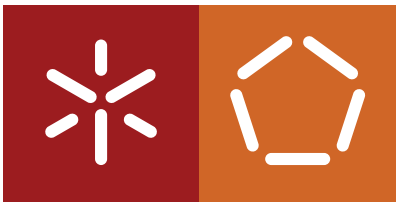
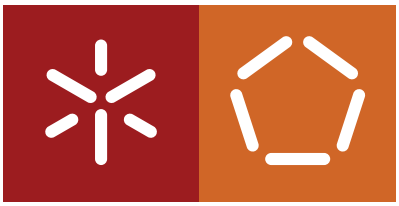


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Hugo Fernandes Matias

Suporte wearable para Mobile Documents (mDoc) ISO/IEC 18013-5

October 2023



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Hugo Fernandes Matias

Suporte wearable para Mobile Documents (mDoc) ISO/IEC 18013-5

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Dissertação supervisionada por
João Marco Cardoso da Silva
Vítor Francisco Fonte

October 2023

AGRADECIMENTOS

Aos meus pais por todo o apoio desde que iniciei esta jornada e pela oportunidade de poder formar-me na Universidade do Minho.

Aos meus orientadores, João Marco Silva e Vítor Fonte, pela disponibilidade, conhecimento, e inspiração que me transmitiram durante o meu percurso académico e elaboração desta dissertação.

Aos meus amigos, que me acompanham nesta jornada desde o primeiro dia, e que me apoiaram sempre que o necessitei.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio, nem a qualquer forma de utilização indevida ou falsificação de informações, ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais, declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

ABSTRACT

The popularization of mobile devices and the characteristic pervasiveness of internet access have led to increased interest in developing potentially relevant services for citizens. The various solutions for identifying people using digital devices are an example of this. Among these solutions is the ISO/IEC DIS 18013-5 standard. This standard published by the International Organization of Standardization (ISO) defines the technical requirements necessary for secure transmission, i.e. with guarantees of data integrity and authenticity, of identification attributes related to a driver's license. The interest to implement the standard ISO/IEC DIS 18013-5 has risen due to the increase in development and public interest in wearable devices, and the recent effort by the European Union to encourage the use of digital identification documents. Implementation of this standard is made easier by the existence of cross-development tools, e.g., Xamarin, Flutter, that allow code sharing between different wearable platforms. These tools serve as a catalyst for implementation while maintaining the functionality of the code.

Keywords

smartwatch, digital, ISO, identity, service

RESUMO

A popularização de dispositivos móveis e a pervasividade característica do acesso à internet, têm levado ao aumento do interesse em desenvolver serviços potencialmente relevantes aos cidadãos. As várias soluções para a identificação de pessoas recorrendo a dispositivos digitais, são exemplo disso. Dentre estas soluções, encontra-se a norma técnica *ISO/IEC DIS 18013-5*. Esta norma publicada pela *International Organization of Standardization (ISO)* define os requisitos técnicos necessários para que haja uma transmissão segura, ou seja, com garantias de confidencialidade, integridade e autenticidade de dados, de atributos de identificação relativos à carta de condução digital. Tendo em conta o crescente desenvolvimento e interesse público de dispositivos *wearable*, e o recente esforço da União Europeia para incentivar o uso de documentos digitais de identificação, seria de grande interesse a implementação da norma *ISO/IEC DIS 18013-5*. A implementação vêm a ser facilitada pela existência de ferramentas de desenvolvimento cruzado, *e.g., Xamarin*, *Flutter*, que permitem a partilha de código entre diferentes plataformas *wearable*. Estas ferramentas servem como catalisador da implementação, mantendo ao mesmo tempo a funcionalidade do código.

Palavras-chave smartwatch, ISO, autenticidade, identidade, serviço

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE LISTAGENS

ACRÓNIMOS

API	application programming interface
BLE	Bluetooth Low Energy
IDE	ntegrated Deve- lopment Environment
CBOR	Concise Binary Object Representation
COSE	CBOR Object Signing and Encryption
UUID	Universal Unique Identifier
ISO	International Organization for Standardization
mDoc	mobile Document
mDL	mobile Driving License
MSO	Mobile Security Object
OIDC	OpenID Connect
PCD	Proximity Coupling Device
CoC	Connection-Oriented Channel
PICC	Proximity Chip Card
PSM	Protocol Service Multiplexor
GATT	Generic Attribute Profile
HCCR	Handwriting Chinese Character Recognition
HTTP	Hypertext Transfer Protocol
TCP	Transmission Control Protocol
SDK	Software Development Kit
XML	Extensible Markup Language
JSX	JavaScript Syntax Extension
CDDL	Concise Data Definition Language
JSON	JavaScript Object Notation

ECDH Elliptic Curve Diffie-Hellman

ECKA-DH Elliptic Curve Key Agreement Algorithm – Diffie-Hellman

IV Initialization Vector

ECDSA Elliptic Curve Digital Signature Algorithm

HKDF Hash based Key Derivation Function

MEMS Micro-electromechanical systems

NFC Near Field Communication

PCL Portable Class Library

QR Quick Response

UI User Interface

URI Universal Resource Identifier

RFU Reserved for Future Use

1

INTRODUÇÃO

Com o avanço tecnológico dado em décadas recentes, cada vez se torna mais importante a familiarização e compreensão das consequências relevantes a esta evolução. Uma destes cosequências é a Identidade Digital, que sofreu uma evolução nos últimos tempos. Apesar de grande parte da população estar familiarizada com o conceito de identidade digital, é pequena a percentagem de indivíduos que conhecem as suas implicações e funcionalidades (Sullivan, 2011). Os problemas de autenticação, identidade e acesso fazem parte da identidade digital, e cada um apresenta as suas dificuldades individuais. Para ocorrer uma identificação é requerida uma autenticação de identidade, e acessos a dados devem ser precedidos por autenticação. A dificuldade destes problemas é acentuada pelas nuances que estes têm na aplicação da lei, em questões relacionadas com a justiça (Camp, 2004).

O aumento da relevância do conceito de identidade digital levou, recentemente, a que a Comissão Europeia propusesse uma *framework* (*i.e.,* The Digital ID Act (von der Leyen, 2020)) para uma identidade digital Europeia, que estará disponível para os seus cidadãos, residentes e negócios Europeus . Detentores desta identidade digital poderão aceder a serviços online, e provar a sua identidade a partir de um *click* de um botão nos seus telemóveis.

Tendo em conta a popularização do conceito de identidade digital, é natural que se torne relevante o desenvolvimento de serviços de identificação digital segura. Dentro das soluções propostas, destaca-se a norma técnica *ISO/IEC DIS 18013-5* (Iso, 2021), que define os requisitos técnicos e funcionais para a transmissão segura e robusta de atributos de identificação entre dispositivos digitais. Esta norma apresenta de forma concisa e explícita os requisitos necessários para a implementação da associação entre a carta de condução e um dispositivo digital móvel, e as transações que envolvem atributos de identificação.

Os dispositivos móveis englobam *wearables*, que se referem a componentes eletrónicas, ou computadores desenvolvidos para serem usados pelo corpo humano (Wanjari and Patil, 2016). Dentre estes *wearable* encontra-se o *smartwatch*. A recente evolução e crescimento dos *smartwatches* como dispositivos móveis do dia a dia leva a pensar que estes são uma invenção contemporânea, porém, este não é o caso. Em 1972, a fabricante suíça de relógios de pulso Hamilton Watch Company, desenvolveu o primeiro *smartwatch* chamado Pulsar que pavimentou o caminho para os seus sucessores. Pouco tempo depois, empresas como a *Seiko*, e posteriormente a *Samsung*, começaram a experimentar formas de introduzir mais conteúdo nos *smartwatches*. Em tempos

modernos, mais especificamente em 2012, o *smartwatch Pebble* quebrou recordes como o projecto *Kickstart* com mais dinheiro angariado, mostrando que havia uma procura real por estes dispositivos.

A implementação da norma técnica *ISO/IEC DIS 18013-5* em plataformas *wearable* (*i.e.,* WearOS, watch OS) é relevante tendo em conta o crescimento em popularidade dos *smartwatches*. A existência de plataformas de desenvolvimento cruzado auxiliam o desenvolvimento desta implementação permitindo a partilha do código entre as várias platformas *wearable*. Uma análise dos *smartwatches*, da norma técnica *ISO/IEC DIS 18013-5*, das ferramentas de *cross-platform* e da forma como estes três elementos se relacionam, é apresentada em capítulos posteriores.

1.1 MOTIVAÇÃO

A popularização de dispositivos móveis e a pervasividade do acesso à *internet*, potencializa o desenvolvimento de serviços que serão relevantes aos cidadãos.

O aumento da relevância do conceito de identidade digital e dos *smartwatches*, e a praticidade destes últimos, dão ênfase ao potencial que a implementação da norma *ISO/IEC DIS 18013-5* têm. A autenticação de identidade e troca de dados segura implicados neste *standard* são relevantes, tendo em conta a importância que a componente de praticidade têm no dia-a-dia do cidadão comum. Com esta norma, é possível não só confirmar a identidade de um indivíduo, como também trocar dados *mDL* (*mobile Driving License*) de forma selectiva a partir de um dispositivo que se encontra no pulso.

A implementação desta norma vem a ser facilitada por plataformas de desenvolvimento cruzado (*e.g.,* Xamarin, Flutter, React Native, etc.). Estas, permitem a partilha de código de lógica de negócio entre as várias plataformas *wearable* disponíveis (*e.g.,* WearOS, watchOS, Tizen OS, etc.), ao mesmo tempo que garantem a sua funcionalidade.

1.2 OBJECTIVOS

O principal objectivo do trabalho é o estudo de plataformas de desenvolvimento cruzado e a implementação da norma técnica *ISO/IEC DIS 18013-5* em plataformas *wearable*.

Numa fase inicial é necessária uma assimilação e compreensão dos requisitos técnicos e funcionais que constituem a norma técnica *ISO/IEC DIS 18013-5*. Ainda nesta fase, ocorre uma verificação de compatibilidade entre a norma ténica ISO/IEC DIS 18013-5 e os dispositivos móveis (*i.e., smartwatches*). Componentes relevantes à implementação da norma (*i.e.,* métodos de comunicação, estruturas de dados, criptografia) são consideradas nesta fase.

De seguida, é necessário realizar um estudo das plataformas de desenvolvimento cruzado disponíveis, de modo a selecionar a que apresenta as melhores condições para a implementação da norma em plataformas

wearable. A seleção tem em especial conta a qualidade do suporte a plataformas *wearable* que a plataforma de desenvolvimento cruzado apresenta.

Após selecionada a plataforma de desenvolvimento cruzado a utilizar, são implementados os requisitos funcionais e técnicos da norma *ISO/IEC DIS 18013-5* numa das plataformas *wearable* existentes. Esta implementação é feita com recurso à plataforma de desenvolvimento cruzado escolhida.

Num estado final, o objectivo principal proposto está estar realizado, ou seja, a norma *ISO/IEC DIS 18013-5* encontra-se implementada na plataforma *wearable* considerada, com a garantia da funcionalidade e segurança da implementação.

1.3 ORGANIZAÇÃO DO DOCUMENTO

O documento apresentado demonstra o que se desenvolveu no contexto da implementação de um suporte *wearable* para a norma técnica *ISO/IEC DIS 18013-5*. Inicialmente, são abordados os trabalhos relacionados, as tecnologias utilizadas, o estado de arte do problema e a arquitetura da solução proposta.

Posteriormente, são descritas as estruturas de dados e os mecanismos de segurança utilizados na implementação das funcionalidades apresentadas na norma técnica *ISO/IEC DIS 18013-5*.

De seguida, são apresentados os módulos da aplicação, as tecnologias de comunicação e troca de dados e os fluxos de ação da aplicação.

Por fim, os resultados que comprovam a viabilidade da prova de conceito são apresentados e, por último, são tiradas as conclusões relativamente ao trabalho efetuado e é abordado o trabalho futuro para a continuação do desenvolvimento do projeto atual.

2

TRABALHOS RELACIONADOS

Durante os últimos 15 anos, os *smartwatches* evoluíram bastante e oferecem agora muitas funcionalidades. Desde o lançamento do Pebble Smartwatch em 2012, os *smartwatches* têm recebido cada vez mais atenção por parte dos meios de comunicação populares (Cecchinato et al.,2015). Com este crescimento em popularidade, veio um crescimento tecnológico no desenvolvimento destes dispositivos. Tendo isto em conta, é considerada a seguinte definição de smartwatch (Cecchinato et al.,2015):

"Um dispositivo de pulso com poder computacional, que pode conectar-se com outros dispositivos através de conexão sem fios de pouco alcance; fornece notificações de alerta; coleciona dados através de uma gama de sensores e armazena-os; e têm um relógio integrado."

Os *smartwatches* hoje em dia, têm as mais diversas aplicações devido ao potencial funcional que apresentam. As aplicações dos *smartwatches* têm uma grande abrangência, desde a medicina moderna até à utilização no nosso dia-a-dia. Foi previamente demonstrado que os *smartwatches* podem ser usados para a deteção precoce da doença de *Lyme*, e que os seus sensores, que permitem captar dados sobre batimentos cardíacos e temperatura da pele, podem ser usados para detetar infecções respiratórias (Mishra et al.,2020). Mais recentemente, foram utilizados *smartwatches Fitbit* para a coleção de dados relevantes à detecção pré-sintomática de *COVID-19* (Mishra et al.,2020).

Numa vertente mais social e casual, *SmartHandwriting* (Zhang et al.,2020) é um sistema HCCR (Handwriting chinese character recognition) baseado em *smartwatches*. Este sistema, permite o reconhecimento de caracteres chineses tirando partido dos movimentos de pulso e do gesto de escrita. *SkinWatch* (Ogata and Imai,2015), uma aplicação *wearable* para *smartwatches*, permite aos utilizadores introduzir comandos e operações lineares através da deteção de deformações da pele perto do dispositivo *wearable*. Por fim, *AirText* (Gao et al.,2021), o primeiro método de entrada de texto com uma só mão, que alcança precisão e caligrafia prática no ar para *smartwatches* comerciais.

Os exemplos acima referidos mostram que a característica prática dos *smartwatches* não se encontra apenas no facto de serem dispositivos *wearable*. Esta praticidade também se reflete nas funcionalidades que estes apresentam, e nos potenciais usos destas funcionalidades.

2.1 ISO/IEC DIS 18013-5

A pervasividade do acesso à internet e o aumento de popularidade dos dispositivos móveis vêm dar relevo ao problema de identificação segura no meio digital. A norma técnica *ISO/IEC 18013-5* (Iso,2021), publicada pela *ISO (International Organization for Standardization)*, é uma das soluções a este problema. A norma define os requisitos necessários para a transmissão segura de dados de identificação entre dispositivos.

A norma técnica *ISO/IEC DIS 18013-5*, é um documento que descreve a interface e os requisitos técnicos necessários para facilitar as funcionalidades da carta de condução digital em conformidade com a *ISO* num dispositivo móvel. Este documento, especifica as interfaces entre o *mDoc* (*Mobile Document*) e o leitor *mDoc*, e a interface entre o leitor *mDoc* e a autoridade emissora. O leitor *mDoc*, consiste num dispositivo com a capacidade de enviar, receber e verificar dados relativos a *mDocs*. O leitor *mDoc* encontra-se associado a um verificador *mDoc*, que corresponde à pessoa/organização que controla o leitor *mDoc* para efeitos de verificação de um *mDoc*. Um *mDoc* está associado a um portador *mDoc*, que representa o indivíduo ao qual um *mDoc* foi emitido. A autoridade emissora é um agente ou organização que detêm autoridade para emitir uma carta de condução. Neste âmbito, o documento possibilita a outras entidades para além da autoridade emissora a:

- Usar um dispositivo para obter dados *mDL*;
- Ligar o *mDL* ao portador *mDL*;
- Autenticar a origem dos dados *mDL*;
- Verificar a integridade dos dados *mDL*.

Antes de apresentar as interfaces representadas na norma técnica *ISO/IEC DIS 18013-5*, é necessário clarificar que o termo *mDL (mobile Driving License)* se refere à carta de condução que corresponde a uma instância dos *mDocs*.

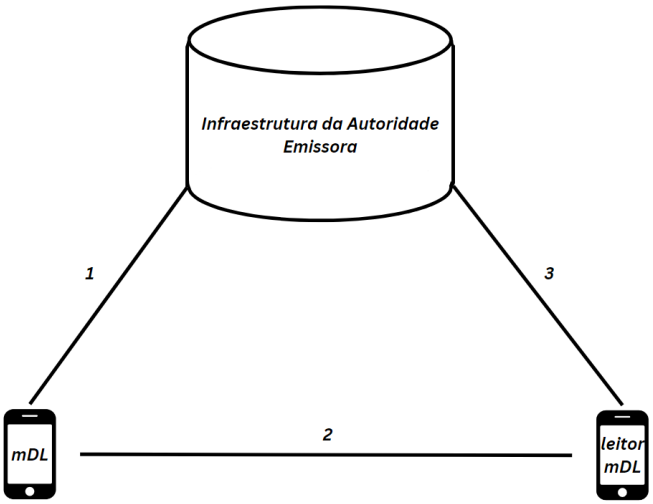


Figura 1: Interfaces *mDoc*

A Figura 1 identifica três interfaces:

- Interface 1: Interface entre o *mDoc* e a autoridade emissora;
- Interface 2: Interface entre o *mDoc* e o leitor *mDoc*;
- Interface 3: Interface entre o leitor *mDoc* e a autoridade emissora.

Tendo em conta a natureza dos dispositivos wearable, apenas é considerada a interface entre o *mDoc* e o leitor *mDoc*. O *smartwatch* têm o papel de portador *mDoc*, logo, é possível descartar a interface entre o leitor *mDoc* e a autoridade emissora. A interface entre o portador *mDoc* e a autoridade emissora ainda não se encontra no escopo deste standard, o que a torna, de momento, irrelevante para o problema em questão.

2.1.1 *Requisitos funcionais do standard ISO/IEC DIS 18013-5*

A norma técnica *ISO/IEC DIS 18013-5* apresenta e descreve os seguintes requisitos funcionais:

- Um verificador *mDoc* juntamente com um leitor *mDoc* será capaz de pedir, receber e verificar a integridade e autenticidade de um *mDoc* independentemente da conectividade com a internet;
- Um verificador *mDoc* não associado à autoridade emissora deverá ser capaz de verificar a integridade e autenticidade de um *mDoc*;
- Um verificador *mDoc* deverá ser capaz de confirmar a vinculação entre a pessoa que apresenta o *mDoc* e o portador *mDoc*;

- A interface entre o *mDoc* e o leitor *mDoc* deverá suportar a transmissão selectiva de dados *mDoc* para um leitor *mDoc*.

2.1.2 *Requisitos técnicos do standard ISO/IEC DIS 18013-5*

Os requisitos técnicos descrevem o modelo de dados *mDL*, as trocas de dados *mDL*, e os mecanismos de segurança usados nas interacções mencionadas neste documento.

- **Modelo de dados mDL:** O modelo de dados *mDoc* apresenta a organização e estrutura dos elementos de dados presentes num *mDoc*, e descreve os identificadores e formatos dos elementos de dados. Elementos como o *nome* ou *data de nascimento* são obrigatórios, enquanto que elementos como o *sexo*, *altura* ou *peso* são opcionais. O elemento *driving_privileges*, que representa as permissões do portador *mDoc* encontra-se formatado de forma diferente. Neste caso, *driving_privileges* é representado por uma estrutura correspondente a um *array*, onde os elementos são permissões que portador *mdoc* tem enquanto detentor do *mDoc*.
- **Trocas de dados mDL:** As trocas de dados *mDL* consistem em três fases: fase de inicialização, fase de conexão entre dispositivos e fase de recolha de dados.
  - Fase de inicialização: Durante a fase de inicialização o *mDoc* é activado. A activação do *mDoc* é feita pelo portador *mDoc*.
  - Fase de conexão entre dispositivos: Durante esta fase, a informação necessária para preparar e assegurar uma troca de dados segura é trocada entre o *mDL* e o leitor *mDL*. As tecnologias de transmissão disponíveis para a troca de informação são *NFC (Near Field Communication)* e *QR code*.
  - Fase de transmissão de dados: Durante esta fase, ocorre a transmissão de dados *mDL* entre os dispositivos. Esta fase é posterior à fase de conexão de dispositivos. Há dois tipos de transmissão de dados:
    - Transmissão de dados entre dispositivos: Método de transmissão de dados usando exclusivamente a interface entre o leitor *mDoc* e o portador *mDoc*. O leitor *mDoc* e o portador *mDoc* comunicam usando pedidos *mDoc* e respostas *mDoc* codificadas em *CBOR (Concise Binary Object Representation)*. Há três tecnologias que podem ser utilizadas com este método: *BLE (Bluetooth Low Energy)*, *NFC (Near Field Communication)* e *Wi-Fi Aware*. Neste tipo de transmissão de dados os dispositivos não precisam de estar conectados à *internet* para haver troca de dados;
    - Transmissão de dados entre dispositivo e autoridade emissora: Método de recolha de dados usando a interface entre o leitor *mDL* e a autoridade emissora. Neste tipo de recuperação de dados é necessário o leitor *mDL* estar conectado à *internet*. Para que haja transmissão de

dados entre o leitor *mDoc* e a autoridade emissora é necessário um *token*. Este *token* identifica o portador *mDoc* e o leitor *mDoc* para a autoridade emissora. O leitor *mDoc* obtem este *token* do portador *mDoc* durante a fase de conexão entre dispositivos, ou então, iniciando primeiro uma transmissão de dados entre dispositivos, onde o portador *mDoc* envia o *token* ao leitor *mDoc*. Para este tipo de transmissão há duas tecnologias: *WebAPI* e *OIDC (OpenID Connect)*.

