 Outras tecnologias de segurança no *hardware*

As questões de segurança não estão a ser negligenciadas por parte dos restantes fabricantes de *hardware*. Prova disso é que os principais produtores de CPU e SoC têm desenvolvido as suas próprias tecnologias tentando levar a segurança ao *core* dos seus sistemas.

Intel SGX

A Intel integra nos seus CPUs a tecnologia Software Guard Extensions (SGX). Permite minimizar a possibilidade de ataques oferecendo novas instruções que podem ser utilizadas pelos programadores para estabelecer regiões privadas. Zonas essas que são acedidas apenas por código pré-seleccionado, conforme ilustrado na fig. 2.2, ajudando a prevenir ataques diretos ao código em execução e aos dados carregados em memória [2].

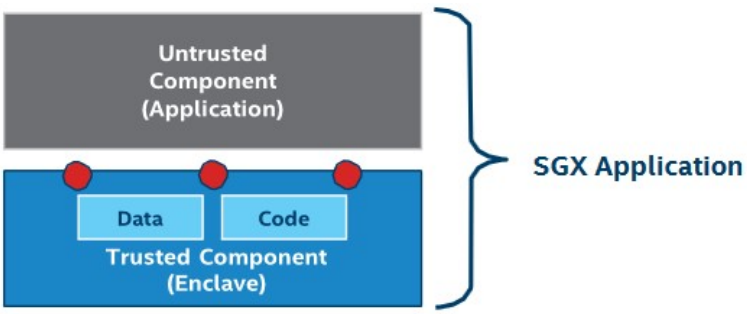

Figura 2.2: Particionamento aplicacional no SGX (*in* Intel product brief [2])

Na fig.2.3 pode-se verificar como o isolamento de um determinado processo crítico é realizado.

- Aplicação desenvolvida composta por *software* não seguro e seguro.
- A aplicação é executada e cria uma zona segura (enclave) que será guardado numa zona de memória protegida.
- A função segura é chamada, a execução é transferida para o enclave.
- O enclave acede aos dados de forma transparente. A tecnologia previne acesso externo aos dados do enclave.

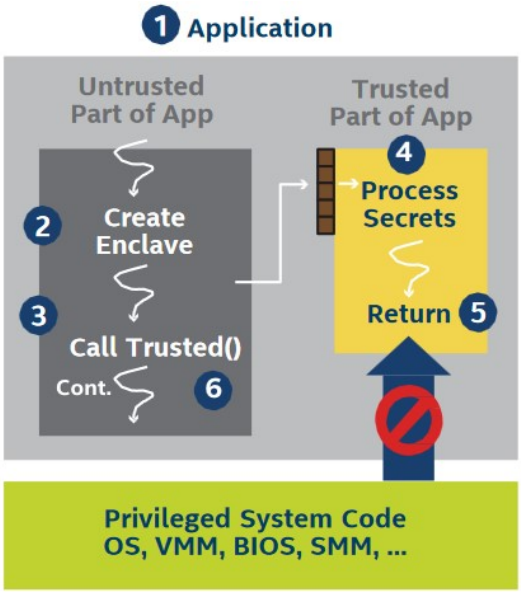

Figura 2.3: Execução em Runtime (*in* Intel product brief [2])

- A função segura retorna à execução normal mantendo os dados protegidos na zona segura da memória.
- A aplicação continua a sua normal execução.

AMD PSP

A AMD incorpora nos seus CPU a Platform Security Processor (PSP). Esta plataforma fornece segurança ao nível do *hardware* recorrendo a um (co)processador ARM com TrustZone. Esta tecnologia é introduzida nos processadores AMD desde 2013 [48] e permite garantir a segurança dos processos ao nível do *hardware*. Não foi encontrada documentação técnica sobre a tecnologia e a sua implementação. Também não foram encontrados estudos ou trabalhos académicos realizados sobre a mesma, não sendo, desta forma, possível conhecer o seu funcionamento. No entanto, tendo por base um processador ARM, com tecnologia TrustZone, o seu funcionamento será semelhante ao já explanado anteriormente. A AMD, à semelhança de outros fabricantes, recorre a Security Through Obscurity (STO) como forma de preservar a segurança do seu sistema. A natureza fechada e desconhecida da STO já demonstrou as suas fragilidades perante os utilizadores. Prova disso são todas as vulnerabilidades reconhecidas deste sistema [49]. A AMD recorre a outras tecnologias de segurança combinadas com o PSP, nomeadamente AMD Memory Guard [50] [51] (encriptação total da memória), Secure Boot e Firmware TPM são disso exemplo [52].

RISC-V MultiZone Security

O projeto desta arquitetura remonta ao ano de 2011 e apresenta como característica a disrupção com as restantes Instruction Set Architecture (ISA) tendo a simplicidade como sua génese. É um projeto *open source* e uma ótima opção para alunos que desejam aprender *assembly*. O projeto apresenta um conjunto de requisitos que devem ser satisfeitos [53]:

1. Adaptada a todos os tipos de processadores, desde o simples microcontrolador até ao computador de alto desempenho.
2. Compatível com uma grande variedade de *software* e linguagens de programação mais populares.
3. Acomodar todas as tecnologias de implementação: FPGAs (Field-Programmable Gate Arrays), ASICs (Application-Specific Integrated Circuits), *chips* customizados e até mesmo futuras tecnologias de dispositivos.
4. Ser eficiente para todos os tipos de microarquitetura: controle microcodificado ou *hardwired*; *pipelines* ordenados, desacoplados ou desordenados; emissão de instrução única ou superescalar; e assim por diante.
5. Apoiar uma ampla especialização para atuar como base para aceleradores customizados, aumentando de importância à medida que a Lei de Moore se desvanece.
6. Ser estável, ou seja, a ISA base não deve mudar. Mais importante, a ISA não pode ser descontinuada, como aconteceu no passado com outras ISA proprietárias, como a AMD Am29000, a Digital Alpha, a Digital VAX, o Hewlett Packard PA-RISC, Intel i860, Intel i960, Motorola 88000 e a Zilog Z8000.

Além de recente e aberto, o RISC-V é modular. A sua base é constituída por uma ISA primária chamada de RV32I. A RV32I nunca será alterado, permitindo aos programadores de linguagem *assembly* estabilidade no desenvolvimento e melhores compiladores. Além deste módulo base inalterável, existem extensões padronizadas opcionais que o *hardware* pode ou não implementar, dependendo das necessidades da aplicação. Esta característica permite implementações muito simples e de baixo consumo energético. Como convenção anexa-se letras da extensão ao nome do módulo base. Como exemplo, o RV32IMFD adiciona as extensões de multiplicação de vírgula flutuante (RV32M), de precisão simples (RV32F) e de vírgula flutuante de precisão dupla (RV32D) às instruções base obrigatórias (RV32I). A sua arquitetura apresenta como extensões de segurança no *hardware* a RISC-V MultiZone. Os seus princípios de funcionamento são, no geral, em tudo semelhante

à ARM TrustZone. Apresenta contudo, uma caraterística diferenciadora: a MultiZone permite criar vários ambientes seguros em simultâneo [54], o que permite uma melhor adaptação aos sistema operativos multi-tarefa atuais, permitindo a criação de vários ambientes seguros isolados, como representado na fig. 2.4. Desta forma as várias aplicações, protocolos, *drivers*, etc, poderão iniciar um processamento seguro e isolado do restante sistema.

