POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Integrazioni NFC nel contesto mobile

Sviluppo di esperienze interattive per un'applicazione iOS

Supervisori

Prof. Luca ARDITO

Marco BALANZINO

Giovanni PRESTIGIACOMO

Candidato

Andrea DELUCA

Aprile 2024

Abstract

La tecnologia Near Field Communication (NFC) ha rivoluzionato il modo in cui interagiamo con il mondo digitale attraverso dispositivi mobili. Consente lo scambio di dati tra dispositivi compatibili, semplicemente avvicinandoli tra loro. Questa sua semplicità d'uso e la sicurezza integrata rendono l'NFC una soluzione ideale per diverse applicazioni: dalle transazioni finanziarie senza contatto ai documenti di viaggio elettronici. Inoltre, la diffusione della tecnologia riflette la sua crescente importanza nel contesto tecnologico attuale e pone le basi per nuovi sviluppi e innovazioni future.

Questa tesi si pone come obiettivo principale quello di sviluppare e fornire nuove esperienze interattive per un'applicazione iOS mediante la tecnologia NFC. Per soddisfare tale requisito si sono principalmente analizzati due diversi casi d'uso, il cui studio suddivide la tesi in due parti. Inizialmente sono state condotte delle analisi sull'utilizzo di un iPhone come terminale POS. Successivamente, si è analizzata la possibilità di integrare la tecnologia NFC all'interno delle applicazioni di mobile banking, combinandola con le specifiche tecniche delle carte d'identità elettroniche conformi alle raccomandazioni internazionali di ICAO. Ciò ha infine fornito gli strumenti necessari per la realizzazione di una libreria software a supporto di tale integrazione all'interno di applicazioni iOS.

Indice

Elenco delle tabelle IV

Elenco delle figure V

Elenco dei codici VI

Acronimi VIII

1 Introduzione 1
- 1.1 Stato dell'arte . . . 2
- 1.2 Obiettivi . . . 3
- 1.3 Struttura della tesi . . . 3

2 Near Field Communication 5
- 2.1 Cenni storici . . . 6
- 2.2 Motivazione ed evoluzione tecnologica . . . 6
  - 2.2.1 Codici a barre . . . 7
  - 2.2.2 Carta a banda magnetica . . . 7
  - 2.2.3 Smart card . . . 8
  - 2.2.4 RFID . . . 13
- 2.3 La comunicazione di prossimità . . . 16
  - 2.3.1 Componenti di un sistema NFC . . . 16
  - 2.3.2 Modalità di funzionamento . . . 17

3 Transazioni digitali 19
- 3.1 Sistema POS . . . 20
  - 3.1.1 Tipologie di terminali . . . 20
- 3.2 Apple Pay . . . 21
  - 3.2.1 Memorizzazione del metodo di pagamento . . . 22
  - 3.2.2 Transazione POS mediante Apple Pay . . . 24

4 Uso di iPhone come POS 25
- 4.1 Panoramica dell'applicazione demo . . . 25
  - 4.1.1 Schermata principale . . . 25

4.1.2 Inserire una nuova transazione . . . 27
4.1.3 Inserire un nuovo metodo di pagamento . . . 27
4.2 API di sistema per l'utilizzo della NFC . . . 29
4.2.1 CoreNFC . . . 30
4.2.2 ProximityReader . . . 33
4.3 Risultati ottenuti . . . 34

5 Documenti d'identità elettronici 36
5.1 In Italia . . . 36
5.1.1 Cenni storici . . . 37
5.1.2 Layout della CIE . . . 37
5.1.3 La CIE come smart card . . . 39
5.2 ICAO Doc 9303 . . . 39
5.2.1 Introduzione ai MRTD . . . 40
5.2.2 Integrazione della tecnologia *contactless* . . . 41

6 Sviluppo del *framework* per la lettura NFC di un MRTD 42
6.1 Codifica dei dati . . . 42
6.1.1 Concetti essenziali di ASN.1 . . . 43
6.1.2 Codificare dati ASN.1 mediante BER . . . 44
6.1.3 Sottoinsiemi di BER . . . 47
6.1.4 Sviluppo del modulo ASN.1 . . . 48
6.2 Comunicazione con l'IC . . . 51
6.2.1 Inizializzazione della sessione NFC . . . 52
6.2.2 Comandi APDU . . . 54
6.3 Lettura e decodifica dei dati . . . 57
6.3.1 Struttura dei dati in LDS1 . . . 58
6.3.2 Implementazione del processo di decodifica . . . 60
6.4 Meccanismi di sicurezza . . . 64
6.4.1 Accesso all'IC . . . 65
6.4.2 Sessione sicura . . . 71
6.4.3 Autenticazione dei dati . . . 72
6.4.4 Autenticazione dell'IC . . . 73

7 Nuove esperienze interattive 76
7.1 Processo di autenticazione dell'utente . . . 76
7.2 Conferma delle operazioni bancarie . . . 78
7.3 Visualizzazione delle informazioni recuperate . . . 80

8 Conclusioni 81

Bibliografia 84

Elenco delle tabelle

2.1 Struttura di un comando APDU 11
2.2 Struttura di una risposta APDU 11

4.1 Specifiche per le carte di pagamento accettate in *app* 29

Elenco delle figure

3.1 Ultime quattro cifre del PAN e del DAN su Apple Wallet 23

4.1 *Splashscreen* ed *homescreen* della applicazione demo 26
4.2 Operazioni possibili nella *homescreen* 26
4.3 Flusso di inserimento di una nuova transazione 27
4.4 Flusso di inserimento di un nuovo metodo di pagamento 28
4.5 Richiesta di pagamento *contactless* con *Tap to Pay on iPhone* . . . 35

5.1 Fronte della CIE in uso dal 29 settembre 2022 38
5.2 Retro della CIE in uso dal 29 settembre 2022 38

6.1 Struttura del formato TLV 44
6.2 Struttura della codifica BER del *tag* 45
6.3 Rappresentazione di un *tag* BER codificato in 1 byte 46
6.4 Rappresentazione di un *tag* BER codificato in 2 byte 46
6.5 Rappresentazione di un *tag* BER codificato in n byte 46
6.6 Codifica BER della lunghezza in *short form* 47
6.7 Codifica BER della lunghezza in *long form* 47
6.8 Codifica BER della lunghezza in *indefinite form* 47
6.9 Struttura dei file in un eMRTD con LDS1 59

7.1 Flusso di autenticazione del cliente 77
7.2 Estensione del flusso di autenticazione tramite NFC 77
7.3 Richiesta di un nuovo bonifico bancario 78
7.4 Invio di un bonifico bancario (non delicato) 79
7.5 Invio di un bonifico bancario (delicato) con verifica tramite NFC . . 79
7.6 Visualizzazione dei dati recuperati dalla CIE 80

Elenco dei codici

4.1 Inizializzazione di una sessione di lettura NFC con CoreNFC 30
4.2 Estensione al protocollo `NFCNDEFReaderSessionDelegate` 31
4.3 Esempio di metodo per gestire la lettura di un messaggio 32
4.4 Esempio di metodo per gestire una sessione invalidata 32
4.5 Lettura di una carta di pagamento con ProximityReader 33

6.1 Esempio di strutture dati definite in ASN.1 43
6.2 Implementazione della struttura `ASN1Node` 48
6.3 Implementazione del tipo enumerazione `ASN1Node.Content` 49
6.4 Implementazione della classe `ASN1Tag` 49
6.5 Tipi enumerazione di supporto `ASN1TagClass` e `ASN1TagForm` 50
6.6 Metodo `parse(_:)` di `ASN1Parser` 51
6.7 Metodo `readPassport(mrzKey:)` di `NFCPassportReader` 52
6.8 Metodo `tagReaderSession(_:didDetect:)` di `NFCPassportReader` 53
6.9 Metodo `send(cmd:)` di `TagReader` 54
6.10 Metodo `selectPassportApplication()` di `TagReader` 55
6.11 Esempio di comando APDU definito in `APDUCommand` 56
6.12 Implementazione della struttura `APDUResponse` 56
6.13 Metodo `startReading(tagReader:)` di `NFCPassportReader` 60
6.14 Metodo `readDataGroups(tagReader:)` di `NFCPassportReader` . . 60
6.15 Metodo `readDataGroup(tagReader:dgTag:)` di `NFCPassportReader` 61
6.16 Metodo `readDataGroup(tagReader:dgTag:)` di `TagReader` 61
6.17 Metodo `readFile()` di `TagReader` 62
6.18 Implementazione della classe `DGDecoder` 63
6.19 Implementazione della classe `DataGroup` 64
6.20 Implementazione della classe `DataGroup1` 64
6.21 Implementazione della classe `DocumentBasicAccessKeys` 66
6.22 Implementazione della classe `BACHandler` 67
6.23 Implementazione della classe `PACEHandler` 70
6.24 Implementazione della classe `NFCSecureSession` 71
6.25 Implementazione della classe `SecureMessaging` 72

6.26 Implementazione della classe `PassiveAuthenticationHandler` . . . 73
6.27 Implementazione della classe `ChipAuthenticationHandler` 74