- **Mecanismos de segurança:** Os mecanismos de segurança têm como objetivo preservar a tríade de confidencialidade, autenticidade e integridade. A arquitetura de segurança apresentada no documento têm quatro objetivos distintos:
  - Proteção contra falsificações;
  - Proteção contra clonagem;
  - Proteção contra intercetações/espionagem;
  - Proteção contra acesso não autorizado.

2.1.3 *Tecnologias de Conexão entre dispositivos*

A fase de conexão entre dispositivos apresenta duas tecnologias para ocorrerem trocas de informação:

- *Near Field Communication*: O uso de *NFC* segue o protocolo *Connection Handover* definido no Forum NFC *Connection Handover (CH) Technical Specification, Version 1.5*. Para a ocorrência de *handover* há duas opções: *Static Handover* e *Negotiated Handover*. No primeiro caso, são apresentadas as tecnologias de transmissão de dados suportadas pelo *mDoc*, das quais o leitor *mDoc* seleciona uma. No segundo caso, são apresentadas as tecnologias de transmissão suportadas pelo leitor *mDoc*, das quais o *mDoc* seleciona uma.
- *QR code*: Se for usado um *QR code*, a estrutura de conexão entre dispositivos, que contêm informação importante para a conexão entre os dispositivos, é transmitida na forma de um código de acordo com a norma *ISO/IEC 18004*. Neste caso o leitor *mDoc* seleciona uma das tecnologias de transmissão de dados presentes na estrutura de conexão entre dispositivos.

A informação partilhada entre o portador *mDoc* e o leitor *mDoc* corresponde à estrutura de conexão entre dispositivos (ver Secção 4.3.2). Esta estrutura contém os dados necessários para o estabelecimento de um canal de comunicação seguro entre o portador *mDoc* e o leitor *mDoc*.

2.1.4 *Tecnologias de transmissão de dados*

Para recuperação de dados entre dispositivos, a fase de transmissão de dados apresenta três tecnologias de transmissão: Bluetooth Low Energy (BLE), NFC e Wi-fi Aware:

- *Bluetooth Low Energy*: O uso de *BLE (Bluetooth Low Energy)* implica uma implementação de acordo com o documento *Bluetooth Core Specification* (Woolley,1999). A estrutura de conexão entre dispositivos indica qual a função do portador *mDoc* na dinâmica da conexão *BLE*. O portador pode ter a função de *Central*, *Peripheral*, ou as duas. A função de um *Central* é iniciar um pedido de ligação para um *Peripheral*, cuja função é partilhar o *PSM (Protocol Service Multiplexor)* pelo processo de *advertisement*. O *PSM (Protocol Service Multiplexor)* corresponde ao valor que é usado para estabelecer a conexão *BLE* entre o portador *mDoc* e o leitor *mDoc*. O *Peripheral* pode aceitar pedidos de ligação depois do processo de *advertisement*. Esta tecnologia de transmissão tem duas fases: estabelecimento de conexão e transmissão de dados. Na fase de estabelecimento de conexão, o portador *mdoc* e o leitor *mDL* utilizam a informação da fase de conexão entre dispositivos para se conectarem um ao outro, criando uma *Bluetooth socket* entre eles. Depois de conectados, é iniciada a fase de transmissão de dados, onde os dados são enviados pela *socket*. A sessão é terminada quando o portador *mDoc* ou o leitor *mDoc* decidirem encerrar a conexão.
- *Near Field Communication*: *NFC (Near Field Communication)* também pode ser usado para a transmissão de dados entre o portador *mDoc* e o leitor *mDoc*. O portador *mDoc* suporta o modo *PICC (Proximity chip card)* e o leitor *mDoc* o modo *PCD (Proximity coupling device)*. Estes modos estão definidos de acordo com a norma técnica *ISO/IEC 14443*, e permitem a troca de dados entre o portador *mDoc* e o leitor *mDoc*.
- *Wi-Fi Aware*: *Wi-Fi Aware* é implementada de acordo com a *Wi-Fi Alliance Neighbor Awareness Networking Specification*. A transmissão de dados com esta tecnologia consiste em três fases: Estabelecimento de conexão, transmissão de dados, encerramento. Na fase de estabelecimento de conexão, o portador *mdoc* e o leitor *mDL* utilizam a informação da fase de conexão entre dispositivos para se conectarem um ao outro. Durante a fase de transmissão de dados, os dados são transferidos usando o protocolo *HTTP (Hypertext Transfer Protocol)*, onde o portador *mDoc* tem a função de servidor *HTTP* e *TCP (Transmission Control Protocol)*, e o leitor *mDoc* tem a função de cliente *HTTP* e *TCP*. Os dados *mDoc* são transmitidos usando o método *HTTP POST*. Por fim, a conexão é encerrada quando o portador *mDoc* ou o leitor *mDoc* receberem um código de encerramento de sessão (ver *Secção 9.1.1.4* da norma *ISO/IEC 18013-5*).

2.2 PLATAFORMAS DE DESENVOLVIMENTO CRUZADO

As plataformas de desenvolvimento cruzado são *frameworks* que permitem desenvolver aplicações para várias plataformas (*i.e., Android, iOS, Windows, etc.*), onde não é necessária a escrita de código separadamente para cada plataforma. O código desenvolvido é reutilizado entre as várias plataformas, sendo apenas necessário escrevê-lo uma vez. Contudo, devido à especificidade de algumas plataformas, nem todo o código pode ser partilhado. As plataformas *wearable* são específicas no seu *UI (User Interface)* o que dificulta a partilha de código neste contexto. Apesar disto, é possível a partilha da lógica de negócio entre as várias plataformas, o que é uma grande vantagem no contexto desta aplicação.

O objetivo principal deste trabalho é a implementação da norma técnica *ISO/IEC 18013-5* em plataformas *wearable*, o que consiste na implementação das estruturas de dados e mecanismos necessários para o bom funcionamento da aplicação. Neste contexto, a vantagem de usar plataformas de desenvolvimento cruzado é a possível partilha destas estruturas de dados e mecanismos entre as várias plataformas *wearable*.

2.2.1 *Suporte wearable em plataformas de desenvolvimento cruzado*

Como foi referido no Capítulo 1, os dispositivos *wearable*, mais especificamente os *smartwatches*, encontram-se num processo recente de crescimento. Isto faz com que o suporte para o desenvolvimento de aplicações em plataformas *wearable* seja pouco extenso. Assim, é necessário escolher uma plataforma cujo suporte *wearable* e a respetiva documentação sejam abrangentes o suficiente para a implementação da norma *ISO/IEC DIS 18013-5*.

Tendo em conta que o suporte *wearable* é critério mais importante na escolha da plataforma de desenvolvimento cruzado a usar, consideram-se as seguintes ferramentas: *React Native*, *Flutter* e *Xamarin*.

React Native

React Native é uma *framework* com base no *React* (Abramov and Nabors,2013), que usa a biblioteca *JavaScript*. *React Native* é usado para desenvolver aplicações móveis, e assim como as aplicações desenvolvidas em *React*, as aplicações *React Native* são escritas usando uma mistura de *JavaScript* e *XML* conhecida por *JSX (JavaScript Syntax Extension)*. O código desenvolvido com *React Native* é escrito em *JavaScript*, e convertido para a linguagem nativa do sistema operacional (*i.e., Objective-C* para *iOS* e *JAVA* para *Android*). Esta característica permite que o código seja escrito apenas uma vez em *JavaScript*, e seja reutilizado em plataformas diferentes. Para além da reusabilidade do código desenvolvido, *React Native* permite ver as mudanças feitas no código em tempo real no dispositivo/emulador, e apresenta bibliotecas *third-party* em *JavaScript*, que tornam o processo de desenvolvimento da aplicação mais fácil e rápido.

Esta *framework* não apresenta suporte *wearable*, o que torna o desenvolvimento de aplicações *wearable* impossível. Porém, existem *plugins* desenvolvidos por utilizadores desta ferramenta, que permitem a criação de aplicações *wearable* com *React Native*.

Flutter

Flutter (Google,2017) é uma plataforma de desenvolvimento cruzado criada pela *Google*, que permite o desenvolvimento de aplicações *mobile* a partir de uma única base de código. Esta *framework* utiliza *Dart*, uma linguagem com sintaxe comparável a *JavaScript*. Uma das principais componentes desta *framework* são os *widgets*. Um *widget* corresponde a um elemento de uma *UI (User Interface)*. *Flutter* apresenta uma seleção de *widgets* personalizáveis para a construção de *UI's* funcionais. Ao contrário das outras *frameworks*, *Flutter* apresenta o *Flutter SDK (Software Development Kit)* que compila o código escrito em *Dart* para código máquina nativo. Esta característica e os *widgets* preexistente, fazem com que as aplicações em *Flutter* tenham um desempenho perto do desempenho nativo. O código *Dart* e os *widgets* podem ser reutilizados em várias plataformas, permitindo assim a partilha de código em aplicações *Flutter*.

Flutter tem suporte *wearable* apenas para a plataforma *WearOS*. Apesar de não suportar a plataforma *watchOS*, utilizadores desta *framework* desenvolveram *plugins* que permitem a criação de aplicações *watchOS*. Contudo, estes *plugins* desenvolvidos pelos utilizadores da *framework* encontram-se desatualizados.

Xamarin

Xamarin (Microsoft,2012) é uma plataforma *mobile* de desenvolvimento cruzado criada pela *Microsoft*, que se encontra integrada em *.NET* (Microsoft,2002). *.NET* é uma *framework* de desenvolvimento cruzado que permite desenvolver aplicações para várias plataformas. A ferramenta *.NET* permite a criação de aplicações *web*, *mobile*, *desktop*, etc., em *C#*, *F#* e *Visual Basic*. *Xamarin* corresponde a uma plataforma *.NET* com ferramentas e bibliotecas específicas para o desenvolvimento de aplicações *mobile* em *C#*. Esta plataforma tem a seu dispor o *.NET Standard*, que corresponde ao conjunto de *API's .NET* que estão disponíveis entre as várias implementações de *.NET*. Assim, as aplicações *Xamarin* podem partilhar o código desenvolvido com bibliotecas *.NET* entre as várias plataformas *mobile*. Apesar das vantagens referidas, a partilha de código é limitada no sentido em que não é possível partilhar o *UI*. Tendo isto em conta, *Xamarin* apresenta uma *framework* de desenvolvimento *UI*, o *Xamarin.Forms*. Esta *framework* permite o desenvolvimento e partilha de *UI's* a partir de uma única base de código.

Xamarin tem um bom suporte *wearable* integrado, com bibliotecas *wearable* que permitem a criação de aplicações *WearOS* e *watch OS* a partir de uma única base de código. No entanto, *Xamarin.Forms* não tem suporte *wearable*, logo não há partilha do código *UI* sendo necessário escrever código específico para cada plataforma *wearable*.

Tendo em conta a natureza deste trabalho, é necessária a plataforma com o maior suporte *wearable* possível. *React Native* não tem suporte *wearable* e *Flutter* tem suporte, mas apenas para a plataforma *WearOS*. Das três plataformas consideradas, *Xamarin* é a única que apresenta suporte *wearable* para as plataformas *WearOS* e *watch OS*. Esta característica é crucial, pois sem ela não é possível a implementação da norma técnica *ISO/IEC DIS 18013-5*. Conclui-se que o *Xamarin* é a ferramenta mais indicada para a implementação da norma técnica *ISO/IEC DIS 18013-5* em plataformas *wearable*. O simples facto de este apresentar um suporte *wearable* muito superior às outras ferramentas, torna *Xamarin* a melhor escolha para a implementação desta norma em plataformas *wearable*.

2.2.2 *Partilha de código em Xamarin*

Como foi referido na Secção 2.2.1, *Xamarin* apresenta suporte *wearable* para as plataformas *WearOS* e *watch OS*, o que significa que é possível criar uma aplicação onde o código de lógica de negócio seja partilhado entre as plataformas *wearable* consideradas (*i.e., WearOS*, *watch OS*). Contudo, e como referido na Secção 2.2.1, a *framework Xamarin.Forms* não é aplicável na dinâmica *wearable*, visto não suportar o desenvolvimento *UI* para plataformas *wearable*.

Xamarin utiliza como *IDE (Integrated Development Environment)* a *framework Visual Studio*. Aqui é criado o projeto *Net_Shared* que corresponde ao projeto *.NET Standard*. Este projeto contém as bibliotecas pertencentes à versão *.Net Standard* considerada e que podem ser partilhadas entre as plataformas *.NET* (incluindo *Xamarin.Android e Xamarin.iOS*). É também neste projeto que é incluído o código desenvolvido que é partilhado entre as várias plataformas *wearable* consideradas. Tendo isto em conta, a solução *wearable* vai ter dois projetos para que seja possível a partilha de código, o projeto *.Net Standard*, que vai conter o código a ser partilhado, e o projeto *wearable* específico, que vai conter todo o código específica à plataforma.

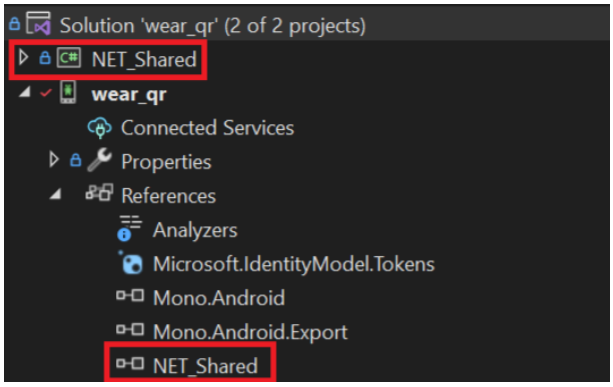


Figura 2: Projeto *wearable*

A solução apresentada na Figura 2 demonstra como a partilha de código é atingida com dois projetos. O projeto *NET_Shared* (a vermelho na Figura 2) contém o código a ser partilhado entre as várias plataformas *wearable*, e o projeto *wear_qr* contém o código específico da plataforma *wearable*. *Net_Shared* é referenciado nos

projetos específicos *wearable*, de modo a que o código desenvolvido possa ser utilizado nas várias plataformas *wearable*.

2.3 RESUMO DO CAPÍTULO

No Capítulo 2 aborda-se a relevância dos dispositivos *wearable*, mais especificamente dos *smartwatches*, a norma técnica *ISO/IEC DIS 18013-5*, e as tecnologias consideradas para a implementação desta norma nestes dispositivos. São referidos exemplos da característica prática dos *smartwatches* e que estes dispositivos tem potencial de implementação da norma técnica *ISO/IEC DIS 18013-5*. O Capítulo descreve o documento que representa esta norma, identificando os participantes das interações nele descritas, apresentando os requisitos funcionais e técnicos, e as tecnologias utilizadas para a conexão entre dispositivos e transmissão de dados. A seguir, são apresentadas as plataformas de desenvolvimento cruzado consideradas para a implementação da norma técnica *ISO/IEC DIS 18013-5* nas plataformas *wearable*. É feita uma introdução a cada ferramenta e uma análise ao seu suporte para plataformas *wearable*. Por fim, é explicado como a plataforma de desenvolvimento cruzado escolhida é utilizada para o desenvolvimento de projetos *wearable*, cujo código pode ser partilhado. No Capítulo 3 é apresentada a solução proposta para a implementação da norma técnica *ISO/IEC DIS 18013-5* nas plataformas *wearable*. Neste capítulo é explicado o funcionamento da aplicação e as decisões tomadas tendo em conta a dinâmica peculiar imposta pela utilização de *smartwatches* para a implementação desta norma.

3

SOLUÇÃO PROPOSTA

A dinâmica entre a norma técnica *ISO/IEC DIS 18013-5* e as plataformas *wearable*, cria uma necessidade de seleção dos elementos relevantes da norma para esta implementação. Para criar uma solução, é necessária a definição de objetivos específicos tendo em conta estes elementos. Estes objetivos implicam uma tomada de decisão em relação aos requisitos da norma técnica *ISO/IEC DIS 18013-5*, da plataforma de desenvolvimento, e das tecnologias a utilizar na construção desta solução.

A identificação dos objetivos requer como base a definição do papel do *smartwatch* no contexto da norma técnica *ISO/IEC DIS 18013-5*. A solução presente neste trabalho considera o *smartwatch* como um portador *mDoc*. Assim, este trabalho implementa as estruturas de dados, tecnologias e mecanismos de segurança que são da responsabilidade do portador *mDoc*, e que permitem o bom funcionamento deste nas interações consideradas.

3.1 OBJETIVOS

Este trabalho tem como objetivo principal a implementação da norma técnica *ISO/IEC 18013-5* (Iso,2021) num *smartwatch*, utilizando uma plataforma de desenvolvimento cruzado. A definição deste objetivo levanta subobjetivos que devem ser realizados:

- Implementação das estruturas *mDoc* (ver Capítulo 4);
- Implementação dos mecanismos de segurança (ver Capítulo 4);
- Uso de uma tecnologia de conexão de dispositivos (ver Secção 4.3.2);
- Uso de uma tecnologia de transmissão de dados (ver Secção 4.3.4);
- Cumprimento dos requisitos funcionais da norma técnica *ISO/IEC 18013-5*;
- Cumprimento dos requisitos técnicos da norma técnica *ISO/IEC 18013-5*.

É importante relembrar que este trabalho considera o *smartwatch* como um portador *mDoc*. Assim, apenas é necessário ter em consideração os requisitos funcionais (ver Secção 2.1.1) que impactam o portador *mDoc*.

3.2 IMPLEMENTAÇÃO E ADAPTAÇÃO DA NORMA TÉCNICA ISO/IEC DIS 18013-5

No desenvolvimento deste trabalho é necessário ter em especial atenção a forma como a norma técnica *ISO/IEC 18013-5* se adapta quando o portador *mDL* é um *smartwatch*:

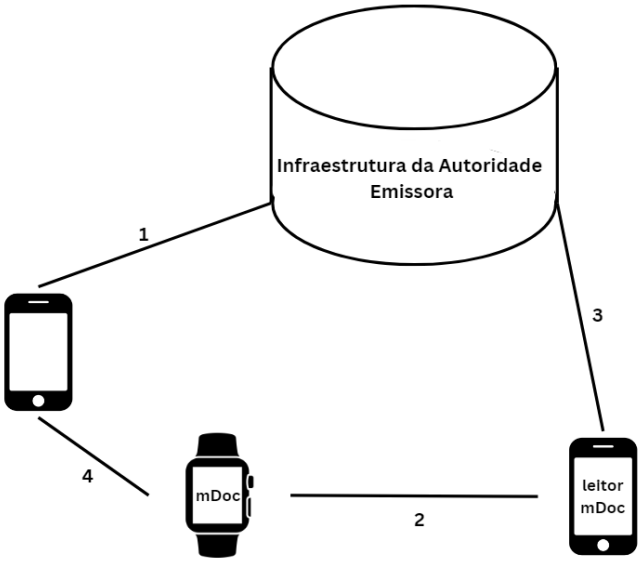


Figura 3: *Interfaces mDoc* com *Smartwatch*

A Figura 3 representa as *interfaces* da Figura 1 adaptadas a inclusão de um *smartwatch*. Olhando para a Figura 3 é possível notar duas diferenças em relação à Figura 1.