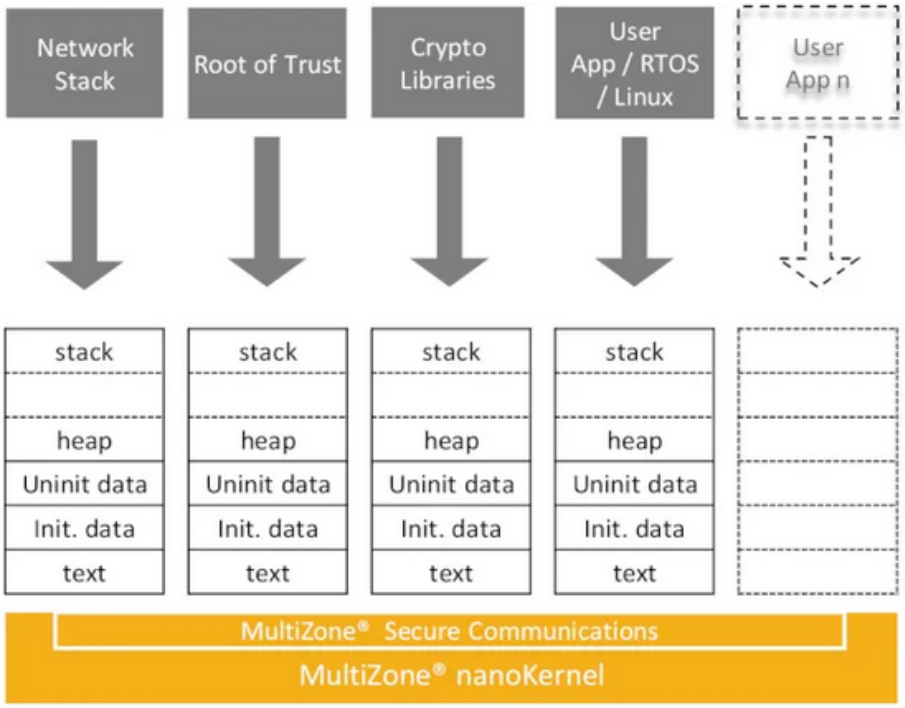

Figura 2.4: MultiZone na arquitetura RiSC-V (*in* [3])

Dentro das TEE, para além da estudada TrustZone, existem outras. Algumas delas menos relevantes, outras com aplicabilidade em diferentes contextos. nomeadamente: SierraTEE, SeCReT, LTZVisor e FreeTEE. Algumas destas TEEs foram algo de testes no inicio deste estudo mas que, pelas mais diversas razões, acabaram por não se enquadrar no mesmo.

2.5 Conclusão

Os sistemas computacionais têm sofrido uma evolução constante conduzindo à massificação dos microcontroladores e com eles a IoT. A proliferação dos dispositivos IoT aliado ao crescimento exponencial na quantidade de dados e operações críticas que são executadas

nestes dispositivos trouxeram novos desafios em matérias de segurança. A engenharia de *software* e processos de desenvolvimento das aplicações evoluíram e acompanharam os avanços tecnológicos. Em muito impulsionada pela consciencialização do grande impacto económico e material que as falhas de segurança apresentam para as pessoas, mas sobretudo para as empresas.

A segurança de uma aplicação crítica está dependente do sistema operativo onde irá ser executada. Devido à sua complexidade apresentam uma grande «superfície de ataque», bem como muitas vulnerabilidades. A sua segurança está dependente de vários pressupostos que nem sempre são fáceis de garantir.

A sofisticação dos ataques aos sistemas computacionais tem sido constante, impulsionando os fabricantes de *hardware* a desenvolver ferramentas que visam permitir integrar a segurança no core dos seus processadores e SoC. As TEE têm apresentado um importante papel na segurança dos sistemas disponibilizando aos programadores métodos e ferramentas que garantem que os processos críticos são isolados, mantendo dados sensíveis, cifras, processos de autenticação, *etc*, em zonas de memória protegidas do próprio sistema operativo e de aplicações que podem estar ou vir a ser corrompidas.

A TrustZone oferece ao programador ferramentas que permitem desenvolver *software* mais seguro e as soluções disponíveis cobrem os mais diversos setores de mercado. Embora seja assumido que as TEEs são muito seguras, nos últimos anos têm sido muitos os ataques bem sucedidos, muitos deles com elevado impacto em diversas plataformas, levantando dúvidas sobre a real fiabilidade que uma TEE pode garantir. Estudos apontam que erros críticos de implementação das TEE estão entre os mais comuns. A ARM apresenta nas suas arquiteturas diversos componentes que nem sempre são implementados pelos fabricantes dos seus SoC conduzindo a falhas de segurança exploradas pelos atacantes.

Embora as vantagens da TrustZone sejam evidentes existem ainda vários aspetos que necessitam ser melhorados.

Capítulo 3

Realização Experimental

3.1 Introdução

A sofisticação dos ataques informáticos tem vindo a aumentar e a criatividade na execução dos mesmos. Os mecanismos e técnicas de segurança devem ser idealizados e integrados em fases iniciais dos projetos e as soluções de segurança implementadas pelo *hardware* apresentam um complemento importante à qualidade do *software*. Os sistemas operativos, pela sua complexidade, apresentam dificuldades naturais em garantir segurança às aplicações que dele dependem.

As soluções ARM TrustZone apresentam TEE desenvolvidas para a proteção das suas aplicações críticas. A TrustZone pode representar um papel chave nos mecanismos de segurança para proteger a integridade e a confidencialidade das aplicações e dados. Tirando partido de *hardware* dedicado, permite a execução de código crítico em domínios, como demonstrado nas próximas secções, isolados do sistema operativo da plataforma.

O desenvolvimento de soluções TrustZone vem permitir uma melhor abordagem à segurança do *software* permitindo maior controlo e isolamento na execução de código crítico ou manipulação de dados.

Este capítulo é dedicado à realização de uma aplicação que decorre da análise das soluções de segurança implementadas pela ARM na sua arquitetura de SoC ARMv8-A. Esta arquitetura pode ser encontrada em *smartphones*, sistemas avançados de condução assistida, soluções de armazenamento, entre outros.

A aplicação tem como objetivo a recolha de dados de sensores, a sua cifragem e gravação em *blockchain*. Os dados e operações consideradas críticas são geridos no mundo protegido (explicado no capítulo 2) na TrustZone, garantindo a sua integridade e confidencialidade no que respeita à segurança de dados.

A aplicação utiliza as extensões TrustZone e foi desenvolvida como prova de conceito. Todas as etapas do processo são detalhadas na secção 3.2 e é apresentada a arquitetura geral do sistema, nomeadamente:

- Operações de aquisição dos dados através de um conjunto de sensores. *Hardware* e bibliotecas utilizadas.
- Utilização de uma base de dados responsável pela indexação dos registos no *blockchain*.
- As operações de cifragem e *hash* com recurso às funções disponibilizadas pela *framework*.
- Armazenamento dos dados numa estrutura do tipo *blockchain* visando garantir a sua disponibilidade e confidencialidade.

A aplicação desenvolvida foi, na continuação deste trabalho, integrada num projeto experimental IoT (IoT Lysimeter System with Enhanced Data Security) como forma de implementar a segurança na recolha e tratamento dos dados adquiridos.

Esse trabalho resultou num artigo científico que foi apresentado e publicado na atas da conferência *APCA International Conference on Automatic Control and Soft Computing (CONTROLO 2022)* [9] que descreve a arquitetura do sistema, a implementação do hardware e o módulo de aquisição de dados utilizando a OP-TEE.

Em seguida resumem-se os passos mais importantes da instalação do sistema OP-TEE (subsecção 3.3.1), a configuração do ambiente de desenvolvimento (subsecção 3.3.2) e a descrição do processo de compilação cruzada (subsecção 3.3.3). O funcionamento da aplicação é explicado na subsecção 3.3.4. As conclusões retiradas de todo o processo de desenvolvimento na secção 3.4 são a parte final do capítulo.

Os fabricantes dos SoC com TrustZone têm relutância em disponibilizar muitos pormenores técnicos sobre a sua implementação, obrigando muitas vezes os programadores a assinar acordos de NDA [55] antes de libertar informações sobre a arquitetura adotada. A pesquisa sobre a TrustZone fica assim dificultada pela oferta limitada em termos de placas de desenvolvimento com as capacidades TrustZone desbloqueadas. Grande parte das placas estão preparadas para executar *boot* pelo ambiente normal, muitas vezes privando o acesso ao ambiente seguro por parte dos programadores [55]. Pelas razões apresentadas, na última década, esta tecnologia tem vindo a ser usada primordialmente pelos fabricantes em

tarefas de monitorização de serviços proprietários, levando ao crescimento das dificuldades no seu estudo pela sua natureza fechada. No entanto, nos últimos tempos tem existido um crescimento no interesse da TrustZone pela indústria e pela academia. Tem sido alvo de estudos em diversos projetos de pesquisa bem como em produtos comerciais. Neste âmbito tem fornecido soluções de segurança em sistemas tais como Samsung Knok [56], Android Keystore [57] e OP-TEE [58].