Acronimi

3DES Triple DES

AES Advanced Encryption Standard

AID Application Identifier

APDU Application Protocol Data Unit

API Application Programming Interface

ASN.1 Abstract Syntax Notation One

BAC Basic Access Control

BER Basic Encoding Rules

BIN Bank Identification Number

CA Chip Authentication

CAN Card Access Number

CER Canonical Encoding Rules

CIE Carta d'Identità Elettronica

DAN Device Account Number

DER Distinguished Encoding Rules

DES Data Encryption Standard

DG Data Group

DH Diffie-Hellman

ECDH Elliptic Curve Diffie-Hellman

EF Elementary File

eMRTD Electronic Machine Readable Travel Document

ETD Emergency Travel Document

GM General Mapping

IC Integrated Circuit

ICAO International Civil Aviation Organization

IFD InterFace Device

IIN Iusser Identification Number

KDF Key Derivation Function

LDS Logical Data Structure

MAC Message Authentication Code

MF Master File

MROTD Machine Readable Official Travel Document

MRP Machine Readable Passport

MRTD Machine Readable Travel Document

MRV Machine Readable Visa

MRZ Machine Readable Zone

NDEF NFC Data Exchange Format

NFC Near Field Communication

OCR Optical Character Recognition

PA Passive Authentication

PACE Password Authenticated Connection Establishment

PAN Primary Account Number

POS Point of Sale

RF Radio Frequency

RFID Radio Frequency Identification

SE Secure Element

SOD Security Object Document

TD Travel Document

TD1 Size 1 Machine Readable Official Travel Document

TD2 Size 2 Machine Readable Official Travel Document

TD3 Size 3 Machine Readable Travel Document

TLV Type Length Value

TSP Token Service Provider

VIZ Visual Inspection Zone

Capitolo 1

Introduzione

Per quanto qualche anno fa potesse risultare futuristico, se non fantascientifico, poter effettuare pagamenti elettronici senza utilizzare la propria carta di credito, oggi non è più inusuale avvicinare uno *smartphone*, oppure uno *smartwatch*, a un terminale POS per saldare il conto al ristorante, acquistare un abito nuovo oppure un biglietto per i mezzi pubblici. Per di più, quest'ultimo esempio, in alcuni Stati del mondo, potrebbe non rispecchiare la realtà in quanto potrebbe essere possibile esibire, ad esempio, ai tornelli della metropolitana il proprio abbonamento urbano avvicinando ad uno specifico terminale, ancora una volta, il nostro dispositivo mobile. Nel 2007, durante la presentazione del primo iPhone della storia, Steve Jobs esclamò al grande pubblico *"It works like magic"*, ma la sola magia che ruota attorno ai risultati futuristici ottenuti nel corso degli anni si chiama, in realtà, innovazione tecnologica ed ingegneristica. Insomma, tutti gli esempi citati in precedenza condividono una caratteristica fondamentale: l'utilizzo della tecnologia *Near Field Communication* (in italiano comunicazione di prossimità, abbreviato NFC).

Emersa durante gli scorsi decenni, la NFC è una tecnologia di ricetrasmissione che fornisce connettività bidirezionale, a corto raggio, a bassa banda e *wireless*. Ad oggi rappresenta sicuramente una delle applicazioni più diffuse del modello dell'*ubiquitous computing*[1] che, opponendosi al paradigma desktop, si pone come obiettivo un'interazione uomo-macchina in cui l'elaborazione delle informazioni viene interamente integrata all'interno di oggetti e attività di tutti i giorni, andando ad azionare diversi sistemi di calcolo simultaneamente, senza necessariamente essere coscienti del fatto che questi macchinari stiano compiendo le proprie operazioni [1].

[1]Il termine fu coniato da Mark Weiser, informatico statunitense, intorno al 1988 presso il Palo Alto Research Center della Xerox, azienda produttrice di stampanti e fotocopiatrici. Weiser definì gran parte della disciplina evidenziandone i principali interessi e i dubbi a riguardo.

1.1 Stato dell'arte

La tecnologia NFC ha rivoluzionato l'interazione con il mondo digitale attraverso dispositivi mobili, permettendo lo scambio di informazioni tra dispositivi compatibili semplicemente avvicinandoli tra loro. Le sue applicazioni sono diverse e in continua crescita. Gli esempi riportati in precedenza fanno sicuramente parte delle implementazioni più affermate.

- Pagamenti mobili: probabilmente, ad oggi, rappresenta l'applicazione di maggiore successo e più diffusa, consentendo transazioni senza contanti tramite *smartphone* e dispositivi indossabili. Sistemi come Apple Pay e Google Pay integrano l'utilizzo della NFC per fornire agli utenti la possibilità di effettuare pagamenti sicuri.

- Trasporto pubblico: sempre più città in tutto il mondo stanno adottando la tecnologia NFC per i sistemi di bigliettazione elettronica dei trasporti pubblici, consentendo agli utenti di accedere ai mezzi di trasporto utilizzando il proprio *smartphone* o una *smart card contactless*.

- Condivisione di contenuti: la comunicazione di prossimità può semplificare la condivisione di file, foto, video e altro tra dispositivi compatibili, consentendo rapidamente lo scambio di dati tra due utenti.

Malgrado i numerosi vantaggi, è necessario tenere in considerazione anche le sfide e le preoccupazioni che ruotano attorno alla tecnologia. Prima di tutto è importante garantire determinati livelli di sicurezza, soprattutto nei contesti più delicati come nelle transazioni senza contatto e nella condivisione di informazioni personali o biometriche. Ad esempio, in questi casi potrebbe essere necessario implementare un corretto meccanismo di protezione degli accessi o di autenticazione dei dati. Inoltre, sebbene la tecnologia NFC sia ampiamente adottata in diversi settori, potrebbe risultare complesso gestire la varietà di standard e la loro interoperabilità, in particolare nei contesti in cui è necessaria la compatibilità tra dispositivi di diversi produttori.

Nonostante tutto, comunque, il futuro della tecnologia NFC appare promettente e si prevedono ulteriori sviluppi sia per quanto riguarda la sicurezza sia per la sua adozione. Attualmente, è in atto una corposa ricerca di nuove e creative applicazioni della NFC per migliorare l'esperienza utente in vari contesti e si pensa che questa tecnologia possa rappresentare un tassello fondamentale anche nell'*Internet of Things*, consentendo interazioni tra dispositivi domestici intelligenti in maniera più efficiente e conveniente.

1.2 Obiettivi

La tecnologia NFC ha già trasformato molti aspetti delle nostre vite quotidiane e continuerà a farlo nel prossimo futuro, continuando ad essere una forza trainante nell'evoluzione della tecnologia *wireless* e delle interazioni digitali. Pertanto, sulla base delle considerazioni esposte, nasce questa tesi svolta nel contesto aziendale di Iriscube Reply, società del gruppo Reply focalizzata nella consulenza IT e sviluppo di soluzioni innovative *web* e *mobile* nel mondo del *Finance Sector*. Questa ricerca si pone come obiettivo principale quello di sviluppare e fornire nuove esperienze interattive per un'applicazione iOS mediante la tecnologia NFC. Per il raggiungimento di questo requisito, si sono principalmente analizzati due diversi casi d'uso in cui poter applicare la comunicazione di prossimità.

- Inizialmente, sono state condotte delle analisi sull'utilizzo di un iPhone come terminale POS per poter richiedere un pagamento digitale direttamente attraverso il proprio *smartphone* e senza l'ausilio di dispositivi terzi.

- Successivamente, si è analizzata la possibilità di integrare l'utilizzo della tecnologia NFC all'interno di un'applicazione di *mobile banking*, combinandola con le specifiche tecniche delle carte d'identità elettroniche conformi alle raccomandazioni internazionali di ICAO. Questo studio ha fornito gli strumenti necessari per la realizzazione di un *framework* in grado di abilitare la lettura NFC dei documenti d'identità e attraverso il quale è stato possibile implementare nuove funzionalità all'interno di un'applicazione iOS realizzata appositamente per questa tesi.

1.3 Struttura della tesi

Gli argomenti trattati per il raggiungimento degli obiettivi della tesi saranno suddivisi nei successivi capitoli secondo la seguente struttura:

- nel capitolo 2 verrà descritta l'evoluzione tecnologia che ha portato alla definizione della tecnologia di nostro interesse, descrivendo i concetti fondamentali sui si basano soprattutto le *smart card* e l'identificazione a radiofrequenza, fino ad arrivare alla NFC;

- nel capitolo 3 si introdurrà il sistema di pagamento mediante terminale POS e si descriveranno i processi che regolano una transazione digitale mediante Apple Pay, nonché la memorizzazione di un metodo di pagamento su Apple Wallet;

- nel capitolo 4, dopo aver brevemente introdotto la struttura dell'applicazione demo che si vuole estendere, si analizzeranno i *framework* offerti agli sviluppatori da Apple per integrare l'utilizzo della NFC nelle applicazione iOS e, infine, si riporteranno i risultati ottenuti sull'utilizzo di un iPhone come terminale POS;

- nel capitolo 5 si introdurranno i documenti d'identità elettronici, in particolare la variante italiana, e il concetto di MRTD, oltre le tecnologie, gli standard e le raccomandazioni internazionali ICAO di riferimento;

- nel capitolo 6 verrà descritta l'implementazione del *framework* sviluppato per il recupero mediante NFC delle informazioni memorizzate in un eMRTD.