A primeira diferença é que há uma *interface* adicional que liga o *smartwatch* ao *smartphone*, pois apesar do *smartwatch* ser o portador *mDoc*, é necessário emparelha-lo com um *smartphone*. A obtenção do *mDoc* é feita pelo *smartphone* enquanto portador *mDoc*, ao qual o *smartwatch* tem acesso por emparelhamento tornando-o um portador *mDoc*. Este processo corresponde à *interface* 1, que como já foi referido anteriormente na Secção 2.1, tanto ela como a *interface* 3 não fazem parte do escopo deste trabalho. Assim, esta solução assume a presença do documento *mDoc* e da *MSO (Mobile Security Object)* no portador *mDoc*. Para evitar confusão entre o *smartphone* e o *smartwatch* enquanto portadores *mDoc*, o termo "portador *mDoc*" refere-se ao *smartwatch* enquanto portador *mDoc* ao longo do trabalho.

A segunda diferença é a *interface* 2, que na Figura 3 liga o *smartwatch* ao leitor *mDoc*. O *smartwatch* é o portador *mDoc* considerado para o desenvolvimento desta implementação, sendo ele quem comunica com o leitor *mDoc*.

Apesar da necessidade de emparelhar o *smartwatch* com um *smartphone*, a aplicação desenvolvida é *standalone*. Isto significa que o próprio *smartwatch* pode armazenar os dados *mDoc* e realizar os processos crip-

tográficos que garantem o bom funcionamento da aplicação, sendo completamente independente do *smartphone* durante o funcionamento da aplicação.

3.2.1 *Decisões e módulos da aplicação*

Tecnologias de conexão e comunicação

A utilização do *smartwatch* enquanto portador *mDoc* leva à consideração de quais as tecnologias de conexão e comunicação, referidas na norma técnica *ISO/IEC 18013-5*, devem ser escolhidas. São apresentadas duas tecnologias de transmissão de conexão de dispositivos: *QR Code* e *NFC*; e três tecnologias de transmissão entre dispositivos: *BLE (Bluetooth Low Energy)*, *NFC (Near Field Communication)* e *Wi-fi Aware*.

O portador *mDoc* implementado utiliza *QR Code* como a sua tecnologia de conexão de dados. *NFC* tem um alcance menor que o *QR Code*, e nem todos os *smartwatches* tem uma *NFC tag*, que é um *chip* necessário para utilizar *NFC*. *QR Code* é uma tecnologia universal, no sentido em que não é preciso nenhum *scanner* especial para ler um *QR Code* e podem ser gerados facilmente.

Quanto à transmissão de dados entre o portador *mDoc* e o leitor *mDoc*, é utilizada a tecnologia *Bluetooth Low Energy (BLE)*. *BLE* é ideal para transmitir dados de pequeno tamanho, e apesar do alcance da tecnologia *Wi-fi Aware* ser maior, *BLE* tem um poder de consumo menor que *Wi-fi Aware*, que é uma característica ideal para *smartwatches*. *NFC* foi excluído pelas mesmas razões que não foi utilizado como tecnologia de conexão entre dispositivos.

Módulos do portador mDoc

As implementações referidas na Secção 3.2 tem em consideração o objetivo principal deste trabalho: a implementação de um suporte *wearable* para a norma técnica *ISO/IEC 18013-5*. Como se encontra referido na Secção 2.2.1, *Xamarin* permite a partilha do código desenvolvido, mas não apresenta ferramentas para a partilha do código *UI*. Tendo isto em conta, a aplicação é desenvolvida na plataforma *wearable WearOS*, onde todo o código é partilhado, exceto o código *UI* e o código referente a tecnologia de transmissão *BLE (Bluetooth Low Energy)*.

A utilização da tecnologia de transmissão *BLE* é feita através de bibliotecas *BLE* escritas em *C#*. Estas bibliotecas tem dependências específicas a plataforma, o que não torna possível a partilha do seu código.

A Figura 7 demonstra as implementações como módulos do portador *mDoc*:

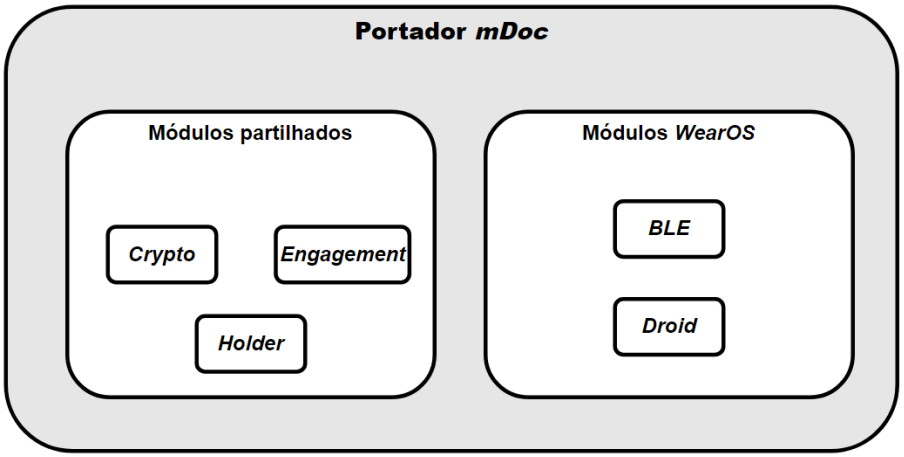


Figura 4: Módulos do portador *mDoc*

Os módulos da aplicação do portador *mDoc* podem ser divididos em módulos partilhados, e módulos específicos à plataforma *wearable WearOS*. A divisão dentro de cada grupo de módulos é feita tendo em conta o papel das suas implementações na norma técnica *ISO/IEC DIS 18013-5*. Os módulos partilhados utilizam bibliotecas que não dependem da plataforma *wearable*, e constituem a maior parte do código da aplicação. Os módulos específicos a *WearOS* utilizam bibliotecas *Android* para a implementação dos módulos.

Como foi referido anteriormente, com a exceção do *UI* e do *BLE*, todas as estruturas de dados e mecanismos de segurança fazem parte do grupo de módulos partilhados. No módulo *Engagement* implementa-se a estrutura de conexão entre dispositivos (ver Secção 4.3.2) e gera-se o *QR Code*. Em *Crypto*, estão implementadas as estruturas *COSE* (ver Secção 4.3.1) e os algoritmos criptográficos necessários para os mecanismos de segurança. *Holder* é o módulo onde está implementado o armazenamento das estruturas de dados e criptográficas necessárias para a comunicação portador-leitor, e é também onde se inicializam os mecanismos criptográficos do portador *mDoc*.

O módulo específico *BLE* utiliza bibliotecas *Bluetooth Android* para implementar o *BLE Peripheral* (ver Secção 4.3.4), que corresponde ao papel do portador *mDoc* na dinâmica *BLE*, e o *BLE Advertiser*, que é a implementação do processo de *advertising* e dos pacotes *BLE*. No módulo *Droid* está implementada a comunicação entre o módulo *BLE* e o módulo *Holder*. Este módulo, surge da necessidade das estruturas implementadas no módulo *Holder* precisarem de estruturas e variáveis do módulo *BLE* para a comunicação portador-leitor. Por último, o módulo *UI* implementa a interface de utilizador que permite que o detentor do portador *mDoc* interaja com a aplicação.

3.2.2 *Conexão e comunicação entre o portador mDoc e o leitor mDoc*

De acordo com a norma técnica *ISO/IEC 18013-5*, o portador *mDoc* suporta no mínimo uma tecnologia de conexão entre dispositivos, e uma tecnologia de transmissão de dados. Tendo isto em conta, e com o intuito de simplificar o processo de prova de conceito, não são consideradas todas as tecnologias de conexão e transmissão de dados. Nesta aplicação, é utilizada a tecnologia *QR Code* (ver Secção 4.3.2) para a conexão de dados, e a tecnologia *Bluetooth Low Energy* (ver Secção 4.3.4) para a transmissão de dados entre dispositivos.

A tecnologia *QR code* é utilizada para transmitir a informação criptográfica necessária para a criação da conexão entre o portador *mDoc* e o leitor *mDoc*. O portador *mDoc* está encarregue de gerar uma estrutura (ver Secção 4.3.2) que contém informação relativamente às tecnologias de transmissão que utiliza (*i.e., BLE*), e elementos criptográficos (*i.e.,* chave efémera pública do portador *mDoc*) necessários para a criação de um canal de comunicação seguro com o leitor *mDoc*. A informação relativa à tecnologia de transmissão de dados indica que o portador *mDoc* utiliza *BLE*, e qual o papel do portador *mDoc* neste contexto. A informação criptográfica é utilizada pelo leitor *mDoc* para gerar as chaves de sessão (ver Secção 4.3.1), que são estruturas criptográficas utilizadas para cifrar e decifrar os pedidos e respostas *mDoc* (ver Secções 4.1), respetivamente.

Depois do leitor *mDoc* criar as chaves de sessão, e verificar que o portador utiliza *BLE* como tecnologia de transmissão de dados, é iniciado o processo de criação do canal *BLE* para a transmissão de dados entre o portador *mDoc* e o leitor *mDoc*. O portador *mDoc* começa por criar um *server socket* e iniciar o processo de *advertising*, que consiste na transmissão de pacotes *BLE* com informação necessária para a criação do canal *BLE*. Esta informação corresponde ao *PSM (Protocol Service Multiplexer)*, cujo valor é o número do *port* da conexão *BLE*. O leitor *mDoc* recebe os pacotes *BLE*, e utiliza o valor do *PSM* para se conectar à *server socket* do portador *mDoc*. Com o canal *BLE* criado, o portador *mDoc* e o leitor *mDoc* podem iniciar a troca de dados *mDoc*.

Como foi referido anteriormente, são utilizadas chaves de sessão para cifrar e decifrar pedidos e respostas *mDoc*. Assim como o leitor *mDoc*, é necessário que o portador tenha a informação necessária para gerar as chaves de sessão, de modo a poder decifrar os pedidos *mDoc* recebidos do leitor *mDoc*, e cifrar as respostas *mDoc* a estes pedidos. Posto isto, o leitor *mDoc* envia a primeira mensagem para o portador *mDoc*, que contém um pedido *mDoc* cifrado e a chave efémera pública do leitor *mDoc*. Com a chave efémera pública do leitor *mDoc*, o portador *mDoc* gera as chaves de sessão e decifra o pedido *mDoc*. O portador *mDoc* analisa o pedido *mDoc* e verifica quais os dados *mDoc* que o leitor *mDoc* requer. Por fim, o portador *mDoc* gera uma resposta *mDoc* de acordo com o pedidos *mDoc* recebido.

É importante ter em consideração que a aplicação desenvolvida é *standalone*, e não *companion*. Todos os processos, desde a geração e armazenamento de estruturas de dados e realização de processos criptográficos, são feitos pelo *smartwatch*, enquanto portador *mDoc*, sem qualquer influência do *smartphone* ao qual se encontra associado.

3.2.3 *Implementação das estruturas de dados e mecanismos de segurança*

As estruturas de dados e os mecanismos de segurança presentes neste trabalho encontram-se implementadas de acordo com o processo descrito na Secção 3.2.2, e de acordo com a norma técnica *ISO/IEC DIS 18013-5*. As estruturas de dados e mecanismos de segurança estão implementadas de acordo com o papel de portador *mDoc* do *smartwatch*, e de acordo com a tecnologia transmissão de dados (*i.e., BLE*).

Esta implementação tem três estruturas de dados principais, em relação às quais se desenvolvem outras estruturas de dados e mecanismos de segurança:

- Estrutura de conexão de dispositivos (ver Secção 4.3.2);
- Pedido *mDoc* (ver Secção 4.1);
- Resposta *mDoc* (ver Secção 4.1).

O conteúdo da estrutura de conexão de dispositivos corresponde a informação necessária para que seja possível a criação de um canal de comunicação entre o portador *mDoc* e o leitor *mDoc*. Isto inclui a chave efémera pública do portador *mDoc* utilizada para criar as chaves de sessão (ver Secção 4.3.1), e dados relacionados com a tecnologia de transmissão de dados *BLE*.

O pedido *mDoc* é a estrutura enviada pelo leitor *mDoc* que contém os dados *mDoc* requisitados. Para além destes dados *mDoc*, o pedido *mDoc* tem informação criptográfica sobre a autenticação do leitor *mDoc* (ver Secção 4.3.1).

Ao receber o pedido *mDoc*, o portador *mDoc* envia uma resposta *mDoc* ao leitor *mDoc*. Esta estrutura apresenta os dados *mDoc* solicitados no pedido *mDoc*, a substrutura *MSO* (ver Secção 4.3.1) utilizada para a autenticação destes dados, e informação criptográfica sobre a autenticação do portador *mDoc*.

Estas estruturas de dados correspondem a objetos que contêm pares chave-valor, que de acordo com a norma técnica *ISO/IEC DIS 18013-5*, são codificadas em *CBOR (Concise Binary Object Representation)* (ver Secção 4.2) antes de serem transmitidas. Esta codificação consiste numa serialização destas estruturas para uma *byte string CBOR*, de modo a que possam ser transmitidas entre o portador *mDoc* e o leitor *mDoc*.

De modo a garantir uma transmissão segura destas estruturas de dados, é necessário verificar a sua confidencialidade, integridade e autenticidade. A garantia desta tríade por parte do portador *mDL* é feita recorrendo aos seguintes mecanismos criptográficos (ver Secção 4.3.1):

- Cifragem da sessão;
- Autenticação do leitor *mDoc*;
- Autenticação do portador *mDoc*;

A cifragem da sessão corresponde a criação de chaves de sessão, cujo intuito é a cifragem e decifragem das mensagens *mDoc* (*i.e.,* pedido *mDoc* e resposta *mDoc*). Duas chaves de sessão são geradas, tanto pelo

portador *mDoc* como pelo leitor *mDoc*: Chave de sessão do leitor *mDoc* e a chave de sessão do portador *mDoc*. A primeira é utilizada pelo leitor *mDoc* para cifrar os pedidos *mDoc*, e utilizada pelo portador *mDoc* para decifrar estes pedidos *mDoc*. A chave de sessão do portador *mDoc* é utilizada pelo leitor *mDoc* para cifrar as respostas *mDoc*, e utilizada pelo leitor *mDoc* para decifrar estas respostas *mDoc*. Este mecanismo atribui autenticação dos dados *mDoc* a serem transmitidos, protegendo-os de ataques de intercetação/espionagem e alterações de dados.

O portador *mDoc* confirma a autenticidade do leitor *mDoc* através da chave pública de autenticação deste último, incluída nos pedidos *mDoc* enviados. O leitor *mDoc* utiliza uma chave privada de autenticação para autenticar o pedido *mDoc*. Dentro deste pedido *mDoc*, encontra-se um certificado que contém a chave publica de autenticação do leitor *mDoc*, que é utilizada pelo portador *mDoc* para confirmar a autenticidade do leitor *mDoc*.

Assim como o portador *mDoc* confirma a autenticidade do leitor *mDoc*, também o leitor *mDoc* confirma a autenticidade dos dados enviados pelo portador. As respostas *mDoc* são autenticadas pelo portador *mDoc* utilizando uma chave de autenticação privada. A chave pública de autenticação do portador *mDoc* é armazenada num certificado, que se encontra na estrutura *MSO* (Secção 4.3.1). Esta estrutura é enviada como parte da resposta *mDoc* para o leitor *mDoc*, que vai utilizar a chave pública de autenticação do portador *mDoc* para confirmar a autenticidade do portador *mDoc*.

A estruturas criptográficas implementadas nestes mecanismos de segurança utilizam *COSE (CBOR Object Signing and Encryption)* (Schaad,2017) na sua formatação. *COSE* descreve como criar e processar assinaturas, códigos de autenticação de mensagens e cifragens utilizando *CBOR*. A Secção 4.2.1 descreve de uma forma detalhada como os serviços básicos de segurança *COSE* funcionam.

3.3 RESUMO DO CAPÍTULO

No Capítulo 3 é abordada a implementação e adaptação da norma técnica, e as decisões consideradas para o desenvolvimento do portador *mDoc*. São também descritos os subobjetivos que devem ser realizados para esta implementação. São descritas as alterações da norma em relação à utilização do *smartwatch*, e que consequência é que tem nas *interfaces mDoc*. Aqui também é discutida a relação entre o *smartwatch* e o *smartphone* ao qual se encontra associado, na dinâmica da norma. É descrito o processo de conexão e comunicação portador-leitor. São apresentados os métodos de conexão e comunicação utilizados entre o portador *mDoc* e o leitor *mDoc*, e a utilização destes métodos para a transmissão de dados. Depois, são apresentadas as principais estruturas de dados e mecanismos de segurança considerados para esta implementação. É descrito o conteúdo das estruturas de dados e a sua formatação. Os mecanismos de segurança também são aqui descritos, e é explicado o seu propósito. A seguir, são apresentados módulos da aplicação, sua organização, e o que cada um implementa. São também esclarecidas as decisões tomadas em relação à implementação do portador *mDoc*. No Capítulo 4 são descritas as implementações de cada módulo. Esta descrição passa pela demonstração das classes criadas para representar as estruturas de dados, os seus elementos e o seu propósito, e as estruturas e algoritmos criptográficos utilizados na construção dos mecanismos de segurança.

4

MÓDULOS E IMPLEMENTAÇÕES

Neste capítulo são apresentadas as implementações de cada módulo introduzido no Capítulo 3. A relação que os módulos da aplicação tem entre si, é exposta de modo a ficar claro o funcionamento do portador *mDoc*. Para além disto, também é explicado o propósito de cada implementação intra-modular. Esta explicação passa pela definição e descrição das classes, métodos e variáveis criadas para desenvolver estas implementações. Porém, antes de apresentar os módulos e os seus constituintes, é necessária uma familiarização com os dados *mDoc*, e uma introdução aos métodos utilizados para a formatação das estruturas de dados e criptográficas criadas (*i.e., CBOR (Concise Binary Object Representation)* e *COSE (CBOR Object Signing and Encryption)*).

4.1 ESTRUTURAS E DADOS *mdoc*

Os elementos de dados *mDoc* correspondem não só à informação relativa à carta de condução de um portador *mDoc*. Estes elementos são definidos na norma técnica *ISO/IEC DIS 18013-5* (Iso,2021) pelas seguintes características:

- ***Identificador***: Identificador do elemento de dados em formato *tstr*;
- ***Significado***: Significado do elemento de dados;
- ***Definição***: Definição do elemento de dados;
- ***Presença***: Indicador da necessidade da presença deste elemento de dados no *mDoc*. Um elemento de dados pode ser obrigatório (M) ou opcional (0)
- ***Formato de codificação***: Indicador de como o elemento de dados deverá ser codificado.

De modo a exemplificar a caracterização dos elementos de dados, são apresentados os seguintes exemplos:

Tabela 1: Exemplos de elementos de dados *mDoc*

Identificador	**Significado**	**Definição**	**Presença**	**Formato de codificação**
family_name	Nome de família	Apelido, nome próprio ou identificador principal, do titular da *mDoc*. O valor deve utilizar apenas caracteres $latin1^b$ e deve ter um comprimento máximo de 150 caracteres.	**M**	tstr
given_name	Nomes atribuídos	Nome(s) próprio(s), outro(s) nome(s), ou identificador secundário identificador secundário, do titular da *mDoc*. O valor deve utilizar apenas caracteres $latin1^b$ e deve ter um comprimento máximo de 150 caracteres.	**M**	tstr
issuing_authority	Autoridade emissora	Nome da autoridade emissora. O valor deve utilizar apenas caracteres $latin1^b$ e deve ter um comprimento máximo de 150 caracteres.	**M**	tstr
height	Altura $(cm)^a$	Altura do portador *mDoc* em centímetros	**O**	uint
portrait	Retrato do portador *mDoc*	Imagem de retrato do portador *mDoc* que deve seguir os requisitos da norma técnica *ISO/IEC 18013-2:2020 (Iso,)*.	**M**	bstr

Alguns dos elementos de dados apresentados na norma técnica *ISO/IEC DIS 18013-5* apresentam descrições e estruturas atípicas. Isto significa que a sua estrutura segue um conjunto de regras específicas. Um exemplo destes elementos de dados é o elemento *Portrait of mDL holder* apresentado na tabela **??**. Neste caso em específico é necessário seguir os requisitos de imagem especificados na norma técnica *ISO/IEC 18013-2:2020*. Outro exemplo são os privilégios de condução, que são representados pela estrutura *DrivingPrivileges*. A definição dos elementos constituintes desta estrutura encontram-se descritos na cláusula 5 da norma técnica *ISO/IEC 18013-1:2018* (Iso,2018). Os valores destes elementos estão definidos nos *standards ISO/IEC 18013-2:2020* e *ISO/IEC 18013-1:2018*. Os restantes elementos de dados atípicos encontram-se definidos na norma técnica *ISO/IEC DIS 18013-5*.