3.2 Arquitetura do Sistema

A arquitetura da aplicação desenvolvida como prova de conceito sobre o funcionamento de uma aplicação segura com recurso a uma TEE é aqui apresentada.

Após os primeiros testes com o uso da tecnologia, ficaram claras as limitações aos objetivos do projeto. A plataforma de *hardware* seleccionada trouxe alguns constrangimentos e limitações, o que levou à sua substituição. O projeto inicial previa a utilização do RPi4 por ser uma plataforma de *hardware* mais recente. Durante a análise da tecnologia TrustZone concluiu-se que esta plataforma não tem todo o *porting* da *framework* OP-TEE. Apresentado o RPi3 este suporte, foi a plataforma que naturalmente acabou por ser utilizada. As plataformas de *hardware* escolhidas para o desenvolvimento deste projeto, Raspberry Pi 3B+ (RPi3) e Raspberry Pi 4 (RPi4), implementam as extensões e estados da TrustZone, mas não possuem as funções nem o *hardware* que permita realizar um *secure boot*, partição da memória e periféricos, entre outras funções de segurança derivadas da TrustZone. A TEE OP-TEE [58], escolhida como sistema operativo, apenas disponibiliza o *porting* para RPi3 com o intuito de permitir projetos académicos ou prototipagem [59].

A aplicação desenvolvida consiste em duas partes de código bem distintas entre si:

1. Código principal da aplicação cliente (*host*) que é executado no mundo normal e que inclui uma grande parte das funções desenvolvidas.

2. *Trusted Aplication* (TA) responsável pelas operações de cifragem (AES e SHA256).

A OP-TEE implementa especificações da GlobalPlatform. Organização que visa identificar, desenvolver e publicar especificações que facilitam a implementação de segurança e interoperacionalidade e gestão de múltiplas aplicações embutidas com tecnologias de segurança em microprocessadores [60]. A aplicação desenvolvida utiliza as funções disponibilizadas pela GlobalPlatform API como especificado em:

- API Specification v1.3.1 (GPD_SPE_010) [61].

Esta documentação foi a base orientadora para o desenvolvimento de todas as funções utilizadas pois define as regras de como os dois sistemas (OP-TEE e TEE) comunicam entre si.

Existem cinco funções com implementação obrigatória.

```
1 TEE_Result TA_EXPORT TA_CreateEntryPoint( void );
```

TEE_Result é uma variável do tipo inteiro utilizada para devolver os códigos pelas API. A função *TA_CreateEntryPoint()* é o construtor usado pela *framework* quando se cria uma nova instância de uma TA.

```
1 void TA_EXPORT TA_DestroyEntryPoint( void );
```

A função *TA_DestroyEntryPoint()* é o destrutor da TA, invocado pela *framework* quando a instância é destruída. A *framework* garante, assim, que nenhuma sessão se encontra aberta no processo cliente. Quando terminada a sessão nenhum outro *entry point* desta instância será novamente invocado. A sua destruição libertará todos os recursos utilizados pela instância.

```
1 TEE_Result TA_EXPORT TA_OpenSessionEntryPoint(
2                         uint32_t paramTypes,
3                         [inout] TEE_Param params[4],
4                         [out][ctx] void** sessionContext);
```

A *framework* invoca a função *TA_OpenSessionEntryPoint()* quando existe um pedido de um cliente para abrir uma sessão com a TA. O pedido de *open session* poderá resultar na criação de uma nova instância tal como definido na documentação. O *software* cliente pode especificar parâmetros numa operação já aberta que serão passados para a instância da TA como *paramTypes* e *params*. Estes argumentos também podem ser utilizados pela instância da TA para devolver dados para o cliente.

```
1 void TA_EXPORT TA_CloseSessionEntryPoint(
2                         [ctx] void* sessionContext);
```

A *framework* invoca a *TA_CloseSessionEntryPoint()* para encerrar uma sessão com o cliente. A implementação da TA é responsável por libertar todos os recursos consumidos pela sessão a ser encerrada. A TA não poderá recusar o fecho da sessão, mas pode reter o fecho enquanto não retorna da *TA_CloseSessionEntryPoint()*.

```
1 TEE_Result TA_EXPORT TA_InvokeCommandEntryPoint(
2                       [ctx] void* sessionContext,
3                       uint32_t commandID,
4                       uint32_t paramTypes,
5                       [inout] TEE_Param params[4] );
```

Função *TA_InvokeCommandEntryPoint()* é chamada pela *framework* quanto um cliente invoca um comando dentro de uma sessão em aberto. A TA consegue aceder aos parâmetros enviados pelo cliente através dos argumentos *paramTypes* e *params*. Esses argumentos podem também ser usados para transferir informação de volta para o cliente. Um comando é sempre invocado dentro de um contexto de uma sessão do cliente.

A aplicação desenvolvida implementa a utilização destas funções visando a segurança dos dados garantindo o isolamento no seu tratamento. A fig. 3.1 representa o fluxo dos dados e as principais funções correspondentes da aplicação.

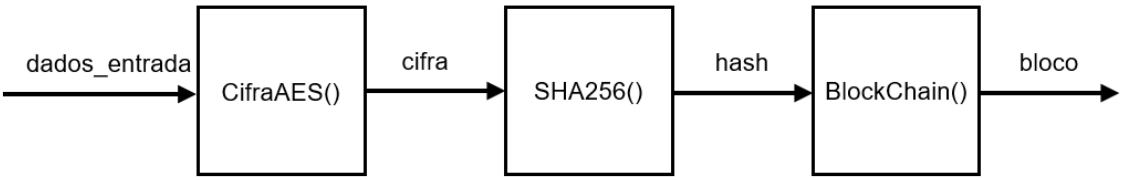

Figura 3.1: Sequência geral dos dados na aplicação

A aplicação cliente recolhe os dados dos sensores e representa-los numa estrutura de dados. Nessa estrutura são adicionados um *timestamp* e um ID que têm como função identificar inequivocamente cada leitura. A informação é armazenada na base de dados. o campo ID é usado como *index* de controlo para o *blockchain*.

Após a inicialização das variáveis usadas na gestão da comunicação do cliente com a TA, é criada uma nova instância desta sendo gerada uma sessão e um contexto. Depois a ligação é enviada para a TA com a chave a utilizar na cifra AES. Os dados recolhidos são enviados para a TA onde serão sujeitos ao processo de cifragem. A TA devolve a cifra ao cliente que enviará a mesma novamente para a TA, mas agora para ser gerada uma *hash* da presente cifra com vista a ser guardada no *blockchain* como prova de integridade dos dados. Nesta fase é gerada uma outra *hash*, mas do bloco anterior (ID-1), o que garante a integridade do próprio *blockchain*. Logo a *hash n-1* deve ser igual à *hash n* do bloco anterior.

A aplicação termina todo o processo libertando os recursos reservados pelo uso da TA terminando a sessão e a finalizando o contexto criado.

A cifra, a *hash* da cifra n e a *hash* da cifra n-1 vão constituir um novo bloco que será guardado no ficheiro do *blockchain*.

O processo será descrito de forma mais detalhada mais adiante na subsecção 3.3.4.

3.3 Realização do sistema

3.3.1 Instalação da OP-TEE

Outras opções foram testadas, algumas referidas anteriormente, mas a falta de suporte à plataforma de *hardware* escolhida entre diversos fatores levaram à escolha da *framework* OP-TEE.

O projeto OP-TEE foi escolhido por diversos motivos:

1. Oferece suporte ao RPi3. O RPi3 foi escolhido como plataforma de desenvolvimento pela sua forte ligação com o mundo IoT permitindo a prototipagem rápida de sistemas.

2. O projeto disponibiliza uma boa documentação técnica. Nem sempre acessível para os menos experientes, mas completa.

O sistema operativo recomendado, o *Linux* Ubuntu LTS (por questões de estabilidade), foi instalado num portátil físico para preparação do *software* da OP-TEE. É utilizada uma máquina física, e não uma máquina virtual, devido a:

1. Facilitar as comunicações com o RPi3 em partes mais avançadas do projeto. As comunicações via UART/USB podem apresentar limitações de funcionamento em ambiente virtuais.

2. Facilitar o acesso ao cartão SD. A utilização de cartões SD em máquinas virtuais obriga a operações mais complexas de montagem dos mesmos.