- nel capitolo 7, attraverso l'integrazione del *framework* all'interno dell'applicazione iOS dimostrativa, verranno mostrati i risultati in merito allo sviluppo di nuove esperienze interattive.

Come si evince dalla struttura della tesi, non si esporranno contenuti teorici su Swift, ossia il linguaggio di programmazione utilizzato per lo sviluppo di applicazioni iOS native, nonché per la realizzazione del *framework*. Per ulteriori informazioni sul linguaggio si rimanda alla documentazione ufficiale (https://www.swift.org/documentation/).

Inoltre, nel capitolo 6 non verrà presentato il codice sorgente completo ma porzioni di esso, in quanto si è deciso di concentrarsi sull'esposizione dei meccanismi e delle logiche che regolano il funzionamento del *framework*, piuttosto che sull'effettivo codice scritto. Per consultare il codice sorgente completo si rimanda al *repository* del progetto pubblicato su GitHub (https://github.com/andreadeluca/NFCPassportReader).

Capitolo 2

Near Field Communication

La storia della tecnologia NFC ha idealmente inizio nel 1983, quando viene concesso a Charles Walton di depositare il primo brevetto[1] in cui figura la sigla RFID (*Radio Frequency Identification*, in italiano identificazione a radiofrequenza). Difatti, la NFC è una tecnologia di comunicazione che permette a due dispositivi di connettersi, scambiarsi informazioni o attivare funzioni nel momento in cui si trovano ad una distanza ridotta l'uno dall'altro. Essa si basa sul riconoscimento reciproco e rapido tra i due dispositivi e deriva dalla struttura della tecnologia RFID [2]: NFC è un'estensione, o un sottoinsieme, di RFID che, in aggiunta, sfrutta le interfacce tecnologiche delle *smart card* [1]. La RFID rientra nella famiglia tecnologica dei sistemi di identificazione automatica in cui vengono collocati gli strumenti che consentono di fornire informazioni in modo automatizzato e che negli ultimi anni si sono diffusi nei settori più disparati: dalla logistica e *supply chain* alla localizzazione spaziale, espandendosi anche nel contesto del tracciamento dei movimenti e dell'*e-government*[2] [3].

In questo capitolo, dopo aver brevemente ripercorso alcuni degli eventi più importanti che hanno segnato la storia della NFC, per comprendere il contesto e l'evoluzione tecnologica in cui essa si inserisce, nonché il suo funzionamento, verranno innanzitutto introdotti i sistemi precursori: i codici a barre, la RFID, le carte a banda magnetica e le *smart card*. Successivamente, verrà descritta la NFC, mostrando come essa sia effettivamente basata su RFID e sulle interfacce tecnologiche delle *smart card*.

---

[1]Per approfondimenti si veda il brevetto USA 4.384.288 *Portable radio frequency emitting identifier*, concesso a Charles Walton, pubblicato il 17 luglio 1983, consultabile online all'indirizzo https://patents.google.com/patent/US4384288A.

[2]Per *e-government*, o Amministrazione digitale, ma a volte anche *e-gov* oppure governo elettronico, si intende il sistema di gestione digitalizzata della pubblica amministrazione.

2.1 Cenni storici

Nel 2002, fu annunciata l'intenzione da parte di Sony e Philips di cooperare alla creazione di un nuovo standard per la comunicazione tramite radiofrequenza e il 5 settembre del 2002 nasce ufficialmente la tecnologia chiamata *Near Field Communication*. Solo un anno dopo, nel 2003, l'organizzazione internazionale ISO approva la NFC come standard. Nel corso degli anni anche Nokia mostrò interesse nei confronti della tecnologia e nel 2004 fu fondato l'NFC Forum, ossia un'associazione di aziende senza scopo di lucro che promuove l'implementazione e la standardizzazione della tecnologia per garantire l'interoperabilità tra dispositivi e servizi. Nel 2006, vengono così pubblicate le prime specifiche per i *tag* NFC. Nello stesso anno, Nokia si mette concretamente in gioco presentando il primo telefono della storia con tecnologia NFC integrata: il Nokia 6131 NFC. All'inizio del 2009, viene rilasciato lo standard *peer-to-peer* ed, entrati nel nuovo decennio, Samsung presenta il primo *smartphone* Android con NFC: Samsung Nexus S.

Nel 2011, con la presentazione *How to NFC*, tenutasi durante la conferenza annuale Google I/O, vengono mostrate diverse potenzialità della tecnologia, aprendo di fatto il periodo di maggiore intensità per lo sviluppo di applicazioni concrete che sfruttano il nuovo sistema di comunicazione. Nel corso dei successivi anni vengono avviati diversi progetti pilota, tra cui quello tra Endered Italia, Politecnico di Milano e RIM per il buono pasto mobile. Nel 2013, Samsung e Visa collaborano sullo sviluppo di un sistema per i pagamenti digitali, mentre nel 2015 Apple fa il suo debutto nell'uso della comunicazione di prossimità, integrando un *chip* NFC all'interno di iPhone 6, per avviare successivamente Apple Pay, ossia il suo sistema per le transazioni digitali.

2.2 Motivazione ed evoluzione tecnologica

La principale motivazione che ha consentito di investire sulla tecnologia NFC nasce non solo dalla possibilità di una facile e immediata interazione tra dispositivi, ma anche dall'idea di integrare informazioni private e personali, come quelle memorizzate nelle carte di credito o di debito oppure nei documenti d'identità elettronici, all'interno di dispositivi mobili, permettendone un utilizzo rapido ed intuitivo. Inoltre, una comunicazione *wireless*, a differenza di quella cablata, ha il vantaggio di includere la mobilità nella sua definizione. Questa proprietà, in generale, consente a chi utilizza una comunicazione senza fili di essere più flessibile, aumentando la propria produttività [1].

2.2.1 Codici a barre

La tecnologia dei codici a barre, sebbene possa sembrare distante dalla comunicazione di prossimità, costituisce la prima modalità di identificazione a distanza, fungendo da precorritrice della tecnologia RFID. Un codice a barre (in inglese *barcode*) è una rappresentazione visuale dei dati associati a un determinato oggetto. L'informazione contenuta nel *barcode* viene catturata da un apposito *barcode reader* e poi trasferita ad un dispositivo a cui il lettore è collegato, il quale si occupa di decodificare ed elaborare i dati acquisiti [1].

Esistono due principiali tipologie di codici a barre: lineari e bidimensionali. I primi rappresentano i dati variando la larghezza e lo spaziamento delle linee parallele. Tuttavia, a causa di limitazioni strutturali, la loro capacità massima è limitata [1]. Esempi comuni di codici a barre lineari includono gli EAN-13, utilizzati a livello globale per la vendita al dettaglio, e i Farmcode, impiegati nel settore farmaceutico italiano e svizzero. D'altra parte, i codici a barre bidimensionali sono rappresentati principalmente dai diffusissimi *QR code*, che possono essere facilmente letti attraverso fotocamere e *smartphone*. Rispetto ai codici lineari, i bidimensionali presentano un notevole miglioramento nella capacità massima di memorizzazione delle informazioni.

I codici a barre, ancora oggi largamente utilizzati, hanno sicuramente dimostrato di essere strumenti efficaci nell'identificazione a distanza, permettendo la rapida visualizzazione e gestione dei dati associati agli oggetti. Tuttavia, è evidente la necessità di affrontare le sfide legate alla capacità di memorizzazione e alle restrizioni che i *barcode* portano con sé.

2.2.2 Carta a banda magnetica

Una carta a banda magnetica è un dispositivo di sola lettura che contiene uno spazio di archiviazione digitale nel quale le informazioni vengono caricate durante la fase di produzione. La banda è composta da minuscole particelle magnetiche di resina e i dati vengono impressi al suo interno attraverso l'applicazione di un campo magnetico nelle sue prossimità. La lettura avviene tramite contatto fisico, facendo passare la carta di fronte ad un apposito dispositivo dotato di una testina a lettura magnetica [1].

Tra le svariate applicazioni, quelle di maggiore successo si riscontrano sicuramente nelle carte di pagamento e nei sistemi di riconoscimento e autenticazione, nonostante le limitazioni evidenti di questa soluzione tecnologica. Si evidenzia, ad esempio, l'elevato rischio di cancellazione o corruzione delle informazioni a causa del sottile strato della banda magnetica, sensibile sia a danneggiamenti fisici che a smagnetizzazioni causate da fonti elettromagnetiche. Nonostante i successivi tentativi di migliorare la magnetizzazione della banda con l'introduzione delle

carte ad alta coercitività[3], rendendo più difficile la smagnetizzazione senza l'uso di strumenti specifici, le carte a banda magnetica continuano a presentare gravi problemi di sicurezza. Il problema principale è la facilità con cui i dati memorizzati al suo interno possono essere letti.