4.1.1 *Estrutura do documento mDoc*

A norma técnica *ISO/IEC DIS 18013-5* descreve que o modelo de dados *mDoc* utiliza os parâmetros *DocType* e *NameSpaces*, para encapsular o tipo do documento *mDoc* e o espaço em que os dados *mDoc* são definidos. Estes parâmetros encontram-se definidos de acordo com a *CDDL (Concise Data Definition Language)*, que corresponde a uma convenção sobre a notação utilizada em estruturas de dados *CBOR* e *JSON (JavaScript Object Notation)*. *Doctype* é utilizado para indicar o tipo de documento do *mDoc*, que de acordo com a norma técnica *ISO/IEC DIS 18013-5* tem o valor *"org.iso.18013.5.1.mDL"*, e indica que o *mDoc* é do tipo *mDL (mobile Driving License)*. O parâmetro *NameSpaces* encapsula os elementos de dados *mDoc* referidos na Secção 4.1 e tem o valor *"org.iso.18013.5.1"*. Para ter em conta dados nacionais, uma autoridade emissora pode definir o seu *NameSpace* (*e.g., "org.iso.18013.5.1.PT"* para dados portugueses).

```
"mdoc": {
    "docType": "org.iso.18013.5.1.mDL",
    "nameSpaces": {
        "org.iso.18013.5.1": {
            "family_name": "pGhkaWdlc3RJZBjfZnJhbmRvbXgYVi0yOVZGUnNT... ,
            "given_name": "pGhkaWdlc3RJZBRmcmFuZG9teBhNLU1ra1hkalFlT... ,
            "birth_date": "pGhkaWdlc3RJZBhnZnJhbmRvbXgYYVIwUFg1a0VSV... ,
            .
            .
            .
        }
        "org.iso.18013.5.1.PT": {
            "inhibitions": "pGhkaWdlc3RJZBjNZnJhbmRvbXgYSW9LN01rR3RT...
        }

    }
}
```

Listing 4.1: Exemplo do documento mDoc

4.1.2 *Estrutura do pedido mDoc*

Como já foi referido anteriormente, apenas a interface entre o portador *mDoc* e leitor *mDoc* é relevante para o escopo desta aplicação. Assim, sempre que o leitor *mDoc* necessita de informação *mDoc* do portador *mDoc*, envia um pedido de recuperação de dados entre dispositivos (Iso,2021):

```
DeviceRequest = {
    "version" : tstr,
    "docRequests" : [+ DocRequest]
}
```

A estrutura que representa um pedido *mDL* é um *CBOR map* onde o primeiro par corresponde à versão da estrutura *DeviceRequest*, cujo valor, de acordo com a *ISO* (Iso,2021), é "1.0". O ultimo par *docRequests* contém um *array* de todos os documentos pedidos, onde cada elemento do *array* representa um documento.

```
DocRequest = {
    "itemsRequest" : ItemsRequestBytes,
    "readerAuth" : ReaderAuth ; mdoc reader authentication
}
```

O documento pedido é definido por um *CBOR map* onde o primeiro par representa os itens pedidos. Esta estrutura apresenta o tipo do documento, os itens pedidos pelo leitor *mdoc* e os seus respetivos *namespaces*. O segundo elemento é *ReaderAuth* (ver Secção 4.3.1) que consiste numa estrutura *COSE* do tipo *COSE_Sign1* (ver Secção 4.2.1), cujo *payload* é nulo e a assinatura e os *Headers* apresentam os dados necessários para a autenticação do leitor *mDoc*.

4.1.3 *Estrutura da resposta mDoc*

Caso o pedido *mDoc* seja autorizado (ver Secção 4.3.1), o portador *mDoc* envia uma resposta *mDoc* com os dados solicitados no pedido *mDoc* para o leitor *mDoc*. A estrutura enviada pelo portador *mDoc* é codificada em *CBOR* e tem o seguinte formato (Iso,2021):

```
DeviceResponse = {
    "version" : tstr,
    "documents" : [+Document],
    ? "documentErrors": [+DocumentError],
```

```
    "status" : uint
}
```

Esta estrutura é um *CBOR map* onde o primeiro par corresponde à versão da estrutura *"DeviceResponse"*, cujo valor, de acordo com a norma técnica *ISO/IEC DIS 18013-5*, e "1.0". O par *"documents"* contém um *array* de todos os documentos retornados e *"documentErrors"* pode conter códigos de erro para documentos que não foi possível retornar. Como for referido no Capítulo 3, o portador *mDoc* implementado apenas considera documentos do tipo *mDL (mobile Driving License)*.

```
Document = {
    "docType" : DocType,
    "issuerSigned" : IssuerSigned, ;
    "deviceSigned" : DeviceSigned,
    ? "errors" : Errors
}
```

Assim, a resposta *mDoc* vai ter apenas um documento do tipo *mDL* com os itens solicitados pelo leitor *mDoc*. Por fim, o par *"status"* indica o estado da resposta do portador *mDoc* (Iso,2021). O documento retornado na resposta *mDoc* é constituído pelas estruturas *IssuerSigned* e *DeviceSigned*. *IsserSigned* é a estrutura que contém os dados pedidos e a estrutura *IssuerAuth* que corresponde à *MSO* (ver Secção 4.3.1) utilizada para a autenticação dos dados. *DeviceSigned* contém os dados *mDoc* pedidos e a estrutura *DeviceAuth* utilizada para a autenticação do portador (ver Secção 4.3.1).

4.2 CONCISE BINARY OBJECT REPRESENTATION (CBOR)

Concise Binary Object Representation, ou *CBOR*, é um formato de serialização de dados binários, cujos objetivos incluem a possibilidade de construir código extremamente pequeno, e mensagens razoavelmente pequenas, sem a necessidade de negociação de formato. Informação relativa à estrutura e implementação do formato *CBOR* pode ser encontrada no documento *RFC 7049* (Bormann and Hoffman,2013). De acordo com este documento, um item de dados bem formado consiste em dados que seguem a estrutura sintática do *CBOR*. Um item de dados *CBOR* consiste numa *byte string* que contém um item de dados bem formado codificado. O *byte* inicial de cada item de dados codificado, contém tanto informação sobre o seu *major type*, como informação adicional. O *RFC 7049* define *major types* como o valor dos primeiros três *bits* de maior ordem do *byte* inicial, utilizados para indicar o tipo do elemento de dados que se encontra codificado em *CBOR*. Os restantes 5 *bits* são utilizados para informação adicional. O documento *RFC 7049* defina os seguintes *major types*:

- **Major Type 0**: *Unsigned Integer* no intervalo $0..2^{64}-1$ inclusive;
- **Major Typer 1**: *Integer* negativo no intervalo $0..2^{64}-1$ inclusive;
- **Major Typer 2**: *Byte string*;
- **Major Typer 3**: *Text string* codificada como *UTF-8* de acordo com o *RFC 3629*(24)
- **Major Typer 4**: *Array* de items de dados;
- **Major Typer 5**: Mapa com pares de items de dados;
- **Major Typer 6**: *Tag* usada para dar semânticas adicionais a um item de dados
- **Major Typer 7**: Números *Floating point* e valores simples, assim como o código de paragem *break*;

Considere-se o seguinte *map* e a sua representação em *CBOR byte string*:

***Map*:** *{_ "a": 1, "b": [_ 2, 3]}* ***CBOR byte string*:** *0xbf6161016162 9f0203ffff*

Com a ajuda das tabelas de exemplos de dados codificados em CBOR, e *byte* inicial presente no *RFC 7049* é possível extrair a seguinte informação:

- ***0xbf***: mapa de tamanho indefinido, seguem-se pares de itens de dados terminados por um *"break"*;
- ***0x6161***: *UTF-8 string*, mais especificamente, *"a"*;
- ***0x01***: *Integer*, mais especificamente, *1*;
- ***0x6162***: *UTF-8 string*, mais especificamente, *"b"*;
- ***0x9f***: array de tamanho indefinido, seguem-se itens de dados terminados por um *"break"*;
- ***0x02***: *Integer*, mais especificamente, *2*;
- ***0x03***: *Integer*, mais especificamente, *3*;

O *byte 0xbf* declara o início de um mapa de tamanho indefinido, cujo fim é indicado pelo *byte 0xff*. Os *bytes* que se seguem representam os pares desse mapa. O primeiro par, *"a": 1* é representado por *0x616101*, onde *"a"* é *0x6161* e *1* é *0x01*. No segundo par, *"b"* é representado por *0x6162* e o *array* de tamanho indefinido *[2,3]* é representado por *0x9f0203*. Tal como acontece no caso do mapa de tamanho indefinido, no *array* de tamanho indefinido o fim é indicado pelo *byte 0xff*. Assim, tem-se *0x61629f0203* como a representação do segundo par do mapa. É então possível concluir, que a representação do mapa *{_ "a": 1, "b": [_ 2, 3]}* em *CBOR byte string* é *0xbf6161016162 9f0203ffff*.

A codificação dos dados e estruturas de dados referidas na Secção 4.1 é feita com *CBOR*, de acordo com o *RFC 7049* (Bormann and Hoffman,2013). No documento *RFC 7049* estão descritas regras para o uso dos *major types* em *CBOR*. De acordo com a *ISO/IEC DIS 18013-5*, cinco dessas regras devem ser implementadas para todas as estruturas da CBOR, como se segue (Iso,2021):

- *Integers* (*Major Types* 0 e 1) devem ser tão pequenos quanto possível;
- A expressão do tamanho nos *Major Types* 2 a 5 deve ser o mais curta possível;
- Dados de tamanho indefinido devem ser transformados em dados de comprimento definido;
- Mapas não podem ter múltiplos pares com a mesma chave;
- Mapas que são usados em operações criptográficas devem ser comunicados numa *CBOR byte string*.

A implementação de *CBOR* é feita com recurso à biblioteca *NuGet* Occil (Occil,2013). Esta biblioteca é uma implementação do *CBOR* em *C#*. O *NuGet* apresenta a implementação dos *major types* presentes no *RFC 7079*, codificação e descodificação *CBOR*, métodos para ler e escrever dados *CBOR* e métodos de conversão entre os *major types*.

4.2.1 *CBOR Object Signing and Encryption (COSE)*

Há uma necessidade de ter serviços básicos de segurança para o formato de dados *CBOR*. O documento *RFC 8152* (Schaad,2017) introduz o conceito de *COSE*, *CBOR Object Signing and Encryption*. Esta especificação descreve como criar e processar assinaturas, códigos de autenticação de mensagens, cifragem usando *CBOR* e como representar chaves criptográficas usando *CBOR*. É utilizada a biblioteca *NuGet* (Schaad,2015).

A estrutura *COSE* está desenvolvida de forma a que haja uma grande quantidade de código comum quando se faz o *parsing* e processamento dos vários tipos de mensagens de segurança. Todas as mensagens de segurança têm como base o *CBOR array type*, onde os três primeiros elementos do *array* têm sempre a seguinte informação (Schaad,2017):

- Conjunto de parâmetros protegidos de *header* envoltos numa *byte string*;
- Conjunto de parâmetros não protegidos de *header* como um mapa;
- O conteúdo da mensagem, que pode estar em texto simples ou texto cifrado.

A identificação do tipo de mensagem *COSE* é feita ou de acordo com o contexto da aplicação, ou com recurso a uma *tag CBOR*. Neste último caso, a *tag* precede a mensagem como se esta fosse um *CBOR major type 6*. A Tabela 2 apresenta as *tags* para as possíveis mensagens *COSE*, de acordo com o *RFC 8152*:

Mensagens *COSE* são construídas tendo em consideração o conceito de camadas. Foi adotado este conceito de modo a separar os diferentes elementos de uma mensagem *COSE*, de acordo com o seu propósito. Existem, por norma, duas camadas: camada de conteúdo, onde se encontra o texto cifrado e informação sobre a mensagem cifrada; camada de recipiente, onde a *CEK (Content Encryption Key)* cifrada e a informação sobre como está cifrada se encontram.

Tabela 2: *Tags COSE*

CBOR tag	*cose-type*	*Data-Item*	*Semantics*
98	cose-sign	COSE_Sign	COSE Sign Data Object
18	cose-sign1	COSE_Sign1	COSE Single Signer Data Object
96	cose-encrypt	COSE_Encrypt	COSE Encrypted Data Object
16	cose-encrypt0	COSE_Encrypt0	COSE Single Recipient
97	cose-mac	COSE_Mac	COSE MACed Data Object
17	cose-mac0	COSE_Mac0	COSE MAC w/o Recipients Object

Outro conceito presente nas mensagens *COSE* são os *buckets*. *Buckets* são estruturas implementadas como mapas *CBOR* que contêm informação sobre o conteúdo da mensagem, algoritmos criptográficos e chaves. Cada camada apresenta dois *buckets*(19):

- **protegidos:** Contém parâmetros sobre a camada onde se encontra que devem ser cifrados. Caso não vá ser usado numa computação criptográfica, este *bucket* está vazio. O valor é obtido envolvendo o mapa *CBOR* numa *byte string*;
- **desprotegidos:** Contém parâmetros sobre a camada que não se encontram cifrados.

De acordo com a *ISO/IEC DIS 18013-5* (Iso,2021), dos *Data-Items* apresentados na Tabela 2, apenas é necessário considerar as estruturas *COSE_Sign1* e *COSE_Mac0*.

COSE_Sign1 é uma das duas estruturas de assinatura apresentadas. Esta é usada quando apenas uma assinatura vai ser aplicada na mensagem. O elemento que procede nesta estrutura é uma *byte string* que representa o valor da assinatura (Schaad,2017):

```
COSE_Sign1 = [
       Headers,
       payload : bstr / nil,
       signature : bstr
  ]
```

Listing 4.2: COSE_Sign1

COSE_Mac0 é uma das duas estruturas *MAC (Message Authentication Code)* apresentadas. Assim como a estrutura *COSE_Sign1*, *COSE_Mac0* também é um *array CBOR*, onde os primeiros três elementos são os mesmos. O quarto elemento é uma *tag* que contém o valor do *MAC* em *byte string*.

```
COSE_Mac0 = [
       Headers,
       payload : bstr / nil,
       tag : bstr
```

```
    ]
```

Listing 4.3: COSE_Mac0

Como foi mencionado no início desta Secção, *COSE* pode ainda ser usado para representar chaves criptográficas. A estrutura das *COSE_Keys* é diferente das restantes estruturas *COSE*. Enquanto que as estruturas referidas acima têm por base de construção o *array CBOR*, a *COSE_Key* tem como base um mapa *CBOR*. De acordo com o *standard ISO/IEC DIS 18013-5*, a estrutura da *COSE_Key* é a seguinte (Iso,2021):

```
COSE_Key = {
    1 => int,          ; kty
    -1 => bstr,        ; crv
    -2 => bstr,        ; x
    ? -3 => bstr/bool, ; y
}
```

Listing 4.4: COSE_Key

O primeiro parâmetro, *kty*, tem a função de identificador da família de chaves que vai ser utilizada para a estrutura *COSE_Key*. Os possíveis valores para esta variável são os seguintes (Schaad,2017):

Tabela 3: Tipos de chaves

Nome	Valor	Descrição
OKP	1	*Octet Key Pair*
EC2	2	*Elliptic Curve Keys w/ x- and y-coordinate pair*
Symmetric	4	Symmetric Keys
Reserved	0	This value is reserved

O portador *mDoc* implementa algoritmos criptográficos que utilizam curvas elípticas (Iso,2021). Assim, apenas são consideradas as chaves do tipo *OKP* e *ECP2*. *ECP2* utiliza duas coordenadas, *x* e *y* com compressão de ponto opcional (enviar apenas a coordenada *x* e resolver coordenada *y* a partir desta). Para curvas do tipo *OKP* apenas é necessária a coordenada *x*. Mais à frente é feita uma análise detalhada sobre a aplicação e funcionamento das curvas elípticas, e os algoritmos que as usam (ver Secção 4.3.1).

O segundo parâmetro, *crv*, contêm o identificador da curva do tipo *ECP 2* ou *OKP*, que vai ser usada com a chave. O *RFC 8152* (Schaad,2017) apresenta os valores possíveis para este parâmetro. É importante notar que o parâmetro *crv* existe porque apenas são consideradas famílias de chaves dos tipos *OKP* e *EC2*.

Os parâmetros *x* e *y* contêm os valores para as coordenadas do ponto da curva elíptica. A coordenada *y* apenas é aplicável caso o parâmetro *kty* seja *EC2*.

4.3 MÓDULOS

Nesta secção são definidos os módulos e as suas respetivas implementações. As estruturas de dados e criptográficas aqui definidas utilizam as das estruturas utilizadas para estabelecer a conexão e transmissão de dados entre o portador *mDoc* e o leitor *mDoc*. Esta definição passa pela descrição destas estruturas de dados enquanto classes, e das variáveis e métodos criados para lhes atribuir funcionalidade.

4.3.1 *Crypto*

Nesta secção são descritos os mecanismos de segurança utilizados e a sua implementação na aplicação. Esta descrição corresponde à apresentação das estruturas de dados e criptográficas definidas para o desenvolvimento destes mecanismos. É utilizada a biblioteca *C# BouncyCastle (BouncyCastle,2000)* para os algoritmos criptográficos aqui utilizados.

MSO

A norma técnica *ISO/IEC DIS 18013-5* define um *Mobile Security Object (MSO)* com o propósito de verificar a integridade e autenticidade dos dados *mDoc* fornecidos às entidades verificadoras (*i.e.,* leitor *mDoc*). O *MSO* contém o *digest* de cada elemento de dados presente no *mDoc*, assim como o certificado que contém a chave pública de autenticação do portador *mDoc* e outras informações de validação. A autoridade emissora assina o *MSO* com uma chave privada, permitindo ao leitor *mDoc* verificar a validade dos elementos de dados *mDoc* recebidos durante a comunicação com o portador *mDoc*, e da autoridade emissora que assinou os dados verificando a sua assinatura.

Cipher Suite

Na norma técnica *ISO/IEC DIS 18013-5* é definida uma *cipher suite* que corresponde ao conjunto de algoritmos e operações utilizados nos mecanismos de segurança implementados. Apesar do mecanismo de transmissão de dados entre dispositivos suportar múltiplas *cipher suites*, a *ISO/IEC DIS 18013-5* descreve apenas uma, que é definida pelo valor "1". A *cipher suite* a ser utilizada é indicada na estrutura de conexão de dispositivos (ver Secção 4.3.2). A *cipher suite* apresenta várias curvas elípticas que podem ser utilizadas na implementação dos mecanismos de segurança. Tendo em conta o objetivo deste trabalho e por uma questão de simplicidade, o portador *mDoc* implementa apenas as seguintes curvas elípticas:

O portador *mDL* não precisa suportar todas as curvas apresentadas na Tabela 4, já o leitor *mDL* tem de suportar todas as curvas obrigatoriamente. A implementação destas curvas vai de acordo com definição da *COSE_Key* descrita na Secção 4.2.1.

Tabela 4: Curvas elípticas implementadas da *cipher suite 1*.