3. Disponibilidade de equipamentos físicos (portáteis). A existência de alguns equipamentos pessoais sem utilização de momento facilitaram a opção.

4. Instabilidade dos sistemas virtuais foram observadas durante as fases iniciais. Por diversas ocasiões as atualizações do sistema Ubuntu causaram congelamentos no arranque das máquinas. As portas de comunicação nem sempre respondiam como esperado.

À instalação do sistema operativo Ubuntu deve seguir-se de uma imediata atualização. Há pré-requisitos de instalação necessários para o funcionamento da solução. As *toolchains* de compilação são descarregadas diretamente dos servidores da ARM. Nesta fase surgem diversas questões relacionadas com os componentes, nomeadamente:

1. Questões relacionadas com caminhos para as aplicações, bibliotecas e ficheiros de configuração que requerem uma análise dos erros com vista à atualização de alguns ficheiros de configuração.

2. A instalação, configuração e utilização das bibliotecas necessárias ao bom funcionamento da *framework* requerem conhecimentos aprofundados do sistema *Linux*.

3. As variáveis de sistema necessárias para a boa compilação do projeto requerem diversas correções com vista ao bom funcionamento da *framework* e compilação dos projetos.

É necessária uma procura profunda de soluções sobre os problemas de instalação, configuração e utilização da *framework*. A comunidade OP-TEE é relativamente reduzida e suporte técnico não existe.

Após a transferência de todos os componentes para a pasta do projeto e a sua compilação é altura de preparar o cartão SD que irá receber o *bootloader*[1] U-Boot [63]. Inicialmente o processo de transferência da OP-TEE para o cartão SD aparenta ser mais complexo do que realmente é.

Finalizada a preparação do sistema, o cartão SD deve ser inserido no RPi3 e o mesmo iniciado. É consequência da redução ao mínimo, no que respeita a componentes do sistema OP-TEE, o não funcionamento da porta HDMI. O RPi3 deverá ser ligado a um equipamento informático via UART como representado na fig. I.1. É utilizada uma placa FTDI conversora de modo serial UART para USB. O *pinout* a utilizar para interligar o RPI3 ao conversor serial para USB é representado na fig. I.2. É iniciado o terminal/emulador picocom (modo privilegiado) antes de ligar a alimentação do RPi3 no equipamento que vai ser utilizado como interface de comunicação.

A linha de comandos da OP-TEE no RPi3 é apresentada após terminada a sequência de arranque do sistema, fig. I.3.

Uma ferramenta de diagnóstico para testar o correto funcionamento do sistema OP-TEE, o *xtest*, está disponível de acordo com a documentação. Na fig. I.4 verifica-se

[1] O bootloader é um programa presente em todos os dispositivo que usem um sistema operativo, sendo responsável por gerir a inicialização do sistema [62]

o diagnóstico final apresentado pela aplicação indicativo de que tudo está a funcionar corretamente.

3.3.2 Ambiente de desenvolvimento

A OP-TEE é baseada numa versão reduzida do sistema operativo Buildroot [2] que neste caso apresenta poucos pacotes do sistema instalados tornando, por exemplo, o acesso remoto difícil. Uma das soluções passa pela possibilidade de adicionar pacotes à OP-TEE. Utilizando a documentação, foram adicionadas funcionalidades que trouxeram diversas facilidades ao processo de desenvolvimento.

Lista de alguns dos pacotes mais importantes que foram adicionados:

SQLite Biblioteca de manipulação de base de dados usando a linguagem em SQL. Gratuita e não necessita de servidor [64].

VIM Editor de texto muito configurável para sistemas Unix [65].

PIGPIO Biblioteca que permite acesso às funções do GPIO do RPi3.

TAR Compressor de ficheiros típico de distribuições Linux. Foi utilizado para realizar cópias de segurança complementares ao GitHub [66].

NTP Protocolo de sincronização do relógio. Utilizado para que o RPi3 mantenha a sua data e hora sempre atualizada [67].

Para proceder à instalação dos *packages* no sistema OPTEE procede-se da seguinte forma:

1. Editar o ficheiro /optee/build/common.mk.
2. Procurar a lista com as variável *BR2_PACKAGE* e adicionar uma nova variável por cada pacote que se pretenda adicionar ao sistema:
   a) BR2_PACKAGE_SQLITE ?= y
   b) BR2_PACKAGE_VIM ?=y
   c) BR2_PACKAGE_TAR ?=y
   d) BR2_PACKAGE_PIGPIO ?=y

[2] Ferramenta que auxilia e automatiza a criação de distribuições Linux para sistemas embutidos. Realiza a compilação cruzada do código para a arquitetura da placa para a qual se quer construir a distribuição.

e) BR2_PACKAGE_NTP ?=y

Sempre que adicionados novos *packages* de *software* é necessária a recompilação do sistema e repetir a transferência da OP-TEE para o cartão SD. Existem alguns serviços que se encontram na instalação base, nomeadamente:

- SSH
- SFTP
- ETHERNET

A instalação destes pacotes aumenta a produtividade na produção de código, depuração e testes da aplicação. O serviço SSH permite acesso ao servidor remoto onde foi desenvolvida a aplicação. Através de terminal ou com recurso ao IDE Visual Studio Code com extensão de acesso remoto SSH instalada passa a ser possível programar de forma mais eficiente e eficaz.

3.3.3 Compilação projeto

O sistema OP-TEE é compilado com a ferramenta de controlo de compilação *GNU make*. O ficheiro principal *Makefile* encontra-se na raiz do projeto, e um conjunto de outros ficheiros *sub.mk* em diversas pastas do sistema. É no *Makefile* que são adicionados, por exemplo, os módulos a compilar e as bibliotecas a incluir durante a compilação da aplicação *host* e TA. É um sistema recursivo que tem como principal função aplicar as regras de compilação da *framework*. Mais informações podem ser encontradas na documentação bem como na página do GitHub referente ao projeto OP-TEE [68].

O projeto OP-TEE disponibiliza uma página no GitHub de forma a facilitar a instalação e configuração inicial da *framework*. Nesta página são disponibilizados todos os ficheiros e os *makefiles* necessários para preparação de toda a *framework* [69].

A empresa Sequitur Labs [70] foi a responsável pelo desenvolvimento do *porting* inicial para RPI3 bem como alterações ao U-Boot, ao *Trusted Firmware A*[3] e *kernel Linux*.

Parte do sucesso das operações de compilação passa pela correta instalação e configuração das bibliotecas necessárias. As pastas do *optee client* a ter em conta estão localizadas em:

[3]Referência de implementação do *software* do mundo seguro para processadores das arquiteturas ARMv7-A e ARMv8-A.

```
$PASTA_PROJETO/optee_client/build/libteec
$PASTA_PROJETO/optee_client/build/libckteec
```

A utilização de um *script bash* [4] facilita o processo de compilação da aplicação cliente e TA. Na pasta *host* e TA é usado o ficheiro *bash* com o código que consta na fig. I.5. Este ficheiro é importante visto que inicializa as variáveis de sistema necessárias para o processo de compilação.

A conclusão do processo deverá ser idêntico ao da fig. I.6. Em caso de erro ou dificuldade em executar a compilação é conveniente a «limpeza» do projeto com a instrução *make clean*.

As aplicações de teste disponibilizadas pela *framework* têm assinaturas Universally Unique Identifier (UUID) [5] que não devem ser utilizadas nas aplicações em produção. Por esta razão foi gerada nova UUID para a aplicação desenvolvida [71]. O novo UUID pode ser gerado de diversas formas e deverá posteriormente ser atualizada em:

```
$PASTA_PROJETO/include/$NOME_APLICACAO_ta.h
```

Na linha 35:

```
#define TA_$NOME_APLICACAO_UUID
```

Atualizar também no ficheiro:

```
$PASTA_PROJETO/ta/Makefile
```

Na linha 4 (BINARY=).

O ficheiro da aplicação cliente (gerado na pasta host) é colocado no cartão SD em /usr/bin. O ficheiro da TA, extensão «.ta» (ex: a734eed9d6a14244aa507c99719e7b7b.ta) deve ser adicionado à pasta /lib64/optee_armtz.