Per affrontare queste vulnerabilità, soprattutto nel contesto delle carte di credito, a partire dagli anni '90 è stato introdotto un chip all'interno di queste carte, trasformandole in *smart card*. Questa evoluzione ha significativamente migliorato la sicurezza, riducendo il rischio di clonazione e consentendo una gestione più avanzata delle informazioni. Tale avanzamento tecnologico ha così rappresentato un significativo passo in avanti rispetto alle tradizionali carte a banda magnetica.

2.2.3 Smart card

Una *smart card* (in italiano *carta intelligente*), si configura come un dispositivo hardware che racchiude potenzialità di elaborazione e memorizzazione dati in grado di garantire elevati standard di sicurezza. Vengono messi insieme diversi componenti, tra cui microprocessore, memorie e antenne, all'interno di un unico circuito elettrico, formando così un circuito integrato (in inglese *integrated circuit*, abbreviato IC) che costituisce il nucleo principale della *smart card*.

L'origine di questa innovazione si deve ai due inventori tedeschi Jürgen Dethloff e Helmut Grötrupp che, nel 1968, incapsularono un circuito integrato in un supporto di plastica. Tuttavia, il brevetto per una scheda trasportabile capace di memorizzare e gestire dati fu concesso a Roland Moreno nel 1974. L'effettiva rivoluzione e diffusione della tecnologia si manifestò dagli anni 2000, grazie alle nuove tecniche di miniaturizzazione che consentirono la produzione di IC sempre più compatti a costi più contenuti.

Il microcircuito integrato è connesso a un'interfaccia di collegamento, come una contattiera o un'antenna, la quale consente il dialogo con i terminali di lettura. In base delle caratteristiche del microcircuito integrato e del tipo di interfaccia di collegamento, le *smart card* possono essere categorizzate come:

- *smart card* a sola memoria oppure a microprocessore;
- *smart card* con contatto, senza contatto oppure a doppia interfaccia.

[3]Per coercitività si intende l'intensità del campo magnetico inverso che è necessario applicare a un materiale per annullare la sua magnetizzazione dopo che questa ha raggiunto il suo valore di saturazione.

Smart card a sola memoria e a microprocessore

Le *smart card* a sola memoria, anche dette *memory card*, si focalizzano esclusivamente sulla memorizzazione sicura dei dati. Il microcircuito integrato all'interno di queste carte permette la lettura e la scrittura attraverso un insieme di funzioni pre-programmate in un circuito logico durante la fase di produzione. Questo circuito logico include un meccanismo di protezione dell'accesso ai dati basato su un insieme di permessi.

D'altra parte, le *smart card* a microprocessore, o *microprocessor card*, offrono un livello superiore di potenza di calcolo, garantendo elevata affidabilità, inattaccabilità e la capacità di elaborare e memorizzare informazioni in modo sicuro. Nella memoria del microcircuito è installato una sorta di sistema operativo, il quale gestisce internamente la memoria e fornisce funzioni avanzate: non solo lettura e scrittura dei dati, ma anche programmazione dei permessi di accesso, operazioni crittografiche, e altro ancora. La presenza di un sistema operativo rende queste *smart card* altamente programmabili, consentendone l'ottimizzazione e la personalizzazione per specifiche applicazioni o anche l'integrazione di più applicazioni, eventualmente che interagiscono tra loro, sullo stesso dispositivo.

Smart card con e senza contatto

La classificazione delle *smart card* a seconda della necessità di contatto o meno si basa sul tipo di interfaccia di collegamento esistente tra il microcircuito integrato e il mondo esterno. Le *smart card* con contatto (*contact smart card*) vengono inserite all'interno di un apposito dispositivo terminale e utilizzano una contattiera per ricevere l'alimentazione e dialogare con l'esterno. Le *smart card* senza contatto (*contactless smart card*), invece, adottano la tecnologia RFID. In generale, fanno uso di un'antenna a radiofrequenza che reagisce alla presenza di un campo elettromagnetico emesso da un particolare dispositivo di lettura e scrittura. Ciò consente al microcircuito integrato di scambiare informazioni con l'esterno, purché l'antenna si trovi entro una certa distanza minima dal dispositivo.

In aggiunta, le carte a doppia interfaccia offrono un modello ibrido che incorpora entrambe le soluzioni di comunicazione con l'esterno. Questo consente, ad esempio, di integrare sullo stesso dispositivo sia applicazioni complesse, come quelle di firma digitale tipiche delle *smart card* con contatto, sia applicazioni più semplici e veloci, come quelle di controllo dell'accesso ad aree riservate, che funzionano bene con un'interfaccia senza contatto. Questo approcco ibrido offre flessibilità nell'uso delle *smart card*, consentendo una vasta gamma di applicazioni in diversi contesti e scenari.

Standard tecnici

Le *smart card* aderiscono allo standard internazionale ISO/IEC 7816[4]. Esso rappresenta un'estensione del modello ISO/IEC 7810[5] sulle caratteristiche fisiche delle carte d'identificazione. All'interno di quest'ultimo vengono definite quattro diverse dimensioni di carte, tutte con uno spessore di 0,76 mm:

- ID-000 (25 mm x 15 mm): utilizzato per le *mini-SIM card*;
- ID-1 (85,6 mm x 53.98 mm): il formato più comune, utilizzato per patenti europee, carte di credito e documenti d'identità elettronici;
- ID-2 (105 mm x 88 mm): utilizzato per le carte d'identità tedesche fino al 2010;
- ID-3 (125 mm x 88 mm): comunemente utilizzato per passaporti e visti.

Lo standard ISO/IEC 7816 si compone di più di dieci parti, ma durante questa ricerca è stata maggiormente attenzionata ISO/IEC 7816-4[6]. Questa specifica parte del documento delinea, in particolare, i seguenti contenuti [4]:

- contenuto delle coppie *command-response* scambiate sull'interfaccia;
- mezzi per recuperare i dati presenti all'interno della carta;
- modalità di accesso ai *file* e ai dati presenti sulla carta;
- un'architettura di sicurezza che definisce i permessi di accesso ai *file* e ai dati contenuti nella carta;
- meccanismi per identificare e indirizzare le applicazioni presenti nella carta;
- meccanismi di comunicazione sicura mediante *secure messaging*.

[4]Per approfondimenti si veda ISO/IEC 7816, *Identification cards – Integrated circuit cards*.

[5]Per approfondimenti si veda ISO/IEC 7810:2019, *Identification cards – Physical characteristics*.

[6]Si evidenzia che la prima versione dello *standard* ISO/IEC 7816-4, datata 1995, era valida per le sole *contact smart card*. Dal 2005, invece, viene specificato dalla stessa ISO che "ISO/IEC 7816-4:2005 è indipendente dalla tecnologia fisica di interfaccia. Si applica alle carte cui si accede tramite uno o più dei seguenti metodi: contatto, accoppiamento stretto e radiofrequenza". Per approfondimenti si veda ISO/IEC 7816-4:2020, *Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange*.

Il primo punto sopra definisce, essenzialmente, la struttura e il contenuto dei comandi che vengono scambiati tra lettore e *smart card*. Questi comandi prendono il nome di APDU (*Application Protocol Data Unit*). Un comando APDU, inviato dal lettore alla *smart card*, è composto da un *header* di 4 byte e da un corpo di dimensione variabile tra 0 a 255 byte.

header				body		
CLA	INS	P1	P2	Lc	Data	Le

Tabella 2.1: Struttura di un comando APDU

In particolare, per i byte dell'*header* e del *body* si ha che:

- *CLA* indica la classe dell'applicazione;
- *INS* rappresenta il gruppo di istruzioni;
- P_1 e P_2 sono i parametri impiegati nel comando;
- L_c specifica la lunghezza in byte del campo *Data*;
- *Data* contiene i dati da trasferire sulla *smart card* per eseguire il comando;
- L_e specifica la lunghezza in byte dei dati che ci si aspetta essere restituiti in risposta al comando inviato.

Una risposta APDU, inviata dalla *smart card* al lettore, è invece composta da 2 byte, SW_1 e SW_2, riguardanti lo stato della *response* affinché si possa indicare se l'operazione è avvenuta con successo o meno, e da un corpo di dimensione variabile tra 0 e 65536 byte per i dati.

SW1	SW2	Data

Tabella 2.2: Struttura di una risposta APDU

Focalizzandoci sulle *contactless smart card*, è importante notare che, in base alle loro caratteristiche operative, sono state definite tre diverse tipologie [1]:

- le *close coupling smart card* (in italiano *carte ad accoppiamento stretto*) aderiscono allo standard ISO/IEC 10536[7] e operano a una distanza massima di circa 1 cm dal lettore;

[7]Per approfondimenti si veda ISO/IEC 10536, *Identification cards – Contactless integrated circuit(s) cards – Close-coupled cards.*

- le *proximity coupling smart card* (in italiano *carte di prossimità*) aderiscono allo standard ISO/IEC 14443[8] e operano a una distanza massima dal lettore di circa 10 cm ad una frequenza di 13,56 MHz;
- le *vicinity coupling smart card* (in italiano *carte di vicinanza*) aderiscono allo standard ISO/IEC 15693[9] e operano a una distanza massima dal lettore di circa 1 m, sempre ad una frequenza di 13,56 MHz.