Definição	Especificação	Identificador de curva	Propósito
Curva P-256	FIPS 186-4	IANA COSE registry	ECDH/ECDSA
Curva P-384	FIPS 186-4	IANA COSE registry	ECDH/ECDSA
Curva P-521	FIPS 186-4	IANA COSE registry	ECDH/ECDSA

Cifragem da sessão

A cifragem dos pedidos e respostas mdoc com chaves de sessão atribui autenticação aos dados mdoc a serem transmitidos, protegendo-os assim de ataques de espionagem e alterações. Esta cifragem utiliza *ECDH (Elliptic-curve Diffie-Hellman)* com chaves efémeras de modo a gerar chaves de sessão para a cifragem simétrica autenticada. As chaves são geradas independentemente pelo portador *mDoc* e pelo leitor *mDoc*. O processo de cifragem da sessão tem quatro passos (Iso,2021):

- **Conexão de dados**: O *mDoc* gera um par de chaves efémeras (pública e privada), e inclui na estrutura de conexão de dispositivos a chave efémera pública (ver Secção 4.3.2).
- **Estabelecimento da sessão**: O leitor *mDoc* gera um par de chaves efémeras (pública e privada) usando a curva elíptica identificada pelo portador *mDoc* no passo anterior. O leitor *mDoc* cifra o pedido *mDoc* com a chave de sessão, e envia-o para o portador *mDoc* juntamente com a chave efémera pública do leitor *mDoc* na estrutura *SessionEstablishment*:

```
SessionEstablishment = {
    "eReaderKey" : bstr ; Chave publica do leitor mdoc em bytes,
    "data" : bstr ; Pedido mdoc cifrado,
}
```

Listing 4.5: SessionEstablishment

  O portador *mDoc* utiliza a informação desta estrutura para gerar as chaves de sessão e decifra o pedido *mDoc*;

- **Dados de sessão**: O portador *mDoc* cifra a resposta *mDoc* com a chave de sessão apropriada e envia-a para o leitor mdoc numa mensagem de resposta protegida pela chave de sessão derivada no passo anterior:

```
SessionData = {
     ? "data" : bstr ; Pedido ou resposta mdoc cifrados request,
     ? "status" : uint ; Codigo de estado,
}
```

Listing 4.6: SessionData

Opcionalmente, o portador mdoc e o leitor mdoc poderão continuar a trocar mensagens de dados de sessão contendo pedidos e respostas mdoc;

- **Encerramento da sessão**: A sessão é terminada caso ocorra um dos seguintes cenários:
  - Depois de um *time-out* por falta de envio de mensagens entre o portador mdoc e o leitor mdoc. Este *time-out* é de 300 segundos;
  - Caso o portador mdoc não queira aceitar mais pedidos mdoc;
  - Caso o leitor mdoc não pretenda enviar mais pedidos mdoc.

  Tanto o portador mdoc como o leitor mdoc descartam as suas chaves de sessão, e o canal utilizado para a recuperação de dados entre dispositivos é fechado.

Os pedidos *mDoc* são cifrados com a chave de sessão do leitor *mDoc*, enquanto que as respostas *mDoc* são cifradas com a chave de sessão do portador *mDoc*. Como se trata de uma cifragem simétrica, o portador *mDoc* calcula a sua chave de sessão e a chave de sessão do leitor *mDoc* utilizando os algoritmos *ECKA-DH (Elliptic Curve Key Agreement Algorithm - Diffie-Hellman)* e *HKDF (Hash based Key Derivation Function)*, de modo a poder decifrar os pedidos *mDoc* e cifrar as respostas *mDoc*. O portador *mDoc* utiliza o algoritmo *ECKA-DH* com a sua chave efémera privada e a chave efémera pública do leitor *mDoc* para gerar um segredo partilhado. Este segredo partilhado cujo valor apenas é conhecido pelo portador *mDoc* e leitor *mDoc*, e é utilizado como *input* no algoritmo *HKDF* para derivar as chaves de sessão do portador *mDoc* e do leitor *mDoc*. A derivação das chaves de sessão é feita de acordo com os seguintes parâmetros:

SKDevice:

- Hash: SHA-256,
- IKM: Segredo partilhado,
- salt: SHA-256(*SessionTranscriptBytes*),
- info: “SKDevice” (codificada como uma *UTF-8 string*),
- Tamanho: 32 octetos

SKReader:

- Hash: SHA-256,
- IKM: Segredo partilhado,
- salt: SHA-256(*SessionTranscriptBytes*),
- info: “SKReader” (codificada como uma *UTF-8 string*),

- Tamanho: 32 octetos

SessionTranscriptBytes é a estrutura *SessionTranscript*, definida em 4.3.1, codificada em *CBOR bytestring*.

Para a cifragem do pedido e resposta *mDoc* é utilizado o algoritmo *AES-256-GCM (GCM: Galois Counter Mode)*. O leitor *mDoc* cifra os seus pedidos *mDoc* com a chave *SKReader*, e o portador *mDoc* gera esta mesma chave de modo a decifrar o pedido. A resposta mdoc a este pedido é cifrada com a chave de sessão *SKDevice*.

O *IV (Initialization Vector)* usado para a cifragem é a concatenação entre um identificador e um contador de cifragens que tem o tamanho padrão de 12 *bytes*. O identificador é um valor de 8 *bytes*, cujo valor para o portador mdoc é *0x00 0x00 0x00 0x00 0x00 0x00 0x0 0x01*, e para o leitor mdoc é *0x00 0x00 0x00 0x00 0x00 0x00 0x0 0x00*. O contador é um *big-endian unsigned integer* de 4 *bytes*. O valor inicial é 1 e é incrementado antes de cada cifragem (não inclui a primeira cifragem). Caso sejam usadas chaves de sessão diferentes o valor do contador reinicia com o valor 1.

As chaves efémeras do portador *mDoc* são geradas utilizando as curvas da Tabela 4 e estão formatadas de acordo com a estrutura *COSE_Key* (ver Secção 4.2.1). Para gerar estas chaves foi criada a classe *Key* que consiste numa implementação da *COSE_Key* tendo em conta as curvas elípticas consideradas:

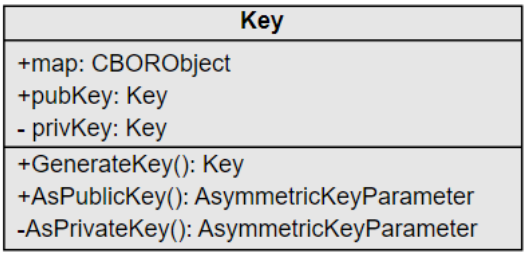


Figura 5: Classe *Key*

Esta classe gera um par de chaves efémeras de acordo com a curva elíptica utilizada. A variável *map* representa este par de chaves como um objeto *CBOR* de acordo com a estrutura *COSE_Key*. O método estático *GenerateKey* recebe o identificador da curva elíptica a utilizar e gera um objeto *Key*. Este objeto *Key* tem outras duas variáveis do tipo *Key* que tem o propósito de armazenar as chaves efémeras pública e privada, separadamente. Quando é necessário utilizar as chaves nos processos criptográficos, é utilizado o método *AsPublicKey* para a chave efémera pública, e o método *AsPrivateKey* para a chave efémera privada.

Para a geração do segredo partilhado e derivação das chaves de sessão foi criada a classe *Crypto*, que utiliza as classes *HKDF*, *AesGcm256* e *Key* nestes processos criptográficos.

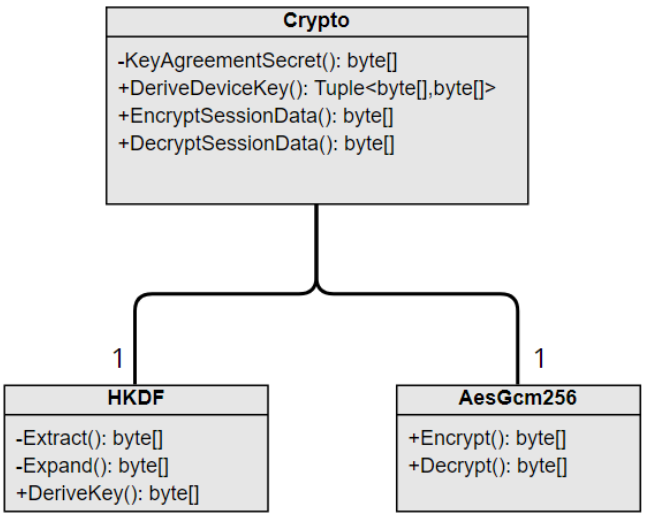


Figura 6: Classe *Crypto*

As classes *HKDF* e *AesGcm256* implementam os algoritmos de derivação de chaves criptográficas e de cifragem de pedidos e respostas *mDoc*, respetivamente. A classe *Crypto* implementa o método *KeyAgreement-Secret*, que utiliza a chave efémera privada do portador *mDoc* e a chave efémera pública do leitor *mDoc* para gerar o segredo partilhado. O método *DeriveDeviceKey* vai receber o segredo partilhado como *input* e vai derivar as chaves de sessão do portador *mDoc* e do leitor *mDoc*, utilizando *HKDF*.

A cifragem e decifragem das mensagens *mDoc* é feita com a classe *AesGcm256* que implementa o algoritmo *AES-256-GCM*. *Crypto* implementa os métodos *EncryptSessionData* e *DecryptSessionData* para cifrar e decifrar as mensagens *mDoc*.

Autenticação leitor mDoc

O portador *mDoc* confirma a autenticação do leitor *mDoc* e do pedido *mDoc*. Uma chave privada de autenticação armazenada no leitor *mDoc* é utilizada para a autenticação do leitor e do pedido *mDoc*. A chave pública de autenticação do leitor *mDoc* está presente num certificado enviado para o portador *mDoc* no pedido *mDoc*. A autenticação é feita através de *ECDSA (Elliptic Curve Digital Signature Algorithm)*, e são utilizadas as curvas da Tabela 4.

Os dados a serem autenticados são a estrutura *ReaderAuthentication*:

```
ReaderAuthentication = [
    "ReaderAuthentication" : str,
    SessionTranscript,
    ItemRequestBytes: bstr
```

```
]
```

Listing 4.7: ReaderAuthentication

O *array CBOR* que define a estrutura *ReaderAuthentication* apresenta três elementos: a *string "ReaderAuthentication"*, a estrutura *SessionTranscript* e a *CBOR bytestring ItemRequestBytes*. *ItemRequestBytes* corresponde à estrutura enviada no pedido *mDoc* e a estrutura *SessionTranscript* é definida da seguinte forma:

```
SessionTranscript = [
    DeviceEngagementBytes : bstr,
    EReaderKeyBytes : bstr,
    Handover = Null
]
```

Listing 4.8: SessionTranscription

SessionTranscript é um *array CBOR* onde o primeiro elemento é a estrutura *DeviceEngagement* da secção 4.3.2, codificada em *CBOR bytestring*, e o segundo elemento é a chave efémera pública da Secção 4.3.1, codificada em *CBOR bytestring* e enviada como parte do pedido *mDoc*. O último elemento corresponde ao *handover*, que corresponde a uma mensagem com informação sobre a tecnologia de conexão de dispositivos. Neste caso, o seu valor é nulo tendo em conta que se vai utilizar *QR code* como tecnologia de conexão de dispositivos.

A estrutura é codificada em *CBOR bytes* e depois é calculada a sua assinatura. *ReaderAuthentication* não é enviada como parte do pedido *mDoc*, apenas a sua assinatura. A assinatura é calculada com a chave privada de autenticação do leitor *mDoc*, e a verificação da assinatura é feita pelo portador *mDoc* com a chave pública de autenticação. A chave pública de autenticação encontra-se num certificado armazenado no *bucket* desprotegido da estrutura *COSE_Sign1* identificada como *ReaderAuth*. No *bucket* protegido encontra-se o identificador do algoritmo de assinatura usado. O *payload* desta estrutura é nulo e o último elemento é a assinatura da estrutura *ReaderAuthentication* codificada em *CBOR bytes*.

A autenticação do leitor *mDoc* leva à criação da classe *SignMessage*. Esta classe implementa a estrutura *COSE_Sign1* e permite a assinatura de dados e verificação de assinaturas.

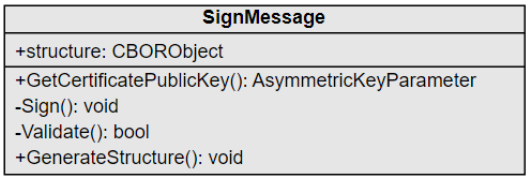


Figura 7: Classe *Crypto*

A variável *structure* é utilizada para armazenar a estrutura *ReaderAuth*. Apesar desta estrutura não precisar de ser gerada, pois é recebida pelo portador *mDoc* no pedido *mDoc*, é implementado o método *GenerateStructure* que é necessário na autenticação do portador *mDoc* (ver Secção 4.3.1). É implementado o método *Sign* que utiliza a chave privada de assinatura do portador *mDoc* para assinar dados em formato *byte string*. Por fim, o método *Validate* recebe como *input* os dados a validar em formato *byte string* e a chave pública de autenticação do leitor *mDoc*, e retorna um *bool* que indica se a assinatura é válida.

Autenticação do portador mdoc

Assim como o portador *mDoc* deve confirmar a autenticidade do leitor e do pedido *mDoc*, também o leitor *mDoc* confirma a autenticidade do portador *mDoc* e dos dados que este lhe envia. Uma chave privada de autenticação armazenada no portador *mDoc* deve ser utilizada para a autenticação do portador e da resposta *mDoc*. A chave pública de autenticação está presente num certificado no *MSO* enviado para o leitor *mDoc* na resposta *mDoc*. A autenticação é feita através de *ECDSA* e são utilizadas as curvas da tabela 4. Os dados a serem autenticados são a estrutura *DeviceAuthentication* a seguir definida:

```
DeviceAuthentication = [
    "DeviceAuthentication",
    SessionTranscript,
    DocType,
    DeviceNameSpacesBytes
]
```

Listing 4.9: DeviceAuthentication

O array CBOR que define a estrutura *DeviceAuthentication* apresenta três elementos: a *string DeviceAuthentication*, a estrutura *SessionTranscript* (ver Secção 4.3.1), uma *string* a identificar o tipo do documento a ser autenticado e os elementos de dados a serem autenticados e os seus respetivos *namespaces* como se encontra na resposta *mDL* (ver Secção 4.1.3).

A estrutura é codificada em *CBOR bytes* e depois é calculada a sua assinatura. *DeviceAuthentication* não é enviada como parte do pedido *mDoc*, apenas a sua assinatura. A assinatura é calculada com a chave privada de autenticação do portador e é armazenada na estrutura *COSE_Sign1* identificada como *DeviceAuth*. No *bucket* protegido encontra-se o identificador *COSE* do algoritmo de assinatura usado. O *payload* desta estrutura é nulo.

Para esta implementação também é usada a classe *SignMessage*, porém com um objetivo diferente. Enquanto que na autenticação do leitor *mDoc* esta classe é utilizada para a verificação de assinaturas, na autenticação do portador *mDoc* é utilizada para gerar a estrutura *DeviceAuth* e gerar uma assinatura sobre a estrutura *DeviceAuthentication*. O método *GenerateStructure* é utilizado para gerar *DeviceAuth* e o método *Sign* é utilizado para assinar a estrutura *DeviceAuthentication*, com a chave privada de autenticação do portador *mDoc*.

4.3.2 *Engagement*

Este módulo define a estrutura de conexão de dispositivos, seus constituintes e a tecnologia *QR Code*. São descritas as classes, variáveis e métodos implementados para gerar esta estrutura e os elementos que fazem parte dela.

Estrutura de conexão de dispositivos

A estrutura de conexão de dispositivos contém a informação necessária para a conexão entre o portador *mDL* e o leitor *mDL*. De acordo com a norma técnica *ISO/IEC DIS 18013-5* (Iso,2021), esta estrutura encontra-se codificada em *CBOR* e formatada da seguinte forma:

```
DeviceEngagement =
{
    0: tstr,
    1: Security,
    ? 2: DeviceRetrievalMethods,
    ? 3: ServerRetrievalMethods,
    ? 4: ProtocolInfo,
}
```

Listing 4.10: DeviceEngagement

A estrutura é representada por um *map*, onde o primeiro par chave-valor corresponde à versão da estrutura de conexão de dispositivos, cujo valor, de acordo com a versão da norma técnica *ISO/IEC DIS 18013-5* utilizada, é "1.0". De seguida, o par com valor *"Security"* representado por um *array* com dois elementos. O primeiro elemento é o identificador da *cipher suite* (ver Tabela 4) e o segundo elemento é a chave efémera pública do portador *mDoc* utilizada para criar as chaves de sessão (ver Secção 4.3.1), em formato *tagged CBOR bytes*.

O terceiro elemento, *"DeviceRetrievalMethods"*, corresponde a um *array* que contém informação sobre o papel do portador *mDoc* na dinâmica da transmissão de dados *BLE*, e é usado quando a conexão entre dispositivos é feita com *QR code*. De acordo com a *ISO/IEC DIS 18013-5* (Iso,2021), *Bluetooth Low Energy* corresponde ao tipo com valor *"2"* e versão de valor *"1"*. As opções adicionais correspondem a uma estrutura que indica quais os modos suportados pelo *BLE* e o seu respetivo *UUID (Universal Unique Identifier)*. Como é referido no Capítulo 4.3.4, o portador *mDoc* apenas suporta o modo de servidor periférico, o que implica que esta estrutura apenas tenha o *UUID* para este modo.

A estrutura *"DeviceEngagement"* apresenta ainda as estruturas *"ServerRetrievalMethods"* e *"ProtocolInfo"*. A interface de interação entre o leitor *mDoc* e a autoridade emissora *"ServerRetrievalMethods"* não se encontra no escopo da comunicação entre o protador *mDoc* e o leitor *mDoc*. Assim, não é necessária a estrutura *"ServerRetrievalMethods"*. Por último, a estrutura *"ProtocolInfo"* contém dados *RFU (Reserved for Future Use)* e

serve para definir informação protocolar, que garanta que a estrutura de conexão de dispositivos seja aplicável a *updates* futuros.

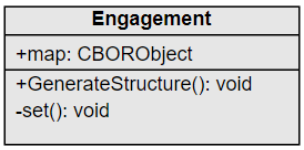


Figura 8: Classe *Engagement*

Para representar a estrutura de conexão entre dispositivos foi implementada a classe *Engagement*. A variável *map* é um mapa *CBOR* que é gerado com o método *GenerateStructure*. É de notar que os campos opcionais *ServerRetrievalMethods* e *ProtocolInfo* não são considerados, pelo facto da sua presença não ser necessária para fornecer a informação relevante ao leitor *mDoc*.

QR code

A implementação da tecnologia de conexão de dispositivos passa pela criação de uma única classe com um único método. A classe *QRCode* implementa o método *Generate*, que recebe como *input* a *string* a ser exibida no *QR code*. Esta *string* vai corresponder à concatenação da *string "mdoc:"* com a estrutura de conexão entre dispositivos, codificada em *Base64Url*, de acordo com a norma técnica *ISO/IEC DIS 18013-5*. Para a implementação do *QR Code* é utilizada a biblioteca *C# QRCoder* (Raffael,2013).

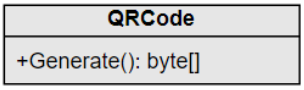


Figura 9: Classe *QRCode*

4.3.3 *Holder*

O módulo *Holder* implementa o armazenamento de variáveis criptográficas e dados *mDoc*, a inicialização dos mecanismos de segurança, e o processamento do pedido *mDoc* e criação da resposta *mDoc*. Este módulo implementa duas classes, *Holder* e *Retrieval*.

Holder

Holder adiciona uma camada de abstração entre o portador *mDoc* e os mecanismos de segurança e estruturas de dados da aplicação. Esta classe contém toda a informação relativa ao portador *mDoc* que esteja relacionada com mecanismos de segurança e estruturas *CBOR* e *COSE*, isto inclui a estrutura de conexão de dispositivos e o pedido *mDoc* recebido.

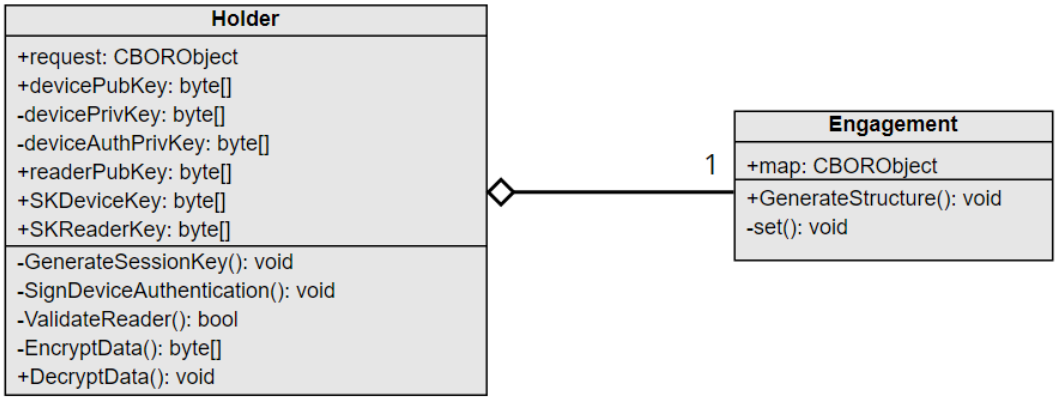


Figura 10: Classe *Holder*

A Figura 10 representa a classe *Holder* e as variáveis e métodos que esta implementa. *Holder* é uma das primeiras classes a ser inicializadas pelo portador *mDoc*. Tendo isto em conta, esta classe tem uma instância da classe *Engagement* que representa a estrutura de conexão de dispositivos exibida no *QR Code* (ver Secção 4.3.2). A variável *request* é utilizada para armazenar o pedido *mDoc* como um objeto *CBOR*. As restantes variáveis correspondem a chaves criptográficas utilizadas nos mecanismos de segurança. As chaves efémeras pública e privada do portador *mDoc* (ver Secção 4.3.1) são representadas pelas variáveis *devicePubKey* e *devicePrivKey*, respetivamente. A variável *deviceAuthPrivKey* corresponde à chave de autenticação privada do portador *mDoc*, e a variável *readerPubKey* à chave efémera pública do leitor *mDoc*. *SKDeviceKey* e *SKReaderKey* são as chaves de sessão do portador *mDoc* e do leitor *mDoc*.