3.3.4 Aplicação

O projeto OP-TEE assenta toda a sua *framework* na linguagem de programação C. As funções, variáveis, constantes e restantes componentes necessários foram organizadas em diferentes ficheiros consoante a sua função, contribuindo para o melhor desenvolvimento de código, depuração, processo de compilação e futura compreensão.

[4] GNU Bash é um interpretador de comandos que permitem a execução de sequências de comandos da shell.

[5] Chave única com 128 *bit* gerada com o MAC do equipamento e um *timestamp*. Utilizada para identificação em sistemas de informação.

As funções da aplicação foram organizadas e os ficheiros mais importantes são:

main.c Estrutura principal da aplicação.

opteerasp.h Contém as principais definições de variáveis e constantes. Agregador de todos as *includes* externos dos restantes ficheiros de código.

ler_dados_sensor.c Funções de manipulação da base de dados e do funcionamento dos sensores.

aes.c Funções, variáveis e restante código relacionado com as operações da cifra AES.

sha256.c Funções, variáveis e restante código relacionado com as operações da *hash* SHA256.

compila64.sh Ficheiro responsável pela compilação do projeto cliente.

Makefile Responsável pela integração dos *packages* integrados no sistema. Configurações das bibliotecas entre outros.

apptest2_ta.c Funções críticas da TA

A primeira função a ser executada é a **verificação de ficheiros**. O seu objetivo é garantir a consistência e integridade dos ficheiros de suporte aos dados gerados e tratados pela aplicação. É verificada a existência do ficheiro da base de dados de suporte à indexação do *blockchain*. Utiliza-se uma base de dados do tipo *SQLite* por ser de tamanho reduzido e fácil integração na OP-TEE. Não é um SGDB e é *software* usado em sistema embutidos.

De seguida é realizada a **inicialização de variáveis** necessárias para uso das funções da API do cliente de acordo com o recomendado tendo como função gerir a comunicação entre os dois mundos:

```
1  TEEC_Result res;
2  TEEC_Context context;
3  TEEC_Session session;
4  uint32_t return_origin = 0;
```

A aplicação do **mundo normal inicia a ligação com a TA** através da função *iniciar_TEEC()* estabelecendo um contexto e uma sessão. Se a ligação for aceite a função retornará da TA, com *TEEC_SUCCESS*, caso contrário devolve o erro da falha.

O próximo passo passa por **informar a TA qual a função que se pretende executar**. Neste presente caso será iniciar a função de cifragem *AES* com operação de codificação (*ENCODE*). A função *AES* foi configurada para trabalhar com blocos de 4096 *bit* e utilizar uma chave de 128 *bit*.

A função *enviar_chave_AES()* é responsável pelo **envio da chave** a utilizar na cifra *AES* invocando a sessão já estabelecida nos passos anteriores.

A **recolha de dados** não é considerada uma operação crítica e, por essa razão é executada no mundo normal. Para a ligação dos sensores que vão recolher os dados foi necessário dotar a OP-TEE de acesso ao GPIO do RPi3. Foi utilizada a biblioteca *pigpio* [72] a qual foi integrada como explicado na subsecção 3.3.2. A função *ler_dados_sensor()* lê os dados, constituídos pela atual temperatura e humidade do ar, e grava-os numa estrutura do tipo DadosSensor.

```
1 struct DadosSensor {
2  int id;
3  long int timestamp;
4  float temperatura;
5  float humidade;
6 };
```

Ainda na mesma função consulta-se na base de dados o *index* (ID) do *blockchain* e regista-se um *timestamp* do sistema na estrutura DadosSensor. Os dados da temperatura e humidade, o *timestamp* e o ID estão prontos para a fase de cifragem. A função envia também ***output* para a *prompt*** sobre os dados recolhidos.

Os dados recolhidos e organizados estão prontos para serem cifrados por forma a garantir a sua integridade e confidencialidade. Essa tarefa está a cargo da função *cifrar_AES()* que, continuando a utilizar a mesma sessão, **envia os dados para a TA onde cifrará os mesmos**.

Na aplicação está implementada também uma função de decifragem que tem como propósito validar o processo de cifragem.

Para a construção de um *blockchain* foi implementada a função *sha256()* que **gera uma *hash SHA256*** de um determinado bloco de dados como forma a garantir que o *blockchain* mantêm a sua integridade. Esta função é interna à *framework* e será executada na TA por ser um processo considerado crítico.

As operações críticas foram concluídas e as comunicações com a TA podem por isso ser terminadas. Existem duas funções que realizam essa operação **terminando a sessão e o contexto** criados no início do processo:

```
1     TEEC_CloseSession(&session);
2     TEEC_FinalizeContext(&context);
```

O funcionamento da aplicação poderá ser resumido no seguinte linha de execução:

1. Verificação do estado e existência dos ficheiros de dados (base de dados e *blockchain*);

a) Inicialmente é estabelecida comunicação com a Base de Dados SQLite que terá uma função de *index* do *blockchain*: bd_ligação();

b) Se necessário inicia uma nova base de dados e *blockchain*;

c) Verificada a disponibilidade e estado da tabela que guarda os túplos: db_criaTabela();

d) Se for uma primeira execução gera o primeiro bloco: *genesis*;

2. Existe um ciclo que permitirá configurar o número de leituras do sensor a serem realizadas:

a) Guardado um *timestamp* do sistema que será utilizado na criação de um novo registo na BD: db_insere();

b) É verificado o número de registo (ID) a ser utilizado na próxima transação: db_consulta_id();

c) São recolhidos os dados do sensor de temperatura e humidade: ler_dados_sensor();

d) Os dados recolhidos do sensor são guardados numa estrutura de dados juntamente com o ID e *timestamp*.

e) De seguida a estrutura é enviada para o mundo seguro onde é cifrada utilizando a cifra AES: cifrar_AES();

f) É gerada uma *hash* da estrutura cifrada no ponto anterior utilizando SHA256: sha256()

g) É gerada uma *hash* da estrutura cifrada do bloco anterior utilizando igualmente SHA256: sha256(le_registo_anterior(ultimo_id1));

h) Os dados são gravados no *blockchain*: grava_dados();

Os processos que invocam a execução de funções do mundo seguro (TEE) executam em concorrência temporal com o código do mundo normal. Por esta razão será produzido *output* na *prompt* em simultâneo causando sobreposição ao *output* do mundo normal. Este constrangimento levou à necessidade de adicionar pausas no código do mundo normal para controlo do seu *output*. A aplicação é executada no RPi3 e apresentou sempre um comportamento estável e rápido.

3.4 Conclusão

Está disponível para dispositivos IoT, nomeadamente RPi3, a TrustZone que implementa um modelo de zonas seguras com a modificação do sistema operativo. Oferece uma TEE

para execução de funções e tratamento de dados críticos no que diz respeito à sua confidencialidade, integridade e disponibilidade. Outras utilizações passam pela execução de processos de autenticação, transações, gestão de DRM, entre outros.

O nível de segurança de um sistema não deve estar assente apenas na qualidade do *software* que executa. Desenvolver uma melhor arquitetura global do sistema computacional é crítico. Nesta ótica, o *hardware* desempenha um papel fundamental ao permitir elevar as barreiras de proteção contra possíveis ataques externos ao oferecer ferramentas que permitem:

- Proteção à corrupção do sistema operativo possibilitando *boot* seguro e encriptação da sua imagem.
- Execução segura de código crítico por isolamento: em ciclos de processamento, memória particionada, controlo de acesso aos periféricos e *bus* do sistema.
- Disponibilizando bibliotecas certificadas.

É possível melhorar a segurança informática separando a execução de código crítico em duas zonas:

- A zona normal, com acesso comum aos recursos do sistema operativo e bibliotecas gerais: *input* dados dos sensores, funções gerais da aplicação e gravação ficheiros.
- A zona segura, com acesso controlado e restrito a bibliotecas críticas em termos de segurança: cifragem e *hash*.

Este capítulo apresentou a arquitetura da aplicação desenvolvida como prova de conceito de um sistema utilizando para o efeito um dispositivo computacional RPi3 conhecido pelas suas características e capacidades IoT. O *hardware* do RPi3 não implementa todos os componentes da TrustZone, nomeadamente, *boot* seguro, partição de memória e controlo de acesso aos periféricos. A OP-TEE permite tirar partido da tecnologia TrustZone no RPi3 apenas para fins de prototipagem e uso académico pois, devido às limitações na implementação, o sistema nunca se garante seguro.