Tra le tipologie più popolari spiccano sicuramente le carte di vicinanza e quelle di prossimità. Le prime, con la capacità di funzionare con campi magnetici meno intensi rispetto alle seconde, sono ampiamente utilizzate per la tracciabilità dei prodotti, il controllo degli accessi, ad esempio per le carte di ingresso nelle camere degli alberghi, come borsellino elettronico e nei sistemi di bigliettazione elettronica. D'altra parte, le *proximity smart card* hanno ricevuto maggiore riconoscimento, soprattutto in ambito NFC. Sono impiegate non solo nei sistemi di bigliettazione elettronica, ma anche nei sistemi di pagamento e per le carte d'identità elettroniche. In aggiunta, ISO/IEC 14443-2[10] introduce, in funzione della modulazione del segnale utilizzata, due tipologie, o modalità, per le carte di prossimità: Tipo A e Tipo B. Attualmente, le implementazioni più diffuse delle *proximity smart card* sono le tre esposte qui di seguito.

- MIFARE: carta di prossimità di Tipo A di proprietà di NXP Semiconductors, azienda *spin-off* di Philips. Ad oggi, questa implementazione è diventata uno standard *de facto*, utilizzata in oltre l'80% di tutte le *smart card* senza contatto del mondo. Inoltre, queste carte si dividono ulteriormente in due famiglie:
  - carte MIFARE standard: *proximity smart card* a sola memoria che aderiscono alle prime tre parti dello standard ISO/IEC 14443 ma che utilizzano un protocollo di trasmissione proprietario;
  - carte T=LC[11]: carte di prossimità a microprocessore che seguono completamente lo standard, inclusa la quarta e ultima parte sul protocollo di trasmissione;

[8]Per approfondimenti si veda ISO/IEC 14443, *Identification cards – Cards and security devices for personal identification – Contactless proximity objects.*

[9]Per approfondimenti si veda ISO/IEC 15693, *Identification cards – Cards and security devices for personal identification – Contactless vicinity objects.*

[10]Per approfondimenti si veda ISO/IEC 14443-2:2020, *Cards and security devices for personal identification – Contactless proximity objects – Part 2: Radio frequency power and signal interface.*

[11]Il protocollo di trasmissione utilizzato dalle carte di prossimità, definito in ISO/IEC 14443-4, prende il nome di T=LC.

- Calypso: standard internazionale per i sistemi di bigliettazione elettronica che utilizzano *smart card* a microprocessore *contactless* di Tipo B. Originariamente progettato da un gruppo di operatori di trasporto europei, provenienti da Parigi, Bruxelles, Costanza, Lisbona e Venezia, con lo scopo di definire un punto di riferimento e consentire l'interoperabilità dei diversi servizi di trasporto nella stessa area territoriale;

- FeliCa: *proximity smart card* di Sony che differisce sia dal Tipo A sia dal Tipo B. Spesso erroneamente denominata di Tipo C, sebbene la modalità fu proposta ma poi rifiutata. Fu adottata principalmente per i sistemi di pagamento elettronico, ma successivamente si diffuse in numerosi sistemi di bigliettazione elettronica asiatici, come in Giappone e a Singapore.

2.2.4 RFID

Lo studio della tecnologia RFID si colloca, insieme alle analisi fatte sulle *smart card*, tra i più importanti per comprendere non solo l'evoluzione tecnologia verso la NFC, ma anche il vero e proprio funzionamento alla base della tecnologia. L'identificazione a radiofrequenza è una tecnologia di riconoscimento e tracciamento che sfrutta la comunicazione mediante onde radio per lo scambio di informazioni tra due elementi a distanza. Essa, generalmente, si basa sulla memorizzazione dei dati in un apposito dispositivo elettronico passivo, detto RFID *tag*, o etichetta, capace di rispondere a particolari dispositivi attivi, detti RFID *reader*.

Sebbene, il primo brevetto in cui si può leggere la sigla RFID si deve a Charles Walton ed è datato 1983, la tecnologia vera e propria fu in realtà depositata dieci anni prima, nell'agosto del 1973. Tuttavia, la sua origine ha riscontri ancora più lontani nel tempo: durante la seconda guerra mondiale. Difatti, la RFID è comunemente accostata al sistema *Identification Friend or Foe* (IFF) sviluppato in Inghilterra nel 1940. Il sistema IFF veniva montato a bordo degli aerei alleati per emettere impulsi elettromagnetici attraverso una sorgente, solitamente sotto forma di ricetrasmettitore. Una volta colpiti i *transponder* presenti su altri aerei in volo entro il raggio d'azione, l'informazione di ritorno veniva processata, identificando così gli aerei alleati e distinguendoli da quelli nemici. Successivamente, la tecnologia RFID prese piede anche in ambito civile, evolvendosi in sistemi per il tracciamento delle rotte dei carri ferroviari, e si diffuse principalmente dagli anni '90 in poi in svariati settori, fino ad arrivare ad oggi, in cui rappresenta una delle tecnologie più massivamente realizzate nella nostra epoca [5].

In un sistema RFID si possono distinguere tre componenti fondamentali:

- il *tag* RFID, ossia un *transponder* a radiofrequenza di piccole dimensioni costituito da un circuito integrato, cioè un *microchip*, con funzioni di semplice

logica di controllo, dotato di memoria non volatile, connesso ad un'antenna RF ed inserito, o incorporato, in ciò che si vuole identificare;

- il lettore RFID, ossia un ricetrasmettitore controllato da un microprocessore ed utilizzato per interrogare i *tag* e ricevere le informazioni da essi;
- possibilmente, un sistema di gestione che può essere connesso in rete con il *reader*, consentendo, attraverso i codici identificativi contenuti nei *tag*, di ricavare tutte le informazioni disponibili associate agli identificati e di gestire queste informazioni per i propri scopi.

I tag RFID

L'elemento principale che caratterizza un sistema RFID è il *tag*. Come detto sopra, la sua componentistica fondamentale è composta da un circuito integrato e da un'antenna RF, montati insieme su un substrato di sostegno. Il *chip* ha anche il compito di mantenere i dati e, per questo motivo, è ulteriormente costituito da una memoria non volatile. In genere, ogni *tag* è associato ad una specifica entità da identificare e, di conseguenza, l'etichetta possiede un codice univoco. L'antenna è, invece, incaricata di ricevere un segnale da un RFID *reader* e di ritrasmettere una *response* al lettore. I *tag* possono essere classificati sulla base della loro gestione delle fonti energetiche [5].

- *Tag* passivo: l'antenna RF trasforma il segnale ricevuto dal lettore in energia elettrica, tramite il principio di induzione, per alimentare l'IC e, una volta attivato il *microchip*, i dati memorizzati al suo interno vengono trasmessi dall'antenna al *reader* irradiando, e dunque modulando, il segnale trasmesso dal lettore e riflettendolo.
- *Battery-assisted passive tag*: viene aggiunta un piccola batteria interna per alimentare i propri componenti. In particolare:
  - *tag* semi-passivo: la batteria interna alimenta il *microchip* e/o ulteriori apparati ausiliari o sensori, ma non un trasmettitore attivo. Di conseguenza, anche qui la trasmissione avviene per induzione e l'etichetta si comporta come un *tag* passivo;
  - *tag* semi-attivo: propriamente, la batteria dovrebbe alimentare il *chip* e il trasmettitore ma, per motivi di risparmio energetico, di solito il *tag* viene disattivato e la sua attivazione si ottiene tramite un ricevitore che opera con la stessa soluzione dei *tag* passivi.
- *Tag* attivo: viene integrata una batteria all'interno del *tag* ma, in questo caso, per alimentare non solo l'IC ed eventuale sensoristica, ma anche l'antenna.

Ciò significa che il *tag* utilizza un trasmettitore attivo, anziché uno passivo, consentendo una comunicazione su lunghe distanze.

Inoltre, a seconda della modalità di attivazione utilizzata, è possibile ulteriormente specializzare i *tag* attivi in *tag* temporizzati, o *beacon*, e in *tag* attivabili, o *transponder*. I primi emettono segnali a intervalli prestabiliti senza la necessità di essere interrogati da un *reader*, mentre quelli attivabili entrano in funzione in modo automatico solo dopo aver ricevuto un segnale da parte di un lettore.

Le frequenze operative

Le frequenze di comunicazione tra *reader* e *tag* dipendono sia dalla natura dell'etichetta sia dalle applicazioni previste e sono regolate da organismi internazionali e nazionali. Tuttavia, la regolamentazione è divisa in regioni geografiche con diverse normative da regione a regione, comportando spesso incompatibilità.

Le bande di frequenza su cui generalmente un sistema RFID opera sono elencate qui di seguito [5].

- La banda LF (*Low Frequencies*), in particolare la sotto-banda 125-145 kHz. Essa è valida in tutto il mondo ed è storicamente la prima banda utilizzata per l'identificazione automatica.