Os métodos implementados na classe *Holder* utilizam as variáveis criptográficas do parágrafo anterior para os processos de geração de chaves de sessão, autenticações do portador *mDoc* e do leitor *mDoc*, e cifragem e decifragem dos pedidos e respostas *mDoc*. O método *GenerateSessionKey* é responsável por gerar as chaves de sessão do portador e do leitor *mDoc*, utilizadas para cifrar e decifrar as mensagens *mDoc* (*i.e.,* pedido *mDoc* e resposta *mDoc*). *SignDeviceStructure* gera e assina a estrutura *DeviceAuthentication* com a chave privada de autenticação do portador *mDoc*, e gera a estrutura *DeviceAuth* enviada na resposta *mDoc*. Para validar o leitor *mDoc* foi criado o método *ValidateReader*, que gera a estrutura *ReaderAuthentication* e verifica a assinatura sobre esta estrutura, com a chave pública de autenticação do leitor *mDoc*. Por fim, os métodos *EncryptData* e *DecryptData* tem o propósito de cifrar e decifrar as mensagens *mDoc*. *EncryptData* utiliza a chave de sessão do portador *mDoc* para cifrar as respostas *mDoc*, enquanto que *DecryptData* utiliza a chave de sessão do leitor *mDoc* para decifrar os pedidos *mDoc*.

Retrieval

Retrieval é a classe com a qual a *UI* comunica diretamente para processar o pedido *mDoc* e gerar a resposta *mDoc* apropriada, e para executar os métodos do *Holder* responsáveis pelos mecanismos de segurança. A classe *Retrieval* e as suas variáveis e métodos estão ilustrados na Figura 11.

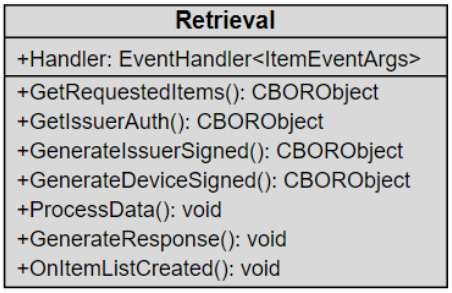


Figura 11: Classe *Retrieval*

Retrieval tem uma única variável que é um *EventHandler*. Esta variável executa uma ação em resposta a um evento. Neste caso em específico, a variável *Handler* tem como objetivo acionar o método *OnItemListCreated* que enviam para a *UI (User Interface)* os dados solicitados no pedido *mDoc*. Assim que o portador *mDoc* recebe o pedido *mDoc*, *OnItemListCreated* envia os dados *mDoc* solicitados para a *UI* de modo a que estes possam ser exibidos no ecrã do *smartwatch*. O portador *mDoc* utiliza o método *ProcessData* para invocar os métodos do *Holder* que geram as chaves de sessão, autenticam o leitor *mDoc* e decifram o pedido *mDoc*. *GetRequestedItems* retorna o valor dos dados *mDoc* para fazerem parte das estruturas *IssuerSigned* e *DeviceSigned*. Os métodos *GenerateIssuerAuth* e *GenerateDeviceSigned* retornam a *MSO* armazenada no portador *mDoc* e o estrutura *DeviceAuth*, de modo a que possam ser utilizadas para gerar as estruturas *IssuerSigned* e *DeviceSigned*, respetivamente. *GenerateIssuerSigned* gera a estrutura *IssuerSigned*, e *GenerateDeviceSigned* a estrutura *DeviceSigned*. Por fim, o método *GenerateResponse* os métodos referidos anteriormente para gerar uma resposta *mDoc*. A resposta *mDoc* é posteriormente enviada pelo canal *BLE* para o leitor *mDoc*.

4.3.4 *BLE*

BLE é um módulo específico à plataforma *wearable WearOS*. As classes, variáveis e métodos utilizam bibliotecas *Android* na sua implementação. Aqui é implementado o canal *BLE (Bluetooth Low Energy)* que vai permitir a transmissão das mensagens *mDoc* (*i.e.,* pedido *mDoc* e resposta *mDoc*) entre o portador *mDoc* e o leitor *mDoc*.

De acordo com a norma técnica *ISO/IEC DIS 18013-5* (Iso,2021), *BLE* é uma das tecnologias de transmissão de dados que pode ser utilizada para a transmissão de pedidos e respostas *mDoc*. A utilização desta tecnologia segue a descrição da *Bluetooth Core Specification* (Woolley,1999).

O portador *mdoc* funciona como um servidor periférico no contexto *BLE*. Isto significa que o portador fica à "espera" que outros dispositivos se conectem a ele. Sendo assim, apenas este modo foi implementado. Para a implementação do servidor periférico é utilizada a *API* de *BLE* fornecida pela plataforma *Android*. São criadas duas classes para esta implementação: *Advertiser* e *Peripheral*. A criação destas classes permite a criação de um canal *BLE* de comunicação portador-leitor.

Advertiser

Esta classe implementa o *Bluetooth Low Energy Advertiser*. É da responsabilidade desta classe a implementação de métodos que permitam ao portador enviar pacotes de dados *BLE* para os dispositivos ao seu redor (*advertising*). Os pacotes de dados *BLE* contém o identificador do modo *BLE* implementado pelo portador *mDoc* e o seu *PSM (Protocol Service Multiplexor)*. Como o portador *mDoc* tem o papel de servidor periférico, o identificador corresponde ao *UUID* que representa este modo. Este *UUID* é o mesmo que se encontra na estrutura de conexão de dados (ver Secção 4.3.2). O *PSM* é o valor que é usado para estabelecer a conexão *BLE* entre o portador *mDoc* e o leitor *mDoc*. O leitor *mDoc* recebe estes pacotes *BLE* e com a informação presente neles conecta-se ao portador *mDoc*.

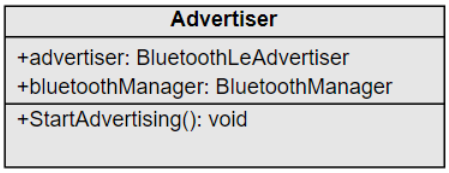


Figura 12: Classe *Advertiser*

A variável *advertiser* é uma instância da classe *BluetoothLeAdvertiser* que implementa o *advertiser BLE*. Esta variável em conjunto com a variável *bluetoothManager*, que implementa um gestor *Bluetooth*, são necessárias para inicializar o processo de *advertising*. O *Advertising* é iniciado pelo método *StartAdvertising*. Este método recebe o *PSM* e o *UUID* do modo servidor periférico como argumentos, e envia pacotes *BLE* que contém estes elementos de dados.

Peripheral

A classe *Peripheral* implementa o modo de servidor periférico para o portador *mDoc* utilizando as classes da *API Android* e outras subclasses, uma delas, a classe *Advertiser*, já descrita anteriormente. A transmissão de dados que a classe *Peripheral* implementa é feita utilizando o método de transmissão *L2CAP* (*Logical Link Control and Adaptation Layer Protocol*). Para a transmissão deve ser estabelecido um *CoC*(*Connection-Oriented Channel*) entre o portador *mDoc* e o leitor *mDoc* de forma a que estes possam trocar dados relevantes à aplicação. A implementação do protocolo *L2CAP* é feita seguindo os requisitos da norma técnica *ISO/IEC DIS 18013-5* (Iso, 2021) e da *Bluetooth Core Specification* (Woolley,1999).

De modo a estabelecer o *CoC* entre o portador *mDoc* e o leitor *mDoc* são utilizadas *threads*, pelo que é necessário implementar as seguintes subclasses:

- ***Advertiser:*** Implementação do *BLE Advertiser* (ver Secção 4.3.4);
- ***AcceptThread:*** Classe que implementa uma *thread* que inicializa o *server socket* e o *advertisement* (ver Secção 4.3.4);

- ***ConnectedThread*** Classe que implementa uma *thread* que gere a conexão *BLE* portador-leitor. Isto implica a receção e envio de dados por parte do portador;

A classe *Peripheral*, assim como a classe *Holder*, é uma das primeiras classes a ser inicializadas pelo portador *mDoc*. O processo de *advertising* é iniciado assim que uma instância *Peripheral* é criada. Isto permite que o leitor *mDoc* compare o *UUID* presente nos pacotes *BLE* com o *UUID* que recebeu na estrutura de conexão de dispositivos. Esta comparação permite ao leitor *mDoc* saber se o dispositivo ao qual se está a tentar conectar é o correto. Assim que estabelecida a conexão, o portador *mDoc* e o leitor *mDoc* podem iniciar a transmissão de dados *mDoc*. O leitor *mDoc* pode enviar mais que um pedido *mDoc* pelo canal *BLE*. Porém, quando o leitor *mDoc* envia um pedido *mDoc*, deve esperar pela resposta *mDoc* do portador *mDoc* antes de enviar outro pedido *mDoc*. Caso não haja troca de mensagens *mDoc* entre o portador *mDoc* e o leitor *mDoc* num período de 300 segundos, é encerrada a conexão. A Figura 13 representa o fluxo de trabalho do *Peripheral*, desde que a sua instância é criada até ao fim da conexão portador-leitor:

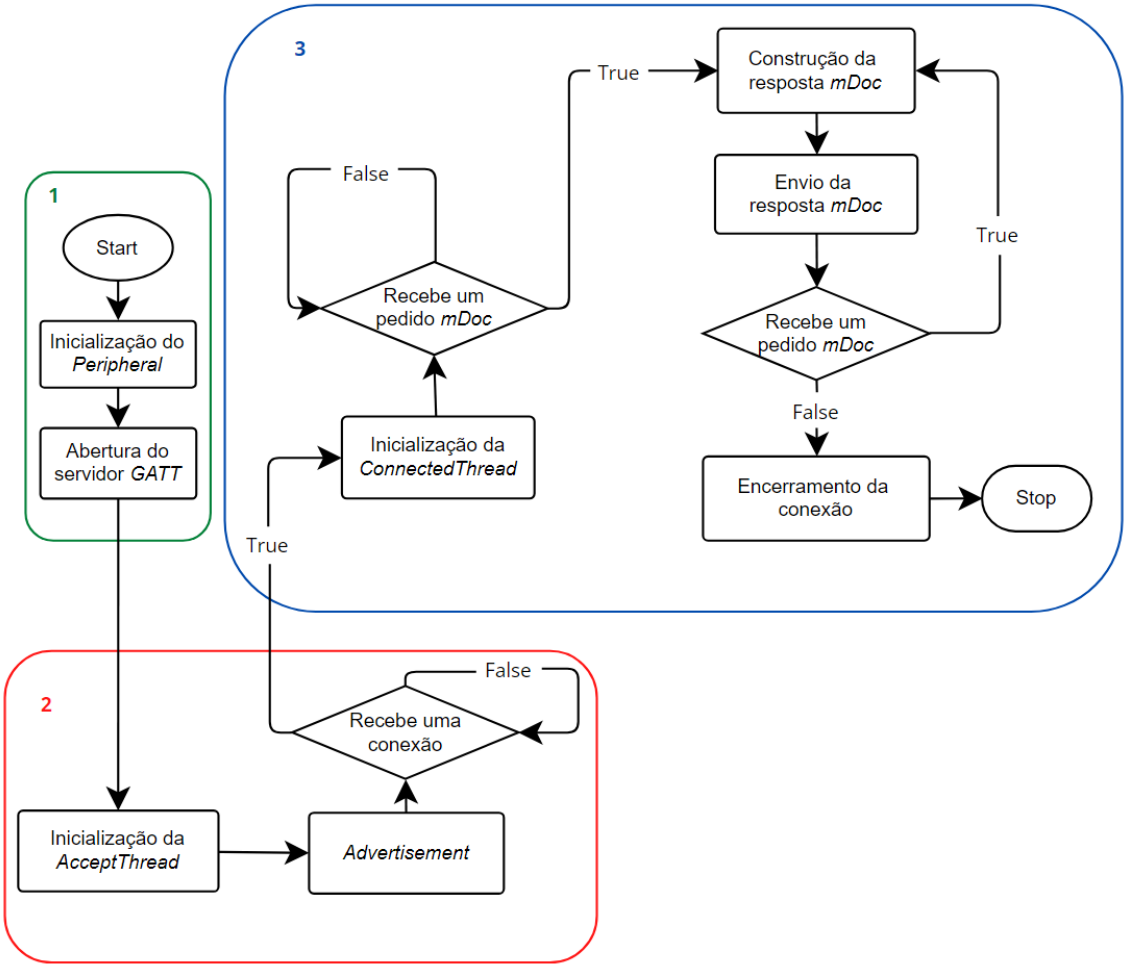


Figura 13: Fluxo de trabalho do *Peripheral*

De acordo com a Figura 13, o fluxo de trabalho pode ser divido em três partes, sendo estas ser consideradas diferentes estados do *Peripheral*:

- **1ª fase**: Fase representada a verde no diagrama. Esta é a fase de inicialização dos recursos necessários para os processos de conexão e transferência de dados. Nesta fase é feita inicialização dos recursos necessários para se poder realizar a conexão *BLE*. Assim, é nesta fase que é criado o *UUID* do serviço que vai conter o valor do *PSM* e se dá a inicialização do servidor *GATT* (*Generic Attribute Profile*) de acordo com o *Bluetooth Core Specification* (Woolley,1999). O servidor *GATT* permite o armazenamento de dados localmente que podem ser acedidos por um cliente *GATT* através de uma conexão *BLE*. Estes dados correspondem a serviços *BLE* que contém características *BLE*. Neste caso, o serviço *BLE* é o modo servidor periférico com um *UUID* associado e a sua característica é o *PSM*. O *PSM* só é atribuido o serviço *BLE* na 2ª fase depois de ser criado o *server socket*. O serviço *BLE* é registado no servidor *GATT* de modo a poder ser utilizado no processo de *advertising*.
- **2ª fase**: Fase representada a vermelho no diagrama. Corresponde à fase da inicialização da *AcceptThread*, que vai permitir o início do *advertisement*, abertura do *server socket* e criação da *socket* para a comunicação portador-leitor. A 2ª fase começa pela inicialização da *AcceptThread* que cria o *server socket BLE* atribuindo o seu *PSM* ao serviço *BLE* criado e inicializa logo de seguida o processo de *advertisement*. O processo de *advertisement* consiste na partilha de pacotes *BLE* com o *UUID* do serviço *BLE* que contém o valor do *PSM*. O leitor *mDoc* vai comparar o *UUID* que recebeu na estrutura de conexão de dispositivos com o *UUID* dos pacotes. Caso os *UUID's* sejam iguais, o leitor *mDoc* vai verificar o valor do *PSM* do serviço *BLE* e vai usá-lo para se conectar ao *server socket BLE* do portador *BLE*. Assim que a conexão seja bem sucedida, é criada uma *BLE socket* entre os dois dispositivos e dá-se inicio a fase de transmissão de dados.
- **3ª fase**: Fase representada a azul no diagrama. É a fase que corresponde à inicialização da *ConnectedThread* que vai permitir a transmissão de dados entre o portador *mDoc* e o leitor *mDoc*. Durante esta fase o portador *mDoc* recebe pedidos *mDoc* e envia as respostas *mdoc* apropriadas. A 3ª fase corresponde à transmissão de dados entre o portador *mDoc* e o leitor *mDoc*. Esta fase começa com a inicialização da *ConnectedThread*, e a criação de *streams* de *input* e *output* para receber pedidos *mDoc* e enviar respostas *mDoc*, respetivamente. Os dados enviados pelo leitor *mDoc* correspondem à estrutura *SessionEstablishment* cifrada com a chave de sessão do leitor *mDoc* (ver Secção 4.3.1), que contém o pedido *mDoc* e a chave efémera pública do leitor *mDoc*. Estes dados *mDoc* são enviados para a classe *Retrieval* de modo a serem decifrados e ser criada a resposta *mDL* (ver Secção 4.3.3). A resposta é enviada para o leitor *mDoc* numa estrutura *SessionData* cifrada com a chave de sessão do portador *mDoc* (ver Secção 4.3.1) através da *stream* de *output*. Após enviada a resposta, o portador volta a ficar à espera de receber mais pedidos. Esta espera pode ser terminada manualmente ou automaticamente se depois de 300 segundos não houver receção, ou envio de dados entre o portador e o leitor. Em qualquer um dos casos é terminada a conexão entre o portador *mDoc* e o leitor *mDoc*.

A Figura 15 representa a implementação classe *Peripheral* e as subclasses a ela associadas:

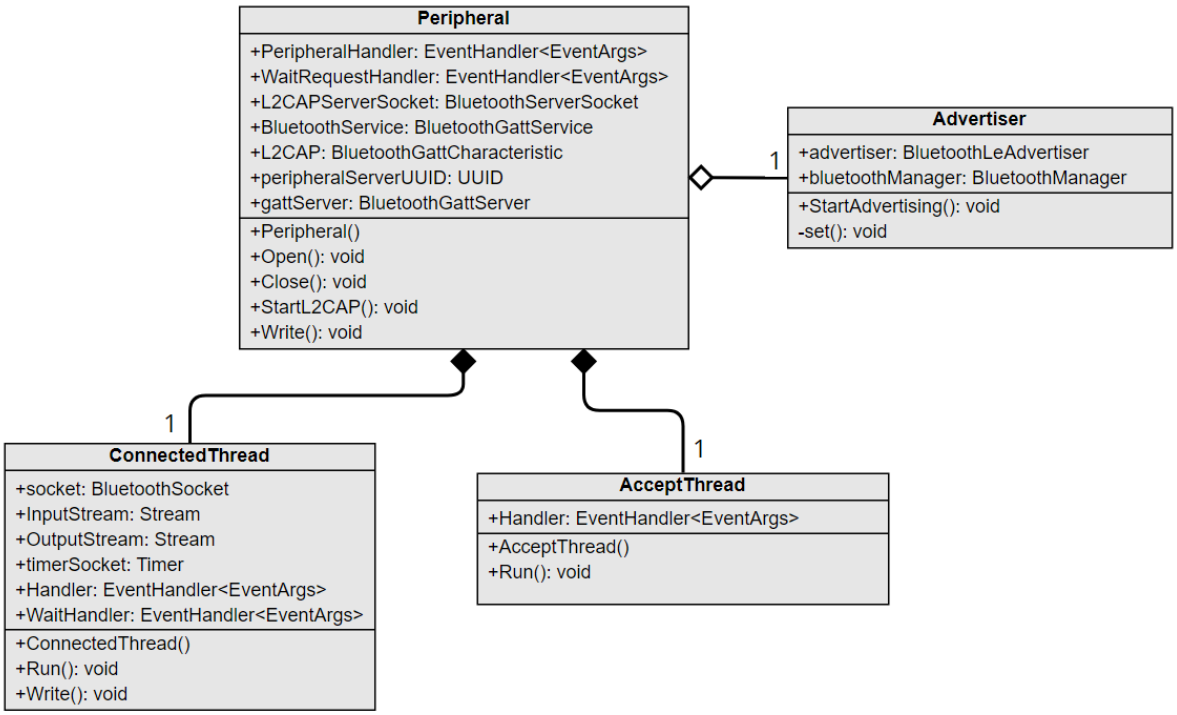


Figura 14: Classe *Peripheral*

Peripheral implementa duas variáveis do tipo *EventHandler* para comunicar mudanças no estado da conexão *BLE* com a *UI*. A variável *PeripheralHandler* comunica ao *UI* que um pedido *mDoc* foi recebido. *WaitRequestHandler* comunica ao *UI* que foi iniciada um coenxão *BLE* e que o portador *mDoc* está pronto para receber pedidos *mDoc*. A *server socket* é armazenada na variável *L2CAPServerSocket*. O servidor *GATT* corresponde à variável *gattServer*, e o seu serviço *BLE* e característica *BLE* às variáveis *BluetoothService* e *L2CAP*, respetivamente. O *UUID* do serviço *BLE* é a variável *peripheralServerUUID*.