Realizou-se uma aplicação demonstrativa de conceito, que permite a um dispositivo IoT, adquirir dados críticos e de modo mais seguro. São utilizadas as funções de cifragem da *framework* OP-TEE que garantem a segurança dos dados e dos processos de cifragem envolvidos. Como complemento de segurança foi implementado o armazenamento de toda a informação gerada pela aplicação em *blockchain* visando a sua segurança.

Outro objetivo foi realizar uma descrição pormenorizada de todas as fases de instalação, configuração, compilação e produção de uma aplicação baseada em tecnologia TrustZone. Visa apoiar futuros projetos a ultrapassarem de forma mais eficaz as dificuldades e constrangimentos iniciais do uso da tecnologia, podendo centrar o foco no desenvolvimento da solução de *software*.

Capítulo 4

Conclusões

A tecnologia TrustZone prova ser uma solução viável para aplicação de um Trusted Execution Environment (TEE). A arquitetura da ARM oferece o *software* e *hardware* que reforça a separação de execução de funções e dados críticos.

As TEE permitem uma separação lógica e física na execução de código levando a que processos críticos não partilhem recursos com o sistema operativo ou restante código, potencialmente inseguro. A segurança de um sistema normalmente assenta a sua segurança na qualidade do seu *software*. Uma mudança de paradigma levou a introduzir o *hardware* como parte integrante dessa equação. Esta ideia é implementada pelo conceito de uma TEE na tecnologia ARM TrustZone. As TEE oferecem um nível de segurança mais elevado do que uma TCB.

Ainda que a TrustZone ofereça um nível elevado de segurança na sua arquitetura de SoC e processadores existem pontos a melhorar:

- Ainda que os mais recentes processadores seja *multicore*, a TrustZone apenas permite a passagem ao mundo seguro de apenas uma tarefa em simultâneo. Obriga a paragem dos restantes processos a decorrer no mundo normal levando a perdas de desempenho substanciais.

- Existência de várias vulnerabilidades documentadas, muitas delas devido à aplicação incompleta da arquitetura por parte dos fabricantes, que nem sempre implementam todos os componentes previstos ou a implementação é defeituosa. Os avanços tecnológicos e técnicos, aliados a uma grande criatividade, aumentam a qualidade e superfície de ataque abrangendo diversos *layers* do sistema.

- A documentação disponível requer conhecimentos com nível técnico mais elevado para a sua boa compreensão. São disponibilizadas poucas fontes de suporte e pouco

eficazes.

- O *hardware* disponível com capacidades da TrustZone pode apresentar obstáculos devido à pouca ou nenhuma documentação técnica sobre a implementação realizada pelo fabricante, levando ao investimento de tempo em testes que podem revelar-se infrutíferos. Pesa também o facto de existir uma constante evolução das placas.

- O ambiente de desenvolvimento apresenta enumeras dificuldades e grandes desafios na produção de aplicações para os menos experientes. A ARM parece estar alerta para este facto pelo exposto de seguida.

Mais recentemente a ARM disponibilizou um sistema integrado de desenvolvimento rápido de software - Total Solutions for IoT - que permite o rápido desenvolvimento de software em ambiente virtual. Uma das ferramentas disponível é o ARM Virtual Hardware que oferece modelos precisos dos SoC mesmo antes de estarem disponíveis em silício. Esta plataforma apresenta enumeras vantagens no *time-to-Market* [73]. Esta solução poderá vir a colmatar as dificuldades com o ambiente de desenvolvimento sentido no presente trabalho de investigação.

4.1 Trabalho futuro

No decorrer dos trabalhos conducentes à elaboração desta dissertação surgiram algumas ideias para trabalho futuro. Apresentam-se, a finalizar este documento, algumas das que se consideram mais interessantes.

A segurança dos dados em sistemas usados na área da Internet das Coisas é um área em que se está a produzir muitos contributos novos. A adoção de tecnologias *blockchain* permite elevar o nível de segurança dos dados e uma pequena demonstração de conceito é apresentada na dissertação. A utilização dos ambientes de execução segura combinada com propostas mais abrangentes de sistemas *blockchain* é trabalho futuro que se entende de grande relevância técnica.

O sistema apresentado foi desenvolvido para a versão 3 do Raspberry Pi. A versão 4 é a mais atual e prevê-se que brevemente seja suportada convenientemente pelo projeto OP-TEE. Há novo *hardware* baseada em arquiteturas RISC V que vão ter maior importância em IoT. Há, portanto, muito trabalho de desenvolvimento de aplicações para novas plataformas de *hardware*.

Os sistemas suportam a programação *multicore* quer em CPU quer em GPU, e é um caminho que se pode explorar.

A finalizar, há a programação, que já se encontra suportada em bibliotecas do projeto OP-TEE, de aplicações com a linguagem de programação `Rust` que eleva grandemente a segurança das aplicações.

Bibliografia

[1] ARM, "ARM Cortex-A Series Programmer's Guide," *ARM*, p. 421, 2011. (citado nas págs. ix e 11)

[2] Intel. Enhanced security features for applications and data in-use. [*Online*]. Disponível: https://software.intel.com/content/dam/develop/public/us/en/documents/intel-sgx-product-brief-2019.pdf (citado nas págs. ix, 16 e 17)

[3] H.-F. Security. The first tee for risc-v. [*Online*]. Disponível: https://hex-five.com/multizone-security-sdk (citado nas págs. ix e 19)

[4] IoT Security Foundation, "IoT Security Compliance Framework," 2016, release 1.0. [*Online*]. Disponível: https://www.iotsecurityfoundation.org/wp-content/uploads/2016/12/IoT-Security-Compliance-Framework.pdf (citado na pág. 2)

[5] T. Alves e D. Felton, "Trustzone: Integrated Hardware and Software Security," *Technology In-Depth*, Janeiro 2004. (citado nas págs. 2 e 9)

[6] S. M. P. Keyur K Patel. Internet of things-iot: Definition, characteristics, architecture, enabling technologies, application & future challenges. [*Online*]. Disponível: http://www.opjstamnar.com/download/Worksheet/Day-110/IP-XI.pdf (citado na pág. 3)

[7] J. P. Cohen, R. S. Ramos, M. Santos, e J. F. Chaves, "TrustFrame, a Software Development Framework for TrustZone-enabled Hardware," in *TrustFrame, a Software Development Framework for TrustZone-enabled Hardware.* Instituto Superior Técnico Lisboa, 2016. (citado nas págs. 5, 15 e 16)

[8] C. Almeida, J. C. Martins, J. M. Santos, e J. Jasnau Caeiro, "Smart lysimeter with crop and environment monitoring," in *Internet of Things. Technology and Applications*, L. M. Camarinha-Matos, G. Heijenk, S. Katkoori, e L. Strous, Eds. Cham: Springer International Publishing, 2022, pp. 48–63. (citado na pág. 5)

[9] G. Oliveira, C. Almeida, J. M. Santos, J. C. Martins, e J. Jasnau Caeiro, "Iot lysimeter system with enhanced data security," in *CONTROLO 2022*, L. Brito Palma, R. Neves-Silva, e L. Gomes, Eds. Cham: Springer International Publishing, 2022, pp. 119–129. [*Online*]. Disponível: https://doi.org/10.1007/978-3-031-10047-5_11 (citado nas págs. 5 e 22)

[10] R. M. Davis, "Evolution of Computers and Computing," *Science*, vol. 195, n. 4283, pp. 1096–1102, 1977, publisher: American Association for the Advancement of Science _eprint: https://science.sciencemag.org/content/195/4283/1096.full.pdf. [*Online*]. Disponível: https://science.sciencemag.org/content/195/4283/1096 (citado na pág. 7)

[11] Intel. The evolution of a revolution. [*Online*]. Disponível: https://download.intel.com/pressroom/kits/IntelProcessorHistory.pdf (citado na pág. 7)

[12] T. Mens, "Introduction and roadmap: History and challenges of software evolution," *University of Mons-Hainaut*, 01 2008. (citado na pág. 8)

[13] G. McGraw, "Software security," *IEEE Security Privacy*, vol. 2, n. 2, pp. 80–83, 2004. (citado na pág. 8)