- La banda HF (*High Frequencies*), in particolare la frequenza a 13,56 MHz. Essa è valida in tutto il mondo, è considerata universale e ad oggi rappresenta la banda più diffusa, nonché quella utilizzata da NFC.

- La banda UHF (*Ultra High Frequencies*) media, in particolare le sotto-bande 865-870 MHz in Europa, 902-928 MHz in USA e 950 MHz in Asia. Essa viene considerata la nuova banda per l'identificazione a radiofrequenza per la logistica e la gestione dei singoli oggetti con distanze operative significativamente maggiori rispetto ad LF e HF, ma purtroppo non esiste una normativa universale.

- La banda UHF alta, in particolare la frequenza a 2,4 GHz. Essa consente una ulteriore miniaturizzazione del *tag*, ma si tratta di una banda molto affollata da altre tecnologie, come Wi-Fi, Bluetooth e ZigBee, con le quali è necessario convivere.

Esistono anche altre frequenze utilizzabili ma molto meno diffuse, come la banda UHF bassa, e in particolare la sotto-banda 433-435 MHz, valida solo in Europa, la banda SHF (*Super High Frequencies*), e in particolare la frequenza a 5,8 GHz, usata ad esempio per il Telepass, e la banda UWB (*Ultra Wide Band*) con frequenze maggiore di 5,8 GHz.

2.3 La comunicazione di prossimità

Mentre le *smart card* si sono affermate come soluzioni intelligenti e versatili per la memorizzazione sicura dei dati e l'autenticazione nelle transazioni elettroniche, la tecnologia RFID offre un approccio ancora più dinamico. Con l'utilizzo di campi elettromagnetici, la RFID consente di identificare e tracciare gli oggetti in modo efficiente, aprendo la strada a una vasta gamma di applicazioni innovative e pratiche. Ciò che rende questa esplorazione particolarmente entusiasmante è la stretta collaborazione che esiste tra le tecnologie delle *smart card* e della RFID. Questa sinergia non solo amplifica le capacità di entrambe le tecnologie ma getta le basi per lo sviluppo della NFC. L'evoluzione continua di queste tecnologie promette di trasformare radicalmente il modo in cui interagiamo con il mondo digitale e fisico, aprendo la strada a una connettività ancora più integrata e immediata.

La tecnologia NFC rappresenta la seconda generazione delle tecnologie di prossimità senza contatto basate sulla radiofrequenza e mette insieme caratteristiche derivanti sia dalle tecnologie delle *smart card* senza contatto sia dalla tecnologia RFID al fine di consentire interazioni a corto raggio e a bassa banda [5]. Più precisamente, la tecnologia NFC opera ad una distanza operativa di circa 10 cm e ad una frequenza di 13,56 MHz. Si noti come i parametri di funzionamento di un sistema NFC coincidano esattamente con le caratteristiche operative delle *proximity smart card* di ISO/IEC 14443. A differenza della tecnologia RFID, la NFC supera la distinzione tra *tag* e *reader*. Infatti, l'idea innovativa sta proprio nell'accoppiamento in un solo circuito integrato di un dispositivo attivo, come un *reader*, e di un *tag* che operi con le stesse caratteristiche operative delle *proximity smart card*. Ciò rende la tecnologia adatta a supportare, ad esempio, una comunicazione *peer-to-peer*, nonché l'accesso sicuro ai dati, basandosi sullo stesso modello delle *smart card* [5]. Una delle caratteristiche fondamentali della NFC è la sua sicurezza implicita, data dalla breve distanza a cui avviene la comunicazione tra dispositivi compatibili. Infatti, la vicinanza tra i due dispositivi rende l'intercettazione del segnale poco probabile. In aggiunta, la distanza operativa così ridotta richiede un'effettiva intenzione dell'utente affinché un sistema NFC possa operare [1].

2.3.1 Componenti di un sistema NFC

La peculiarità introdotta con la tecnologia NFC di andare oltre la distinzione tra lettore e *tag* porta alla definizione di un nuovo insieme di componenti del sistema [1].

- Un NFC *mobile* è uno *smartphone* abilitato all'utilizzo della tecnologia, come componente sia attivo che passivo, e rappresenta il dispositivo più importante del sistema, manifestando di fatto l'innovazione NFC. In aggiunta, offre alla

tecnologia grandi opportunità sia a favore della facilità d'uso che della sua adozione.

- Un NFC *reader* è un dispositivo attivo capace di trasferire dati ed effettuare operazioni di lettura e scrittura su un altro componente NFC. L'esempio più comune è sicuramente dato dal terminale POS senza contatto per eseguire pagamenti elettronici *contactless*.
- Un NFC *tag* è di fatto un RFID *tag* passivo. L'NFC Forum ha, tuttavia, pubblicato vari standard, definendo diverse tipologie di *tag* basate su su ISO/IEC 14443 Tipo A, Tipo B oppure Sony FeliCa.

L'avvicinamento di un dispositivo NFC a un altro rappresenta la condizione necessaria affinché la comunicazione abbia inizio. Non vi è la necessita di interagire ulteriormente con il sistema una volta che un dispositivo NFC viene avvicinato a un altro NFC *tag*, *reader* o *mobile*. Questa caratteristica rappresenta, di fatto, una applicazione concreta del concetto di *ubiquitous computing* [1].

Per ogni sessione NFC si può identificare un *initiator*, ossia il componente che avvia la comunicazione, e un *target*, che risponde alla richiesta inviata dal primo. Risulta immediata l'analogia con la classica architettura *client-server*, in cui il *client* invia una richiesta al *server*, avviando una comunicazione, e quest'ultimo risponde. Si può, inoltre, sostituire l'invio e la ricezione rispettivamente di richieste e risposte HTTP con le corrispettive strutture APDU. Come in RFID, un dispositivo attivo è in grado di generare un campo elettromagnetico e di trasmettere dei segnali, mentre un dispositivo passivo sfrutta il campo elettromagnetico generato e il principio di induzione per alimentare l'IC e rispondere ad un componente attivo. Di conseguenza, l'*initiator* deve sempre essere un dispositivo NFC attivo. D'altra parte, il *target* può essere sia attivo che passivo. Ovviamente, un NFC *tag*, essendo un dispositivo passivo, può esclusivamente agire da *target*, quindi memorizza dei dati che possono essere letti da un componente attivo [1].

2.3.2 Modalità di funzionamento

Un sistema NFC ammette tre diverse modalità di funzionamento che prevedono la comunicazione tra due dispositivi con una valida combinazione. In generale, la tecnologia NFC prevede l'interazione tra un NFC *mobile* e un qualsiasi altro componente [1].

- Analogamente ad un sistema RFID, la modalità *reader/writer* prevede la comunicazione tra un NFC *mobile* e un NFC *tag* per lo scambio di dati. In questo caso, lo scopo è quello di poter recuperare delle informazioni dal *tag* oppure di scrivere e memorizzare dei dati al suo interno. Di conseguenza, è possibile ulteriormente specializzare la modalità in *reader mode* e *writer mode*.

- Dall'innovazione NFC deriva sicuramente la modalità *peer-to-peer* che permette a due NFC *mobile* di scambiare informazioni tra loro. Sebbene entrambi operino come due dispositivi attivi, essi instaurano una comunicazione bidirezionale *half-duplex*: quando un dispositivo invia informazioni, l'altro resta in ascolto e può avviare la propria trasmissione di dati solo quando il primo ha concluso.
- Un'ulteriore caratteristica fondamentale della tecnologia NFC è rappresentata dalla modalità *card emulation*. Essa permette di utilizzare un NFC *mobile* come se fosse una *smart card* senza contatto, come una *proximity smart card* ISO/IEC 14443, per interagire con un NFC *reader*. Esattamente attraverso l'utilizzo di questa modalità di funzionamento, la tecnologia NFC consente di utilizzare il proprio *smartphone* per effettuare transazioni elettroniche.

Lo standard NDEF (*NFC Data Exchange Format*) rappresenta uno degli strumenti più importanti a supporto delle prime due modalità di funzionamento. Esso definisce il formato dei dati per lo scambio di informazioni tra dispositivi NFC. In particolare, rappresenta un formato binario che consente di incapsulare uno o più *payload* in un singolo messaggio. Si basa sul concetto di *record* e ogni messaggio può, dunque, contenere uno o più *record* NDEF, che possono essere concatenati tra loro per supportare l'invio di più dati aggregati [1]. Tuttavia, nonostante l'importanza dello standard, non sempre quest'ultimo viene utilizzato nelle applicazioni NFC.

Capitolo 3

Transazioni digitali

I pagamenti digitali rappresentano una strategia moderna per effettuare transazioni bancarie in modo elettronico, diventando ormai una componente fondamentale nel contesto finanziario e commerciale. Questa forma di pagamento elimina la necessità dei contanti, consentendo di pagare attraverso diversi strumenti digitali [6].