A criação de uma instância da classe *Peripheral* inicializa o *BluetoothService* e gera o respetivo *UUID*. O método *Open* inicia o servidor *GATT*, regista o *BluetoothService* e inicializa uma instância da classe *Advertiser* à qual atribui o *UUID* do *BluetoothService*. *Close* é o método que termina a conexão *BLE* e encerra o servidor *GATT*. *StartL2CAP* inicializa a *AcceptThread* para a criação da conexão *BLE*. Por fim, o método *Write* é utlizado para enviar as respostas *mDoc* pelo canal *BLE*.

A classe *AcceptThread* consiste na implementação de uma *thread* que cria o *server socket* e inicia o *advertisement*. Nesta *thread*, o portador *mDoc* fica à espera de receber uma conexão *BLE* de um dispositivo que tenha o *PSM* de conexão, ao mesmo tempo que inicializa o *advertisement* com o serviço adequado. Do estabelecimento da conexão *BLE* resulta uma *BluetoothSocket*, que vai ser utilizada na transmissão de dados. Por fim, é inicializada uma instância da classe *ConnectedThread* com a *BluetoothSocket* como *input*.

Depois de criado o canal *BLE*, a classe *ConnectedThread* gera uma *stream* de *input*, *InputStream*, e um *stream* de *output*, *OutputStream*. A primeira vai permitir ao portador *mDoc* receber pedidos *mDoc*, e a segunda enviar resposta *mDoc*. O método *Write* permite o envio de dados pela *OutputStream* e é invocado quando o método *Write* da classe *Peripheral* é executado. É implementado um *timer* de 300 segundos de modo a controlar o período de inatividade na conexão portador-leitor. Caso o portador *mDoc* receba um pedido *mDoc* e envie resposta *mDoc*, volta a iniciar o período de espera por mais pedidos *mDoc*.

4.3.5 *Droid*

A classe *Holder* contém uma instância da estrutura de conexão de dispositivos (ver Secção 4.3.3) que necessita de informação sobre a tecnologia de transmissão de dados *BLE*, antes de poder ser exibida no *QR code*. Como o módulo *BLE* utiliza bibliotecas *Android*, implementa-se a classe *Droid* que permite a comunicação entre os módulos *Holder* e *BLE*.

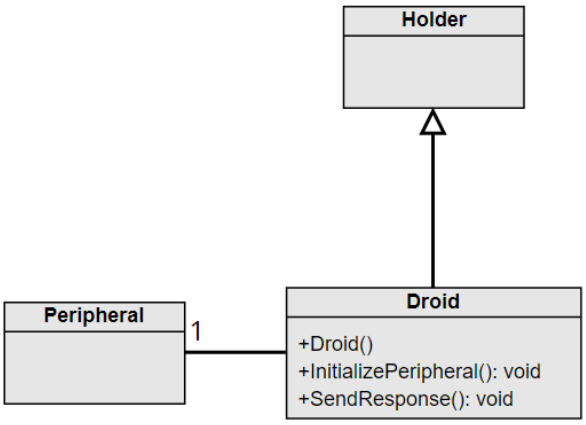


Figura 15: Classe *Droid*

A classe *Droid* herda a classe *Holder*, logo tem uma instância da classe *Engagement*, que representa a estrutura de conexão de dispositivos. *Droid* tem também uma variável que representa uma instância da classe *Peripheral*. A inicialização de uma instância *Droid* provoca a inicialização do *Peripheral*. Os dados *mDoc* referentes ao *BLE* são armazenados na estrutura de conexão de dispositivos. Depois deste passo, a estrutura de conexão de dispositivos está pronta para ser partilhada por *QR code*. O método *InitializePeripheral* invoca o método *Open* do *Peripheral* para iniciar o servidor *GATT*. Por fim, o método *SendResponse* envia as respostas *mDoc* para o *Peripheral*, de modo a serem enviadas pelo canal *BLE*.

4.3.6 *Resumo do Capítulo*

No Capítulo 4 são abordadas as estruturas *mDoc* e os módulos da aplicação e suas implementações. É também descrita a relação que os módulos têm entre si e de qual a sua função no portador *mDoc*. Inicialmente são descritas as estruturas e dados *mDoc*. É feita uma apresentação da estrutura do documento *mDoc*, dos seus dados, do pedido *mDoc* e da resposta *mDoc*. Depois, é apresentado e explicado o formato de serialização de dados *CBOR*. Ainda dentro deste contexto, é descrito *COSE* que corresponde aos serviços de segurança *CBOR* utilizados nos mecanismos de segurança. De seguida, as implementações de cada módulo são descritas. É feita uma explicação do propósito de cada módulo e depois são demonstradas as suas implementações. Estas implementações são representadas por classes, variáveis e métodos, e as interações que têm entre si. No Capítulo 5 é abordado o fluxo da aplicação. São demonstrados os testes feitos e os seus resultados, bem como as decisões tomadas para a sua realização.

5

PROVA DE CONCEITO

Este capítulo descreve o funcionamento da aplicação e implementações realizadas. São descritas as preparações feitas para a realização do teste funcional, cujo objetivo é demonstrar a viabilidade do portador *mDoc* desenvolvido, e o processo sequencial do seu funcionamento. A implementação do portador *mDoc* encontra-se num repositório público [1].

5.1 PREPARAÇÕES E AMBIENTE DE DESENVOLVIMENTO

Para a realização de testes funcionais foi utilizado o *smartwatch Fossil Gen 6* com *WearOS 3.0*. É neste dispositivo que a aplicação do portador *mDoc* é executada. Como esta é uma implementação *standalone* do portador *mDoc*, o documento *mDoc* é armazenado neste dispositivo, bem como todas as estruturas *mDoc*, e todos os mecanismos de segurança são aqui implementados.

Como referido no Capítulo 3, o portador *mDoc* é desenvolvido em *Xamarin*, no *IDE (Integrated Development Environment) Visual Studio*. Foram criados dois projetos: o projeto com os módulos partilhados entre plataformas *wearable*, *NET_Shared*; o projeto *wearOS* com o *UI* e os módulos da plataforma *wearable WearOS*.

No projeto *NET_Shared* são criadas pastas para cada módulo partilhado e são armazenadas as suas dependências. Estas dependências correspondem às bibliotecas que foram utilizadas para as implementações destes módulos (ver Figura 16a).

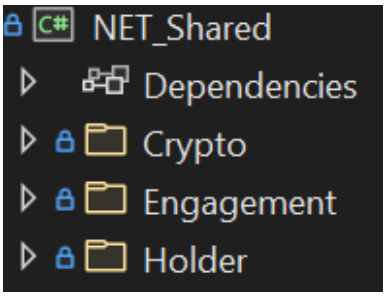
O projeto *wearOS* implementa a *UI* e os módulos *BLE* e *Droid*. Este projeto referencia o projeto *NET_Shared* de modo a poder utilizar as suas implementações. Para além do projeto *NET_Shared*, também são referenciadas bibliotecas auxiliares na implementação dos módulos *BLE* e *Droid*.

O documento *mDoc* é armazenado localmente no portador *mDoc* como um ficheiro *JSON* na pasta de recursos do projeto *wearOS*. É também nesta pasta que são armazenados os *layouts* da *UI*.

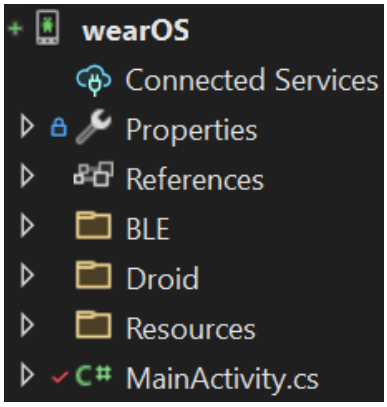
A atividade do portador *mDoc* é controlada no ficheiro *MainActivity*. Este ficheiro é o ponto de acesso da aplicação portador *mDoc*, e é aqui que ocorrem as alterações da *UI*. *MainActivity* é definida como ponto de

1 *mID_wearable* (Matias,2023)

acesso da aplicação portador *mDoc* no ficheiro *AndroidManifest* que se encontra nas propriedades do projeto. Para além disso, *AndroidManifest* tem a versão *Android* do portador *mDoc*, o tipo de *hardware*, e a definição das permissões de *Bluetooth* e localização necessárias para o funcionamento do portador *mDoc*. A Figura 16b apresenta os módulos do projeto *wearOS*.



(a) *NET_Shared*



(b) *wearOS*

5.1.1 *Plataforma watchOS*

O *deployment* para *watchOS* do portador *mDoc* implica estabelecer o canal de conexão *BLE*. As bibliotecas *BLE* utilizadas neste trabalho são específicas à plataforma *WearOS* pelo que é necessário implementar as classes *Peripheral* e *Advertiser*, ou outras com a mesma função, de modo a poder estabelecer um canal *BLE*.

Em *Xamarin*, a implementação portador *mDoc* para *watchOS* tem algumas diferenças de preparação em relação à plataforma *WearOS*. Assim como para a plataforma *WearOS*, o portador *mDoc* para *watchOS* é uma aplicação *standalone*, porém é necessário criar quatro projetos: o projeto que contém o código do portador *mDoc*, *watchOSExtension*; o projeto que contém a *UI* e os recursos do portador *mDoc*, *watchOSWatch*; o projeto que contém o código a ser executado pelo *smartwatch* associado, *watchOSWatch*; o projeto de partilha de código, *NET_Shared*. Como a aplicação é *standalone* não vai ser desenvolvido código no projeto *watchOSPhone*, porém precisa de ser criado para que a aplicação *watchOS* seja funcional.

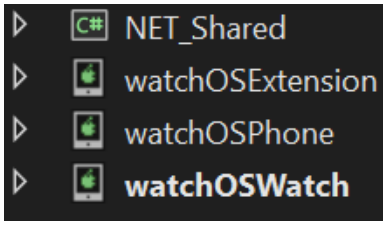


Figura 17: Projetos *watchOS*

No projeto *NET_Shared* estão os módulos partilhados igual à Figura 16a. O projeto *watchOSWatch* tem os módulos específicos à plataforma *watchOS*. A *UI* do portador *mDoc* encontra-se aqui implementada, bem como o ficheiro *Info.plist* que contém permissões *Bluetooth* necessárias para o funcionamento do portador *mDoc*. É também aqui que estão implementados os módulos específicos à plataforma *watchOS*: o módulo que implementa os mecanismos da tecnologia de transmissão de dados, *BLE*; e o módulo de ligação entre os módulos partilhados e específicos à plataforma *watchOS*, *Droid*. Por fim, o projeto *watchOSExtension*, assim como o projeto *watchOSPhone*, não tem código desenvolvido e apenas referencia o projeto *NET_Shared*. Esta referencia permite que o código partilhado seja utilizado como código do portador *mDoc* para *watchOS*.

5.2 LEITOR MDOC PARA TESTE

Para efeitos de teste, é necessária a implementação de um leitor *mDoc* para verificar a interação portador-leitor. Tendo isto em conta, é implementado um leitor *mDoc* com funcionalidades que permitam testar as implementações do portador *mDoc*. O leitor *mDoc* é implementado para *Android 11* e executado num *smartphone Samsung S20 FE*. As seguintes funcionalidades estão presentes no leitor *mDoc* desenvolvido:

- Leitura de *QR code*;
- Criação de um pedido *mDoc*;
- Autenticação do portador *mDoc*;
- Cliente central *BLE*;

O objetivo do leitor *mDoc* é testar as funcionalidades do portador *mDoc*, interagindo com ele. Assim, não são implementados todos os requisitos para um leitor *mDoc* apresentados na norma técnica *ISO/IEC DIS 18013-5*. Apesar das funcionalidades do leitor *mDoc* estarem desenvolvidas de acordo com norma técnica *ISO/IEC DIS 18013-5*, não é necessária a implementação de todos os requisitos nela descritos tendo em conta o objetivo do leitor *mDoc*. Por exemplo, a norma técnica *ISO/IEC DIS 18013-5* requer que o leitor *mDoc* implemente todas as tecnologias de conexão de dados e algoritmos criptográficos de assinatura, porém para testar as funcionalidades do portador *mDoc*, o leitor *mDoc* só precisa de implementar um de cada.

O leitor *mDoc* tem a capacidade de ler o *QR code* gerado pelo portador *mDoc*, criar um pedido *mDoc* conforme a Secçao 4.1 e autenticar o portador *mDoc* segundo a Secção 4.3.1. Por fim, o leitor *mDoc* implementa o cliente central *BLE* que permite o *scan* dos pacotes *BLE* enviados pelo portador *mDoc*, de modo que o leitor *mDoc* se possa conectar com o *BLE server socket* deste último.

5.3 FLUXO DE TRABALHO DO PORTADOR *mdoc*

O portador *mDoc* é inicializado e é gerada uma instância da classe *Droid* (ver Secção 4.3.5). *Droid* tem na sua constituição uma instância da classe *Peripheral* e outra da classe *Engagement* (*i.e.,* estrutura de conexão de dispositivos). A estrutura de conexão de dispositivos é formatada em *CBOR bytes* e posteriormente codificada numa *string Base64Url*. É criado um *URI (Universal Resource Identifier)* cujo *scheme* é a *string mdoc:* concatenada com a estrutura de conexão de dados em *Base64Url*. Este *URI* permite ao leitor *mDoc* verificar se os dados enviados contém informação *mDoc*. De seguida, é gerado o *QR Code* que representa esta estrutura. A Figura 18 ilustra o leitor *mDoc* a fazer um *scan* do *QR Code* gerado de modo a obter a estrutura de conexão entre dispositivos, e apresenta a *string* que originou o *QR Code*.

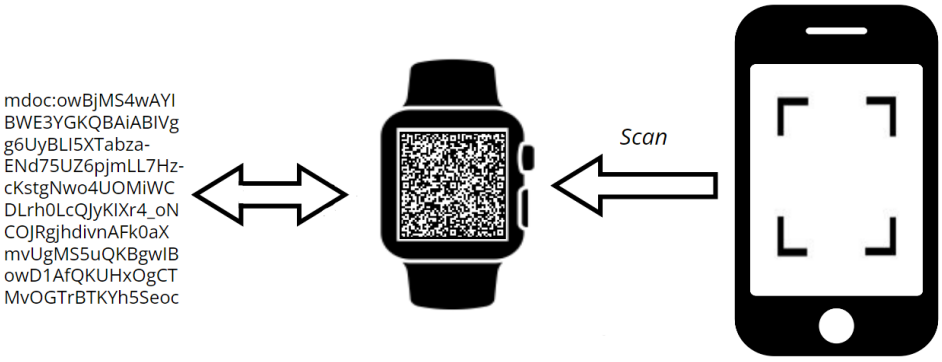


Figura 18: *Scan* do *QR code*

O leitor *mDoc* recebe nesta estrutura a chave efémera pública do portador *mDoc*, que lhe permite gerar as chaves de sessão do portador *mDoc* e do leitor *mDoc*.

De seguida, o *Peripheral* do portador *mDoc* abre o servidor *GATT(Generic Attribute Profile)* e regista o serviço *BLE* que representa o modo de servidor periférico *BLE*. Posteriormente, é inicializada a *AcceptThread* que cria o *server socket BLE* e atribui o valor do seu *PSM (Protocol Service Multiplexor)* à característica do serviço *BLE*. A Figura 19 representa o servidor *GATT* depois de lhe ser atribuído o valor do *PSM* ao seu serviço *BLE*.

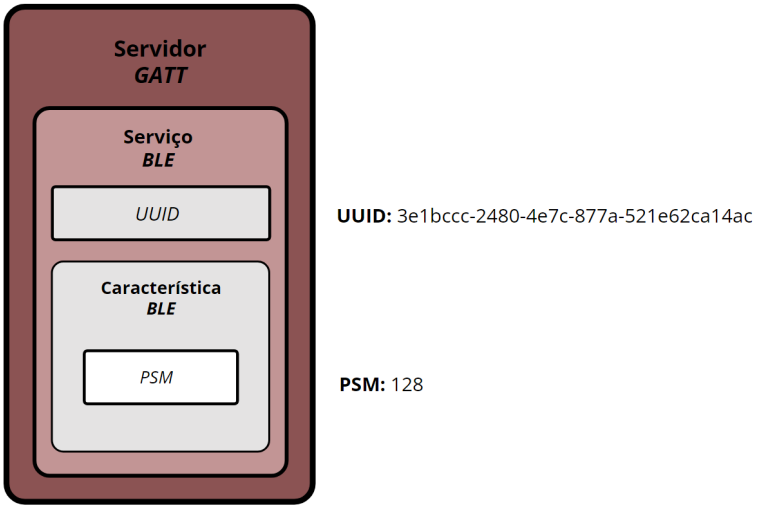


Figura 19: Servidor *GATT*

Enquanto o *server socket BLE* aguarda por uma conexão *BLE*, é iniciado o processo de *advertising*. O *Advertiser* do *Peripheral* cria pacotes *BLE* cujo conteúdo é o *UUID* do serviço *BLE* com o valor da sua característica (*i.e., PSM*). O portador *mDoc* transmite os pacotes *BLE* que são captados pelo leitor *mDoc*. O leitor *mDoc* compara o *UUID* dos pacotes *BLE* com o *UUID* que se encontra na estrutura de conexão entre dispositivos. Caso os *UUID's* coincidam, o leitor *mDoc* verifica o valor da característica do serviço *BLE*, e adquire o valor do *PSM* para se conectar ao *server socket BLE* do portador *mDoc*. A Figura 20 demonstra o processo de *advertising*.

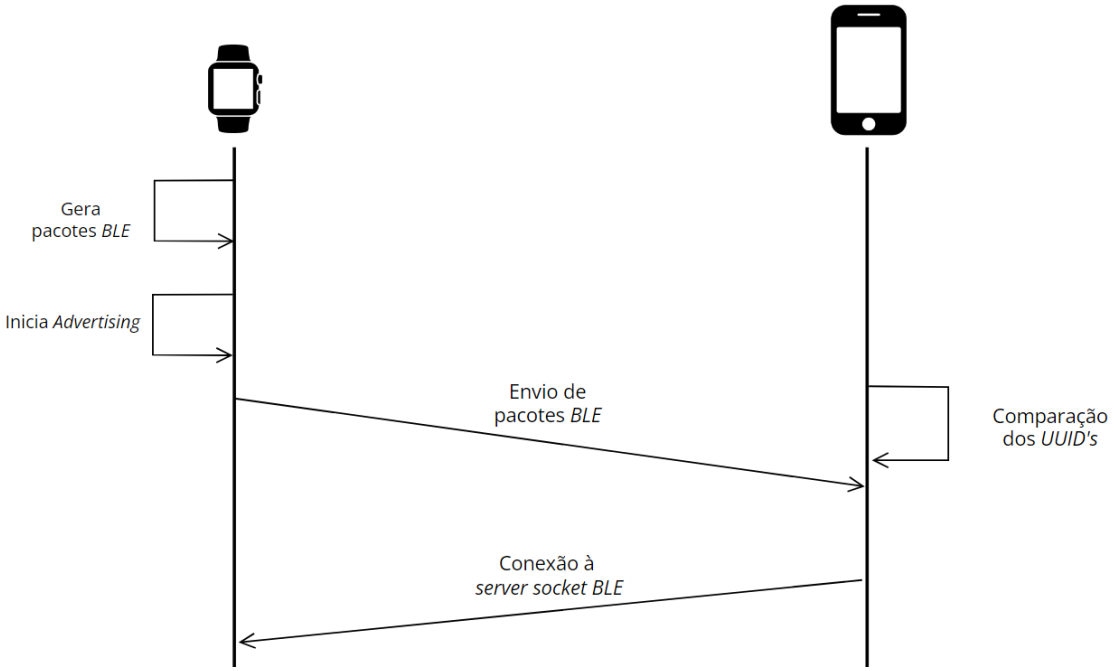


Figura 20: *Advertising*

Com a conexão do leitor *mDoc* ao *server socket BLE* do portador *mDoc*, está criado o canal *BLE* a ser usado para o envio de mensagens *mDoc*. A *ConnectedThread* gere a comunicação *BLE* do lado do portador *BLE* e garante que este fica à espera do pedido *mDoc* no mínimo 300 segundos (Iso,2021). A interação portador-leitor no canal *BLE* é ilustrada no diagrama da Figura 21 e os processos representados são descritos adiante.