[14] R. L. Glass, "Two mistakes and error-free software: A confession," *IEEE Software*, vol. 25, n. 4, pp. 96–96, 2008. (citado na pág. 8)

[15] G. D. et al, "Hardfails: Insights into software-exploitable hardware bugs," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Agosto 2019, pp. 213–230. [*Online*]. Disponível: https://www.usenix.org/conference/usenixsecurity19/presentation/dessouky (citado na pág. 9)

[16] S. Mohanty, V. Thotakura, e M. Ramkumar, "An efficient trusted computing base for manet security," *Journal of Information Security*, vol. 5, pp. 192–206, 11 2014. (citado na pág. 9)

[17] S. J. Murdoch, "Introduction to Trusted Execution Environments (TEE) – IY5606," *University of Cambridge*, p. 34, 2020. (citado na pág. 9)

[18] S. Pinto e N. Santos, "Demystifying Arm TrustZone: A Comprehensive Survey," *ACM Computing Surveys*, vol. 51, n. 6, pp. 1–36, Fevereiro 2019. [*Online*]. Disponível: https://dl.acm.org/doi/10.1145/3291047 (citado nas págs. 9 e 12)

[19] A. Cortex-A. Arm cortex-a series programmer's guide for armv8-a. [*Online*]. Disponível: https://developer.arm.com/documentation/den0024/a/ (citado na pág. 9)

[20] A. R-Profile. R-profile architectures. [*Online*]. Disponível: https://developer.arm.com/architectures/cpu-architecture/r-profile (citado na pág. 10)

[21] A. M-Profile. M-profile architectures. [*Online*]. Disponível: https://developer.arm.com/architectures/cpu-architecture/m-profile (citado na pág. 10)

[22] I. Fowler, "TrustZone technology for ARMv8-M," *ARM Limited or its affiliates*, p. 28, 2017. (citado na pág. 10)

[23] ARM, "ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition," *ARM*, p. 2720, 2000. (citado na pág. 10)

[24] ——, "ARM Security Technology Building a Secure System using TrustZone Technology," *ARM*, p. 108, 2009. (citado na pág. 11)

[25] A. Armv8-A. Arm architecture reference manual for armv8-a. [*Online*]. Disponível: https://developer.arm.com/documentation/ddi0553/latest (citado na pág. 11)

[26] Counterpoint. Amd ready to shake up the mobile gpu market. [*Online*]. Disponível: https://www.counterpointresearch.com/amd-ready-shake-mobile-gpu-market/ (citado na pág. 12)

[27] S. D. Yalew, S. Haridi, M. Correia, M. Freire, KTH, e Skolan för elektroteknik och datavetenskap (EECS), *Mobile Device Security with ARM TrustZone*. KTH School of Electrical Engineering, 2018, oCLC: 1059280684. [*Online*]. Disponível: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-236975 (citado na pág. 13)

[28] H. Sun, K. Sun, Y. Wang, J. Jing, e H. Wang, "Trustice: Hardware-assisted isolated computing environments on mobile devices," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 367–378. (citado na pág. 13)

[29] P. S. Zone. Embedded security specialist joins trustzone ready. [*Online*]. Disponível: https://www.prosecurityzone.com/News_Detail_Embedded_security_specialist_joins_trustzone_ready_20586.asp (citado na pág. 13)

[30] Verimatrix. Verimatrix driving trust. [*Online*]. Disponível: https://www.verimatrix.com (citado na pág. 13)

[31] V. DRM. Trustzone-ready drm technology to telechips for secure set-top boxes. [*Online*]. Disponível: https://www.verimatrix.com/press-releases/inside-secure-and-solacia-team-bring-trustzone-ready-drm-technology-telechips-secure (citado na pág. 13)

[32] V. solutions for mobile devices e tablets powered by QUALCOMM's. https://www.verimatrix.com/press-releases/inside-secures-vice-president-be-panel-speaker-qualcommr-uplinqr2014-mobile. [*Online*]. Disponível: https://www.verimatrix.com/press-releases/inside-secure-and-solacia-team-bring-trustzone-ready-drm-technology-telechips-secure (citado na pág. 13)

[33] V. TrustZone. Embedded security specialist joins trustzone ready. [*Online*]. Disponível: https://www.verimatrix.com/search?keywords=trustzone (citado na pág. 13)

[34] D. Cerdeira, N. Santos, P. Fonseca, e S. Pinto, "Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems," *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1416–1432, 2020. (citado na pág. 14)

[35] N. V. Database. Cve-2015-6639 detail. [*Online*]. Disponível: https://nvd.nist.gov/vuln/detail/CVE-2015-6639 (citado na pág. 14)

[36] ——. Cve-2016-2431 detail. [*Online*]. Disponível: https://nvd.nist.gov/vuln/detail/CVE-2016-2431 (citado na pág. 14)

[37] D. Rosenberg. Qsee trustzone kernel integer overflow. [*Online*]. Disponível: https://wikileaks.org/sony/docs/05/docs/Hacks/us-14-Rosenberg-Reflections-On-Trusting-TrustZone-WP.pdf (citado na pág. 14)

[38] D. Shen. Exploiting trustzone on android. [*Online*]. Disponível: https://www.blackhat.com/docs/us-15/materials/us-15-Shen-Attacking-Your-Trusted-Core-Exploiting-Trustzone-On-Android-wp.pdf (citado na pág. 14)

[39] Y. Chen. Downgrade attack on trustzone. [*Online*]. Disponível: https://arxiv.org/ftp/arxiv/papers/1707/1707.05082.pdf (citado na pág. 14)

[40] J. Guilbon. Attacking the arm's trustzone. [*Online*]. Disponível: https://blog.quarkslab.com/attacking-the-arms-trustzone.html (citado na pág. 14)

[41] Q. Blog. Vulnerabilities in trustzone itself. [*Online*]. Disponível: https://blog.quarkslab.com/introduction-to-trusted-execution-environment-arms-trustzone.html (citado na pág. 14)

[42] M. Gross, N. Jacob, A. Zankl, e G. Sigl, "Breaking trustzone memory isolation through malicious hardware on a modern fpga-soc," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, 11 2019, pp. 3–12. (citado na pág. 14)

[43] P. Qiu, D. Wang, Y. Lyu, e G. Qu, "Voltjockey: Breaching trustzone by software-controlled voltage manipulation over multi-core frequencies," in *ACM SIGSAC Conference on Computer and Communications Security*, 11 2019, pp. 195–209. (citado na pág. 14)

[44] B. Lapid e A. Wool, "Cache-attacks on the arm trustzone implementations of aes-256 and aes-256-gcm via gpu-based analysis," Cryptology ePrint Archive, Report 2018/621, 2018, https://eprint.iacr.org/2018/621. (citado na pág. 14)

[45] K. Ryan, "Hardware-backed heist: Extracting ecdsa keys from qualcomm's trustzone," in *Computer Science*, 11 2019, pp. 181–194. (citado na pág. 14)

[46] S. Bukasa, R. Lashermes, H. Bouder, J.-L. Lanet, e A. Legay, *How TrustZone Could Be Bypassed: Side-Channel Attacks on a Modern System-on-Chip*. Springer Cham, 06 2018, pp. 93–109. (citado na pág. 14)

[47] ARM. Tee reference documentation. [*Online*]. Disponível: https://www.arm.com/why-arm/technologies/trustzone-for-cortex-a/tee-reference-documentation (citado na pág. 15)

[48] Commissum. Are there still security concerns with amd's platform security processor? [*Online*]. Disponível: https://commissum.com/blog-articles/security-concerns-with-amds-psp (citado na pág. 17)

[49] CTS. Severe security advisory on amd processors. [*Online*]. Disponível: https://safefirmware.com/amdflaws_whitepaper.pdf (citado na pág. 17)

[50] AMD. Amd memory guard. [*Online*]. Disponível: https://www.amd.com/system/files/documents/amd-memory-guard-white-paper.pdf (citado na pág. 17)

[51] S. Mofrad, F. Zhang, S. Lu, e W. Shi, "A comparison study of intel sgx and amd memory encryption technology," in *A comparison study of intel SGX and AMD memory encryption technology*, 06 2018, pp. 1–8. (citado na pág. 17)