- Le carte di credito o debito rappresentano uno dei metodi di pagamento digitale più diffusi e consentono di effettuare transazioni sia online sia presso esercizi commerciali.
- I bonifici bancari elettronici permettono il trasferimento di denaro da un conto corrente ad un altro attraverso servizi di *online banking* o tramite applicazioni mobili.
- I portafogli digitali, anche detti *e-wallet*, consentono di memorizzare le proprie carte di pagamento elettroniche e di effettuare transazioni tramite dispositivi mobili, ad esempio mediante Apple Pay e Google Pay.
- Un codice QR può essere implementato affinché consenta di effettuare pagamenti mediante la sua scansione con uno *smartphone*.
- Le criptovalute, come Bitcoin ed Ethereum, sono valute digitali decentralizzate che consentono di effettuare transazioni online in modo sicuro e anonimo.
- La tecnologia NFC permette di effettuare pagamenti avvicinando un dispositivo compatibile, come uno *smartphone* o uno *smartwatch*, a un terminale di pagamento abilitato.

Un terminale POS, invece, è un dispositivo che permette di eseguire pagamenti digitali con carta di credito, debito e prepagata attraverso la lettura di un chip, con o senza contatto, abilitando dunque anche la possibilità di effettuare pagamenti NFC [7].

Il primo caso di studio analizzato al fine di sviluppare nuove esperienza interattive mediante NFC per un'applicazione iOS riguarda l'utilizzo di un iPhone come terminale POS, permettendo transazioni digitali NFC senza l'utilizzo di dispositivi terzi. Di conseguenza, in questo capitolo verrà brevemente introdotto il terminale POS, ad oggi fondamentale per eseguire transazioni digitali, e si approfondiranno, inoltre, i processi svolti durante una transazione eseguita tramite il servizio Apple Pay.

3.1 Sistema POS

Un *point of sale* (letteralmente *punto di vendita*, abbreviato POS), o terminale di pagamento, è un dispositivo che permette di effettuare pagamenti mediante moneta elettronica. Offrendo effettivamente una prestazione bancaria, è necessario stipulare un contratto con la propria banca per poter usufruire del servizio ed eseguire transazioni attraverso il terminale POS. Solitamente, il dispositivo è un elemento del contratto stesso. In particolare, infatti, il servizio viene offerto dal centro di elaborazione della banca, o dal gruppo di banche, che permette di autorizzare ed effettuare l'addebito sul conto corrente del soggetto abilitato e l'accredito sul conto dell'esercente, in tempo reale o differito.

Il terminale POS funge, di fatto, da lettore si carte di pagamento sia fisiche che digitali. Il trasferimento dei dati può avvenire secondo tre diverse modalità di lettura [7]:

- tramite lettura di una carta a banda magnetica;
- tramite lettura dell'IC di una *contact smart card*;
- tramite tecnologia senza contatto, sfruttando la NFC.

3.1.1 Tipologie di terminali

I terminali POS sono a tutti gli effetti dei dispositivi informatici che, oltre ad offrire la possibilità di essere connessi ad una rete LAN, possiedono anche le classiche funzioni di programmazione e configurabilità di un mini computer. In generale, la comunicazione tra il dispositivo e chi fornisce il servizio avviene tramite la linea telefonica oppure attraverso una connessione ad internet. Tra il dispositivo e la stazione ricevente può esserci:

- una rete cablata;
- una rete senza fili, nel caso in cui la comunicazione avviene, ad esempio, mediante Bluetooth o Wi-Fi;

- nessun tipo di rete diretta, come nel caso di un'applicazione installata su uno *smartphone* a cui il POS è connesso.

A seconda delle tecnologie utilizzate e delle caratteristiche del dispositivo, si possono distinguere diverse tipologie di terminale di pagamento [7].

- Il POS fisso rappresenta la tipologia di terminale di pagamento più tradizionale e maggiormente utilizzata. Il dispositivo è connesso tramite linea telefonica o cavo di rete LAN.
- Il POS *wireless* è un'evoluzione del POS fisso che fornisce un modulo funzionale rimovibile dall'unità base connessa via cavo, consentendo la portabilità del terminale di pagamento entro una decina di metri. La comunicazione tra la base e il terminale portatile avviene solitamente tramite Bluetooth.
- Il POS GSM/GPRS è dotato di scheda SIM che permette il suo funzionamento anche in assenza di una linea fissa. Inoltre, con l'avanzamento delle tecnologia di telecomunicazione, i modelli più recenti di questo tipo di terminale integrano anche connessioni UMTS.
- Il POS Mobile, anche detto *mPOS*, è la diretta evoluzione della precedente tipologia, nonché la versione più avanzata di questo tipo di dispositivi. A differenza del precedente, piuttosto che integrare una scheda SIM, prevede una connessione Bluetooth ad uno *smartphone*, o ad un *tablet*, sul quale viene installata un'applicazione dedicata, e sfrutta la connessione di quest'ultimo.

Infine, negli ultimi tempi si sta particolarmente diffondendo un'ulteriore proposta che guarda completamente alla digitalizzazione del servizio. Questa tipologia di terminale di pagamento viene, appunto, chiamata POS digitale e permette di gestire i pagamenti e gli incassi online tramite un'applicazione *web* dedicata. PayPal e Stripe rappresentano le due aziende più affermate ed utilizzate per la gestione dei POS digitali.

3.2 Apple Pay

Apple integra la tecnologia NFC all'interno dei suoi *smartphone* a partire da iPhone 6 e con il rilascio del sistema operativo iOS 8.1, nell'ottobre del 2014, rende disponibile agli utenti il servizio finanziario Apple Pay, offrendo la possibilità di eseguire pagamenti NFC tramite il proprio iPhone. Agli occhi dei consumatori, l'utilizzo del servizio risulta essere molto semplice: è sufficiente inserire i dati di una carta di credito o di debito emanata da un ente finanziario abilitato all'interno del proprio Apple Wallet e, richiamato il proprio metodo di pagamento quando necessario, basta avvicinare il proprio *smartphone* al terminale POS. Sebbene

queste operazioni appaiono estremamente intuitive, una serie di processi ed entità vengono richiamati durante i pochi secondi in cui si esegue una transazione Apple Pay.

3.2.1 Memorizzazione del metodo di pagamento

Per effettuare un pagamento ad un terminale POS mediante Apple Pay, per prima cosa, è necessario associare un metodo di pagamento al proprio Apple Wallet. Quando viene aggiunta una carta di credito o di debito all'interno dell'*e-wallet* di Apple, le informazioni necessarie vengono prima inviate ai server dell'azienda di Cupertino e poi inoltrate alla rete del circuito pertinente: Visa, Mastercard, American Express, o simili. Quest'ultimo ha il compito di validare i dati ricevuti con la banca emittente e, se la validazione si conclude con successo, la rete del circuito prende il ruolo di *Token Service Provider* (TSP). Il TSP è incaricato di generare un *token*, detto *Device Account Number* (DAN), e una *token key* attraverso un processo di *tokenizzazione* [8].

La *tokenizzazione* è un processo sempre più adottato nel contesto della sicurezza dei dati e permette di sostituire dati sensibili con equivalenti non sensibili detti *token*. Un *token* non ha alcun significato o valore sfruttabile, ma rappresenta un riferimento ad un dato sensibile, come un identificatore. La mappatura dei dati originali ai corrispettivi *token* sfrutta metodi che rendono impossibile ricavare i dati sensibili attraverso inversione del *token*, in assenza del sistema di *tokenizzazione*. In Apple Pay, durante l'inserimento di una carta di pagamento in Wallet, viene utilizzata la *tokenizzazione* per rimpiazzare il numero della carta, detto *Primary Account Number* (PAN), con un *token*, ossia il DAN, ancora riconducibile ad un numero di carta, ma non al PAN originale [8]. Ciò permette di non utilizzare il numero originale del proprio metodo di pagamento durante le transazioni. Inoltre, se il *token* venisse rubato, esso risulterebbe privo di significato da solo. Non esistendo un algoritmo capace di derivare il PAN dal DAN, si rende impossibile per i criminali il lavoro di *reverse engineering*.

Il DAN e la *token key* generati dal TSP vengono inviati ai server Apple, poi inoltrati al dispositivo dell'utente per essere memorizzati all'interno del proprio *Secure Element* (SE) [8]. GlobalPlatform[1] definisce un SE come «una piattaforma resistente alle manomissioni (in genere un microcontrollore sicuro a singolo chip) in grado di ospitare in modo sicuro le applicazioni e i loro dati riservati e crittografici (ad esempio la gestione delle chiavi crittografiche) in conformità alle regole e ai requisiti di sicurezza stabiliti da un insieme di autorità fidate ben identificate» [9].

[1]Associazione senza scopo di lucro che identifica, sviluppa e pubblica specifiche per facilitare l'implementazione e la gestione di più applicazioni su *smart card* garantendo sicurezza ed interoperabilità.

Essenzialmente, si tratta di un componente incorporato nel chip NFC del dispositivo, utilizzato per garantire sicurezza sia ai dati memorizzati sia alle applicazioni abilitate alla NFC, come le transazioni mediante *card emulation*. Di conseguenza, Apple Pay utilizza l'SE per memorizzare le informazioni segrete associate ad una *tokenized card*.

Visualizzazione del PAN e del DAN su Apple Wallet

Le ultime quattro cifre del DAN associato ad un metodo di pagamento inserito in Apple Wallet rappresentano un'informazione visibile all'utente e, in alcune circostanze, potrebbe essere necessario recuperarle.