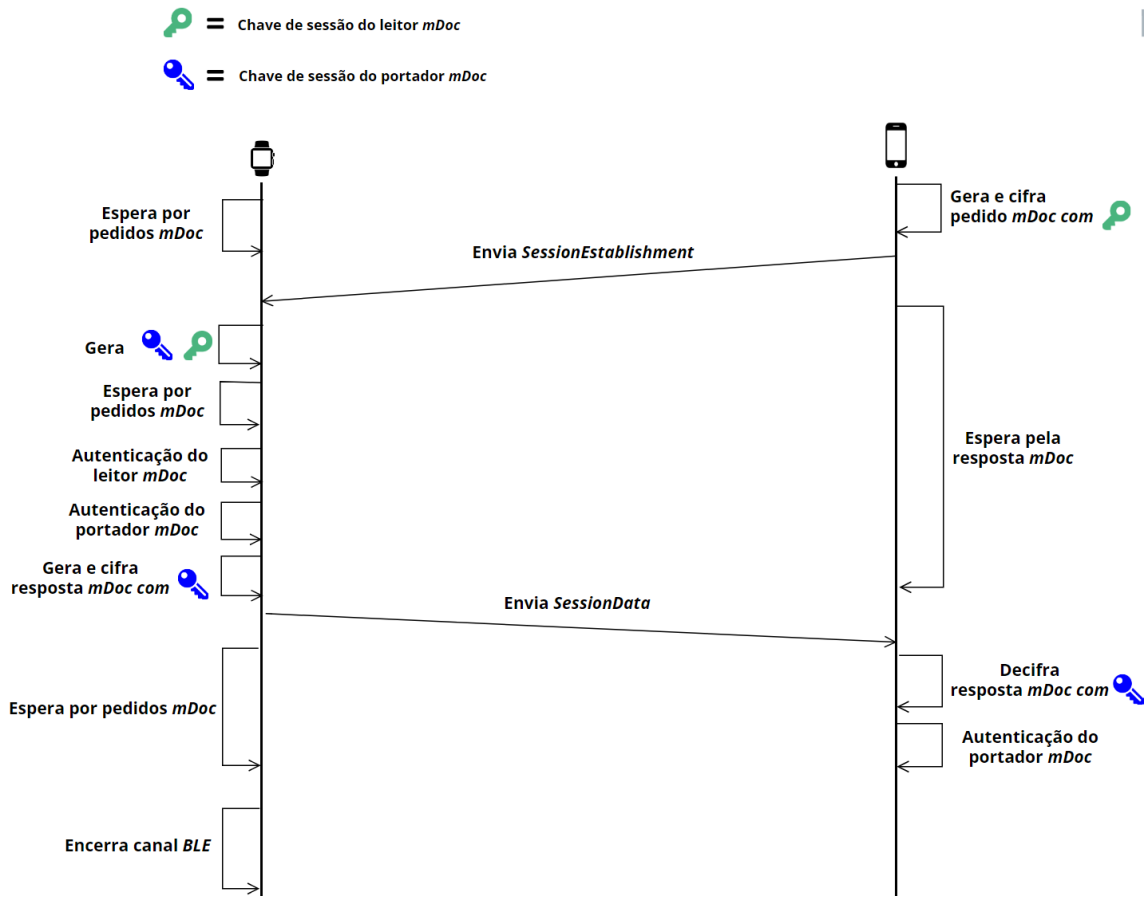


Figura 21: Interação portador-leitor no canal *BLE*

O leitor cria e formata o pedido *mDoc* em *CBOR bytes*, e cifra-o com a sua chave de sessão (*i.e., SKReaderKey*). O pedido é introduzido na estrutura *SessionEstablishment* juntamente com a chave efémera do leitor *mDoc*. De seguida, codifica a estrutura *SessionEstablishment* e envia-a para o portador *mDoc* pelo canal *BLE*. Ao receber a estrutura *SessionEstablishment*, o portador *mDoc* utiliza a chave efémera pública do leitor *mDoc* para gerar a sua chave de sessão e a do leitor *mDoc* (ver Secção 4.3.1). O portador *mDoc* decifra o pedido *mDoc* com a chave de sessão do leitor *mDoc* para ter acesso à estrutura do pedido *mDoc*. Os dados solicitados são apresentados ao utilizador do portador *mDoc* que deve autorizar o seu envio para que a resposta *mDoc* seja enviada e gerada. A Figura 22 ilustra como os dados são apresentados no *smartwatch*.

Figura 22: Dados *mDoc* no *smartwatch*

Ao selecionar a opção *ENVIAR* o utilizador autoriza o envio dos dados apresentados no ecrã e dá início à geração da resposta *mDoc*. Antes de gerar a resposta *mDoc*, o portador *mDoc* precisa de autenticar o leitor *mDoc* (ver Secção 4.3.1). O portador *mDoc* começa por extrair a chave pública de autenticação do leitor *mDoc* do certificado que se encontra na estrutura *ReaderAuth*. De seguida, é gerada a estrutura *ReaderAuthentication* para a verificação da assinatura do leitor *mDoc*. A chave de pública de autenticação do leitor *mDoc* e a estrutura *ReaderAuthentication* são utilizadas para verificar a assinatura do leitor *mDoc*. Para a extração do certificado e para a verificação da assinatura são utilizados os métodos da classe *SignMessage*.

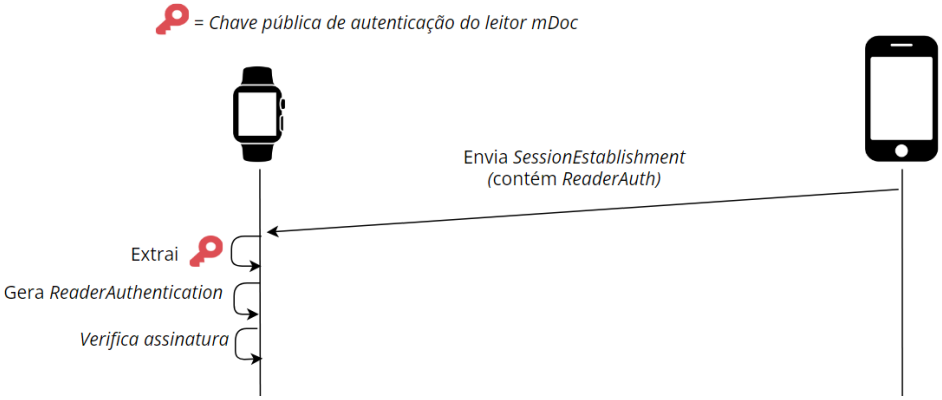


Figura 23: Autenticação do leitor *mDoc*

Caso a autenticação do leitor *mDoc* não se verifique, o pedido *mDoc* é ignorado e o portador *mDoc* fica à espera de receber mais pedidos *mDoc*. Caso contrário, é gerada a resposta *mDoc*. Utilizando a classe *Retrieval*, o portador *mDoc* verifica quais os dados solicitados no pedido *mDoc* e retira o seu valor do documento *mDoc* para gerar a estrutura *DeviceSigned*. Esta estrutura contém os dados *mDoc* solicitados e a estrutura *DeviceAuth*, utilizada para autenticar o portador *mDoc*. É gerada a estrutura *DeviceAuthentication* sobre a qual a chave de autenticação privada do portador *mDoc* vai criar a assinatura. Depois de assinada a estrutura *DeviceAuthentication*, é gerada a estrutura *DeviceAuth* contendo a assinatura. O processo de assinar dados e geração da estrutura *DeviceAuth* é feito utilizando os métodos da classe *SignMessage*.

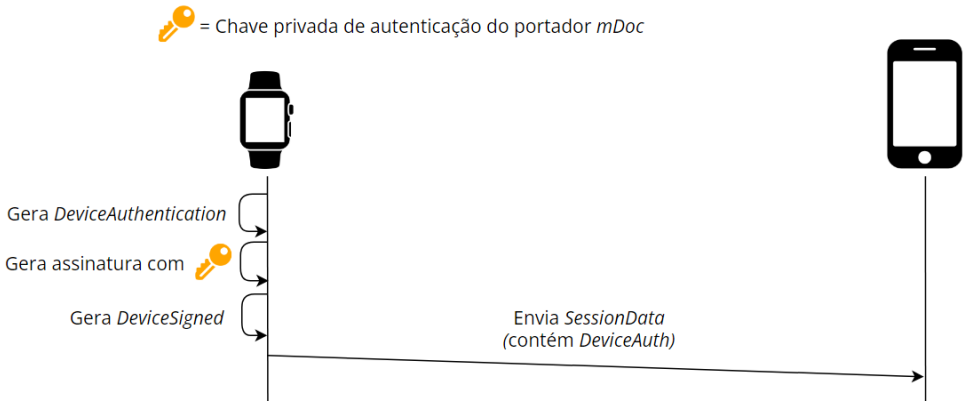


Figura 24: Autenticação do portador *mDoc*

Para além da estrutura *DeviceSigned*, a resposta *mDoc* tem uma outra estrutura de autenticação, *IssuerSigned*. Esta estrutura tem os dados *mDoc* solicitados pelo leitor *mDoc* e a estrutura *IssuerSigned* que contém a *MSO*. *IssuerSigned* permite ao leitor *mDoc* verificar os dados *mDoc* perante a autoridade emissora. Depois de criadas as estruturas *DeviceSigned* e *IssuerSigned*, é gerada a resposta *mDoc* com o método da classe *Retrieval*, *GenerateResponse*. A resposta *mDoc* é codificada para *CBOR bytes* e cifrada com a chave de sessão do portador *mDoc*. Por fim, é enviada a estrutura *SessionData* em formato *CBOR bytes* que contém a resposta *mDoc* cifrada. Depois do leitor *mDoc* receber a resposta *mDoc*, este pode enviar mais pedidos *mDoc* para o portador *mDoc*. O portador *mDoc* fica à espera de receber mais pedidos *mDoc* como ilustrado na Figura 25.

Figura 25: *UI* de espera por pedidos *mdoc*

O canal *BLE* é encerrado automaticamente se não forem enviados dados num período de 300 segundos. O utilizador do portador *mDoc* pode cancelar a espera por pedidos *mDoc* ao selecionar a opção *CANCELAR*. Esta ação encerra a conexão *BLE* e elimina as chaves efémeras e de sessão e o portador *mDoc* volta ao estado de inicialização do *QR Code*, de modo a poder criar novas conexões.

5.3.1 *Resumo do Capítulo*

No Capítulo 5, são abordadas as preparações e o funcionamento da aplicação portador *mDoc*. É descrito o fluxo de trabalho do portador *mDoc* e como as implementações do Capítulo 4 interagem entre si para atingir o objetivo do trabalho. Inicialmente, são descritas as preparações necessárias para o desenvolvimento do portador *mDoc*. É referido o *hardware*, projetos e pastas necessárias para a implementação do portador *mDoc*. Depois é apresentado o fluxo de trabalho do portador *mDoc*. Aqui são descritas as interações portador-leitor, de forma sequencial, tendo em consideração as implementações dos módulos descritos no Capítulo 4. São utilizados diagramas para explicar estas interações e apresentadas as alterações na *UI* que estas causam.

6

CONCLUSÃO

O trabalho realizado tem como base a norma técnica *ISO/IEC DIS 18013-5*, cujo objetivo é delinear os requisitos necessários para a transmissão segura de dados de identificação entre dispositivos. Mais especificamente, a aplicação criada corresponde à implementação destes objetivos por parte do portador do documento *mDL*. Os utilizadores podem ter os seus dados pessoais *mDoc* requisitados por um leitor *mDoc* em formato de pedido *mDoc*, e podem consentir o envio dos dados requisitados em forma de respostas *mDoc*, com a garantia de segurança proporcionada pelo seguimento da norma técnica *ISO/IEC DIS 18013-5*. A aplicação foi desenvolvida em *Xamarin* para *WearOS*, apesar de todo o código exceto a *UI* e a tecnologia de transmissão *BLE* poder ser partilhado entre plataformas *wearable*. *Xamarin* é uma ferramenta que permite a partilha de código entre diferentes plataformas, assim, a maioria do código utilizado para desenvolver a aplicação em *WearOS* pode ser utilizado em outras plataformas *wearable*.

A utilização de um *smartwatch* para a partilha de dados entre o portador e o leitor criou a necessidade de adaptar a norma técnica *ISO/IEC DIS 18013-5*. Enquanto que a implementação das estruturas *mDoc* e dos mecanismos de segurança não foi influenciada, foi necessário ter em consideração a plataforma de desenvolvimento cruzado e as tecnologias de conexão e comunicação a utilizar. As escolhas da utilização de *Xamarin*, *QR Code* e *BLE* para a esta implementação surgiram como consequência da utilização do *smartwatch* enquanto portador *mDoc*. *Xamarin* foi escolhido pois é o que apresentava o maior suporte *wearable* entre as plataformas de desenvolvimento cruzado consideradas; *QR Code* pela sua universalidade de geração e leitura; e *BLE* pelo seu poder de consumo baixo e por ser adequado para enviar dados de tamanho pequeno.

A aplicação implementa as estruturas de dados *mDoc* e os mecanismos de segurança referidos na técnica *ISO/IEC DIS 18013-5*. O pedido *mDoc*, resposta *mDoc* e a estrutura de conexão de dispositivos, bem como todas as suas subestruturas e estruturas das quais fazem parte (*i.e., SessionEstablishment* e *SessionData*), foram implementados com base nos módulos do Capítulo 4. O módulo *Engagement* oferece classes e métodos que permitem gerar um *QR Code* com a informação necessária sobre a estrutura de conexão de dispositivos. O módulo *Holder* permite armazenar as variáveis e subestruturas necessárias na constituição do pedido *mDoc* e da resposta *mDoc*, e apresenta métodos necessários para processar o pedido *mDoc* construir a resposta *mDoc*. Os mecanismos de segurança foram implementados no módulo *Crypto*. Aqui, são criadas classes e métodos que permitem a geração de chaves efémeras, de sessão e de autenticação e outras estruturas criptográficas. As

estruturas criptográficas geradas possibilitam a decifragem dos pedidos *mDoc* e cifragem das respostas *mDoc*, e permitem autenticar o portador *mDoc* e o leitor *mDoc*.

A transmissão de dados *BLE* portador-leitor é feita com a implementação de um servidor periférico *BLE* e um *advertiser*. O módulo *BLE* implementa um *advertiser* que inicia o processo de *advertising* para a partilha de pacotes *BLE*. O *QR code* com o processo de *advertising* fornecem a informação necessária para que o leitor *mDoc* possa iniciar uma conexão *BLE* com o portador *mDoc*. É implementado o servidor periférico que serve como inicializador do *advertiser* e do servidor *GATT*, e aceita e gere conexões *BLE*. Estas duas classes permitem que o portador *mDoc* e o leitor *mDoc* comuniquem entre si, enviando mensagens *mDoc*.

Com estas implementações, o portador *mDoc* está desenvolvido em concordância com os requisitos funcionais e técnicos da norma técnica *ISO/IEC DIS 18013-5*. O portador *mDoc* oferece as estruturas necessárias para enviar dados seletivamente, e para o leitor *mDoc* confirmar a sua autenticidade e a do portador *mDoc*. Os dados e estruturas *mDoc* respeitam o modelo de dados *mDoc* e a comunicação portador-leitor segue os requisitos técnicos da norma técnica *ISO/IEC DIS 18013-5*.

6.1 TRABALHO FUTURO

O principal objetivo deste trabalho foi a implementação da norma técnica *ISO/IEC DIS 18013-5* em *wearable*, recorrendo a plataformas de desenvolvimento cruzado. Tendo em conta a natureza deste trabalho, o passo seguinte é a implementação da tecnologia *Bluetooth Low Energy* e da *UI* com código que pudesse ser partilhado entre várias plataformas *wearable*. No caso do *BLE*, isto implica o desenvolvimento de um *plugin* que permita código que não dependa de plataforma. Para a *UI*, resulta no desenvolvimento de um *plugin* ou então, considerando o aumento de popularidade dos *smartwatches*, a utilização de ferramentas como *Xamarin.Forms* que podem eventualmente suportar estas plataformas.

A implementação de outras tecnologias de transmissão de dados e outros algoritmos criptográficos referenciados na *ISO* também são considerados. No futuro, é ideal a implementação da transmissão de dados utilizando *Wi-Fi Aware* e *NFC*, assim como a implementação de outros algoritmos criptográficos alternativos aos usados nesta aplicação. Estas implementações atribuiriam um grau de adaptabilidade maior a situações diferentes à aplicação.

A interação entre o portador *mDoc* e a autoridade emissora é algo a considerar a implementar. A necessidade do *smartphone* para a obtenção do *mDoc* é eliminada, permitindo ao portador *mDoc* comunicar diretamente com a autoridade emissora para obter o documento.

BIBLIOGRAFIA

[1] Dan Abramov and Rachel Nabors. React. 2013. URL https://react.dev/.

[2] Carsten Bormann and Paul Hoffman. Rfc 7049. 2013. URL https://www.rfc-editor.org/rfc/rfc7049.

[3] BouncyCastle. Bouncycastle, 2000. URL https://www.bouncycastle.org/.

[4] J.L. Camp. Digital identity. *IEEE Technology and Society Magazine*, 23(3):34–41, 2004. doi: 10.1109/MTAS.2004.1337889.

[5] Marta E. Cecchinato, Anna L. Cox, and Jon Bird. Smartwatches: The good, the bad and the ugly? volume 18, pages 2133–2138. Association for Computing Machinery, 4 2015. ISBN 9781450331463. doi: 10.1145/2702613.2732837.

[6] Yi Gao, Siyu Zeng, Ji Zhao, Wenxin Liu, and Wei Dong. Airtext: One-handed text entry in the air for cots smartwatches. *IEEE Transactions on Mobile Computing*, pages 1–1, 2021. ISSN 1536-1233. doi: 10.1109/TMC.2021.3130036. URL https://ieeexplore.ieee.org/document/9625777/.

[7] Google. Flutter. 2017. URL https://flutter.dev/.

[8] Iso. Personal identification-iso-compliant driving licence-part 1: Physical characteristics and basic data set, 2018. URL www.iso.org.

[9] Iso. Personal identification-iso-compliant driving licence-part 2: Machine-readable technologies, 2020. URL www.iso.org.

[10] Iso. Personal identification-iso-compliant driving licence-part 5: Mobile driving licence (mdl) application, 2021. URL www.iso.org.

[11] Hugo Fernandes Matias. Portador_mdoc. 2023. URL https://github.com/HugoFerMatias/mID_wearable.git.

[12] Microsoft. .net. 2002. URL https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet.

[13] Microsoft. Xamarin. 2012. URL https://dotnet.microsoft.com/en-us/apps/xamarin.

[14] Tejaswini Mishra, Meng Wang, Ahmed A. Metwally, Gireesh K. Bogu, Andrew W. Brooks, Amir Bahmani, Arash Alavi, Alessandra Celli, Emily Higgs, Orit Dagan-Rosenfeld, Bethany Fay, Susan Kirkpatrick, Ryan Kellogg, Michelle Gibson, Tao Wang, Erika M. Hunting, Petra Mamic, Ariel B. Ganz, Benjamin Rolnik, Xiao Li, and Michael P. Snyder. Pre-symptomatic detection of covid-19 from smartwatch data. *Nature Biomedical Engineering*, 4:1208–1220, 12 2020. ISSN 2157846X. doi: 10.1038/s41551-020-00640-6.

[15] Peter Occil. Cbor. 2013.

[16] Masa Ogata and Michita Imai. Skin watch: Skin gesture interaction for smart watch. volume 11, pages 21–24. Association for Computing Machinery, 3 2015. ISBN 9781450333498. doi: 10.1145/2735711.2735830.

[17] Herrmann Raffael. Qrcode. 2013. URL https://github.com/codebude/QRCoder.

[18] Jim Schaad. Cose_csharp. 2015. URL https://github.com/cose-wg/COSE-csharp.

[19] Jim Schaad. Rfc 8152, 7 2017.

[20] Clare Sullivan. Digital identity – from emergent legal concept to new reality. 2011.

[21] Ursula von der Leyen. European digital identity. 2020. URL https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/european-digital-identity_en.

[22] Nisha D. Wanjari and Shailaja. C. Patil. Wearable devices. In *2016 IEEE International Conference on Advances in Electronics, Communication and Computer Technology (ICAECCT)*, pages 287–290, 2016. doi: 10.1109/ICAECCT.2016.7942600.

[23] Martin Woolley. Bluetooth ® specification bluetooth core specification, 1999. URL https://www.bluetooth.com/specifications/adopted-specifications.

[24] Francois Yergeau. Rfc 3629, 2003.

[25] Jian Zhang, Hongliang Bi, Yanjiao Chen, Mingyu Wang, Liming Han, and Ligan Cai. Smarthandwriting: Handwritten chinese character recognition with smartwatch. *IEEE Internet of Things Journal*, 7:960–970, 2 2020. ISSN 23274662. doi: 10.1109/JIOT.2019.2947448.