[52] AMD. Strength at the core. [*Online*]. Disponível: https://www.amd.com/en/technologies/pro-security (citado na pág. 17)

[53] D. Patterson e A. Waterman, *Guia prático RISC-V*. Strawberry Canyon LLC, 2019. (citado na pág. 18)

[54] P. Security. A detailed comparison of risc-v to arm's trustzone. [*Online*]. Disponível: https://www.youtube.com/watch?v=CQ7D_V1MpPY (citado na pág. 19)

[55] J. Winter, "Experimenting with arm trustzone – or: How i met friendly piece of trusted hardware," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012, pp. 1161–1166. (citado na pág. 22)

[56] Samsung, "Improve your enterprise work flow with Samsung Knox," 2020. [*Online*]. Disponível: https://docs.samsungknox.com (citado na pág. 23)

[57] A. Developers, "Sistema Android Keystore | Desenvolvedores Android," Maio 2020. [*Online*]. Disponível: https://developer.android.com/training/articles/keystore?hl=pt (citado na pág. 23)

[58] OP-TEE, "Open Portable Trusted Execution Environment," 2020. [*Online*]. Disponível: https://www.op-tee.org/ (citado na pág. 23)

[59] "Raspberry Pi 3 — OP-TEE documentation documentation." [*Online*]. Disponível: https://optee.readthedocs.io/en/latest/building/devices/rpi3.html (citado na pág. 23)

[60] OP-TEE. Globalplatform api. [*Online*]. Disponível: https://optee.readthedocs.io/en/latest/architecture/globalplatform_api.html (citado na pág. 23)

[61] G. Platform. Tee internal core api specification. [*Online*]. Disponível: https://globalplatform.org/specs-library/tee-internal-core-api-specification/ (citado na pág. 23)

[62] Tecnoblog. O que é bootloader? [*Online*]. Disponível: https://tecnoblog.net/313227/o-que-e-bootloader/ (citado na pág. 27)

[63] Github. U-boot. [*Online*]. Disponível: https://github.com/u-boot/u-boot (citado na pág. 27)

[64] SQLite. Sqlite website. [*Online*]. Disponível: https://www.sqlite.org/index.html (citado na pág. 28)

[65] Vim. Vim the ubiquitous text editor. [*Online*]. Disponível: https://www.vim.org/ (citado na pág. 28)

[66] TAR. Gnu tar. [*Online*]. Disponível: https://www.gnu.org/software/tar/manual/html_node/Standard.html (citado na pág. 28)

[67] NTP. Ntp: The network time protocol. [*Online*]. Disponível: http://www.ntp.org/ (citado na pág. 28)

[68] OP-TEE. Build system. [*Online*]. Disponível: https://github.com/ForgeRock/optee-os/blob/master/documentation/build_system.md (citado na pág. 29)

[69] ——. Op-tee build.git. [*Online*]. Disponível: https://github.com/OP-TEE/build (citado na pág. 29)

[70] S. Labs. Sequitur labs webpage. [*Online*]. Disponível: https://www.sequiturlabs.com/ (citado na pág. 29)

[71] D. Corner. Online uuid generator. [*Online*]. Disponível: https://www.uuidgenerator.net/ (citado na pág. 30)

[72] P. library. pigpio library. [*Online*]. Disponível: https://abyz.me.uk/rpi/pigpio/pdif2.html (citado na pág. 32)

[73] ARM. Arm total solutions. [*Online*]. Disponível: https://www.arm.com/solutions/iot/total-solutions-iot?utm_source=google&utm_medium=cpc&utm_campaign=2021_embiot-lowp_mk02_1000heads_na_na_na&utm_content=searchad-554190066686&utm_term=iot%20software&gclid=Cj0KCQiA7oyNBhDiARIsADtGRZYt0r0H51beVTgsDwyhRPp4XDO0ysPWVjhzCwQbRM_vye0SN-1KAjMaAkvWEALw_wcB (citado na pág. 38)

Anexos

Anexo I

Imagens referenciadas no decorrer dos capítulos:

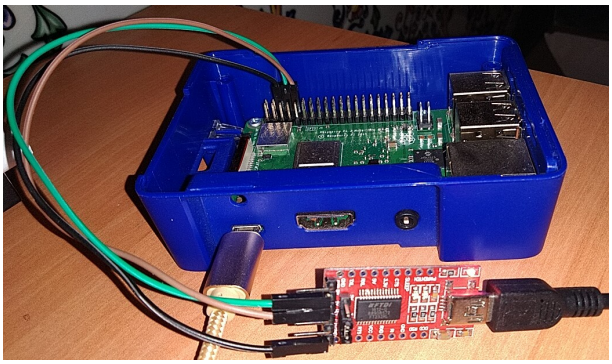

Figura I.1: Rpi3 B+ c/ ligação UART

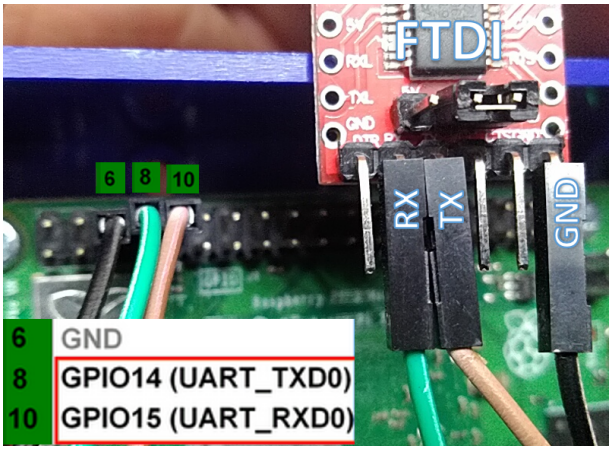

Figura I.2: Rpi3 e FTDI pinout

```
Welcome to Buildroot, type root or test to login
buildroot login: [    3.383969] dwc_otg_handle_wakeup_detected_intr lxstate =
 2
[    3.917719] usb 1-1.1.1: new high-speed USB device number 4 using dwc_otg
[    4.018397] usb 1-1.1.1: New USB device found, idVendor=0424, idProduct=78
00
[    4.030433] usb 1-1.1.1: New USB device strings: Mfr=0, Product=0, SerialN
umber=0
[    4.376292] libphy: lan78xx-mdiobus: probed
[    4.696477] random: crng init done

Welcome to Buildroot, type root or test to login
buildroot login: root
#
```

Figura I.3: Login *framework* (Buildroot)

```
regression_8102 OK
regression_8103 OK
+-----------------------------------
24750 subtests of which 0 failed
91 test cases of which 0 failed
0 test cases were skipped
TEE test application done!
#
```

Figura I.4: Resultado xtest com sucesso

```
1  #!/bin/bash
2  make CROSS_COMPILE=arm-linux-gnueabihf- TEEC_EXPORT=/usr/local --no-builtin-variables
3
```

Figura I.5: Ficheiro de compilação do cliente

```
imfree@imfree-VPCEB2M1E:~/optee/optee_examples/acipher/host$ ./compila.sh
arm-linux-gnueabihf-gcc -Wall -I../ta/include -I./include -I/usr/local/include -
c main.c -o main.o
arm-linux-gnueabihf-gcc -o optee_example_acipher main.o -lteec -L/usr/local/lib
imfree@imfree-VPCEB2M1E:~/optee/optee_examples/acipher/host$ cd ..
imfree@imfree-VPCEB2M1E:~/optee/optee_examples/acipher$ cd ta
imfree@imfree-VPCEB2M1E:~/optee/optee_examples/acipher/ta$ ./compila.sh
  CC      acipher_ta.o
  CC      user_ta_header.o
  AS      ta_entry_a32.o
  CPP     ta.lds
  GEN     dyn_list
  LD      a734eed9-d6a1-4244-aa50-7c99719e7b7b.elf
  OBJDUMP a734eed9-d6a1-4244-aa50-7c99719e7b7b.dmp
  OBJCOPY a734eed9-d6a1-4244-aa50-7c99719e7b7b.stripped.elf
  SIGN    a734eed9-d6a1-4244-aa50-7c99719e7b7b.ta
imfree@imfree-VPCEB2M1E:~/optee/optee_examples/acipher/ta$
```

Figura I.6: Resultado do processo de compilação