Su iPhone, aprendo l'applicazione Wallet, verranno mostrati i metodo di pagamento inseriti. Le carte a schermo permettono la visualizzazione delle ultime quattro cifre PAN, situate in basso a sinistra. Selezionando un metodo di pagamento, verranno visualizzati gli ultimi movimenti eseguiti con quella carta tramite Apple Pay. Tra le opzioni mostrate aprendo il menu in alto a destra, vi è la possibilità di visualizzare il numero carta. La successiva schermata mostrerà sia le ultime quattro cifre della carta fisica, cioè del PAN, sia le ultime quattro cifre per Apple Pay. Quest'ultime corrispondono alle ultime quattro cifre del DAN generato mediante *tokenizzazione* del PAN.

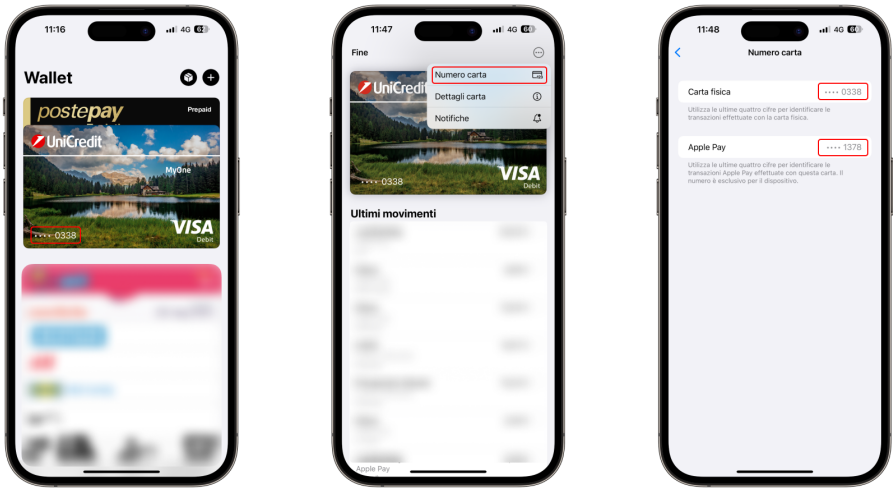

Figura 3.1: Ultime quattro cifre del PAN e del DAN su Apple Wallet

3.2.2 Transazione POS mediante Apple Pay

Inserito un metodo di pagamento in Wallet, è possibile richiamarlo per eseguire un pagamento NFC, in modalità *card emulation*, tramite Apple Pay. In generale, per avviare l'operazione finanziaria, vengono recuperate le informazioni necessarie dall'SE, come il DAN e la *token key*, poi inviate al terminale POS. Quest'ultimo processa la transazione inoltrandola all'*acquirer*: una banca, o un istituto finanziario, che elabora i pagamenti con carta di credito o di debito per conto di un commerciante [8]. L'*acquirer* identifica il *Bank Identification Number* (BIN), anche conosciuto come *Iusser Identification Number* (IIN). Esso è rappresentato dalle prime cifre del numero della carta, le quali consentono di riconoscere l'istituzione che ha emesso il metodo di pagamento utilizzato. In questo modo, l'*acquirer* è in grado di inoltrare le informazioni ricevute alla rete del circuito della carta. Riconosciuto che il numero di carta si tratta di un *token*, le informazioni vengono validate attraverso l'uso della propria copia della *token key* e, dopo una serie di altre validazioni addizionali, la rete del circuito completa la *de-tokenizzazione* del DAN, ottenendo il PAN originale. Concluse queste operazioni, viene inviata una richiesta di transazione, con il PAN, alla banca emittente della carta. Essa autorizza il pagamento ed invia una *response* che sarà poi inoltrata al terminale POS [8].

Capitolo 4

Uso di iPhone come POS

Compreso il funzionamento della tecnologia NFC, nonché le sue modalità operative, e introdotti i concetti di transazione digitale e terminale POS, è ora il momento di analizzare effettivamente la possibilità di utilizzare un iPhone come terminale di pagamento tramite NFC. Ciò consentirebbe, di conseguenza, di accettare pagamenti NFC direttamente tramite il proprio *smartphone*, senza l'ausilio di dispositivi terzi.

Pertanto, in questo capitolo si introdurrà una prima versione dell'applicazione iOS realizzata appositamente per questa tesi al fine di sviluppare e verificare possibili integrazioni di nuove esperienze interattive mediante tecnologia NFC. Successivamente, si analizzeranno le API di sistema e i *framework* che Apple fornisce agli sviluppatori per consentire l'utilizzo del chip NFC presente all'interno dei *device* ed, infine, si riporteranno i risultati ottenuti in merito all'utilizzo di iPhone come terminale POS.

4.1 Panoramica dell'applicazione demo

Dato il contesto in cui si è svolta la ricerca, si è scelto di sviluppare un'applicazione iOS dimostrativa che si rifacesse alle ormai comuni *mobile banking app*, utilizzate ogni giorno per gestire il proprio account bancario. In questa prima versione, l'applicazione è composta da tre schermate e permette di aggiungere, visualizzare e rimuovere metodi di pagamento e transazioni.

4.1.1 Schermata principale

Una volta concluso il caricamento dell'*app*, durante il quale viene visualizzata una semplice *splashscreen*, si atterra nella schermata principale. Essa mostra a schermo i metodi di pagamento e la lista di transazioni inserite all'interno dell'applicazione tramite i corrispettivi *form* (Fig. 4.1).

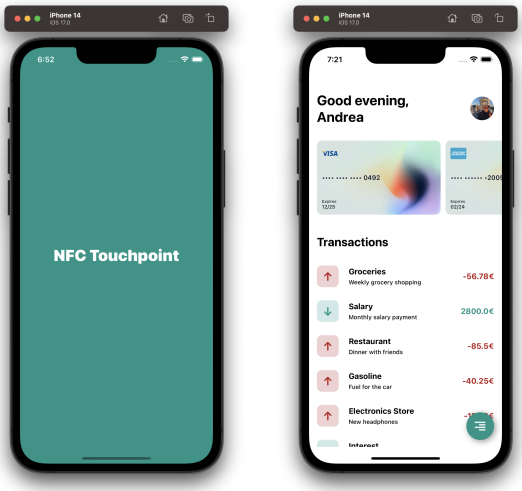

Figura 4.1: *Splashscreen* ed *homescreen* della applicazione demo

Oltre alla semplice visualizzazione, la *home* consente anche di rimuovere le carte di pagamento, attraverso tocco prolungato, e le transazioni, indifferentemente mediante tocco prolungato oppure *swipe* verso sinistra. Inoltre, in basso a destra è presente un *floating button* che permette di aprire un menu per navigare all'interno dell'applicazione (Fig. 4.2).

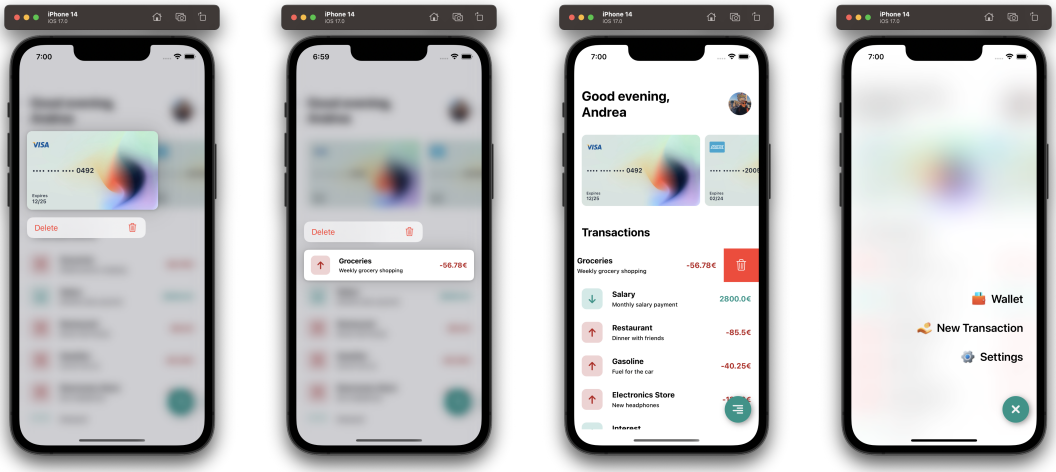

Figura 4.2: Operazioni possibili nella *homescreen*

4.1.2 Inserire una nuova transazione

Raggiunta la schermata che permette di aggiungere una nuova transazione, viene visualizzato un *form* (Fig. 4.3). Per prima cosa, vengono mostrati due bottoni all'utente. Essi consentono di scegliere se aggiungere una transazione in entrata o in uscita. Subito sotto, il modulo richiede le informazioni necessarie alla registrazione di una nuova transazione: importo, titolo e descrizione. La validazione del *form* prevede che tutti i campi siano riempiti dall'utente e, in particolare, per l'importo viene ulteriormente verificato che esso sia positivo e diverso da zero.

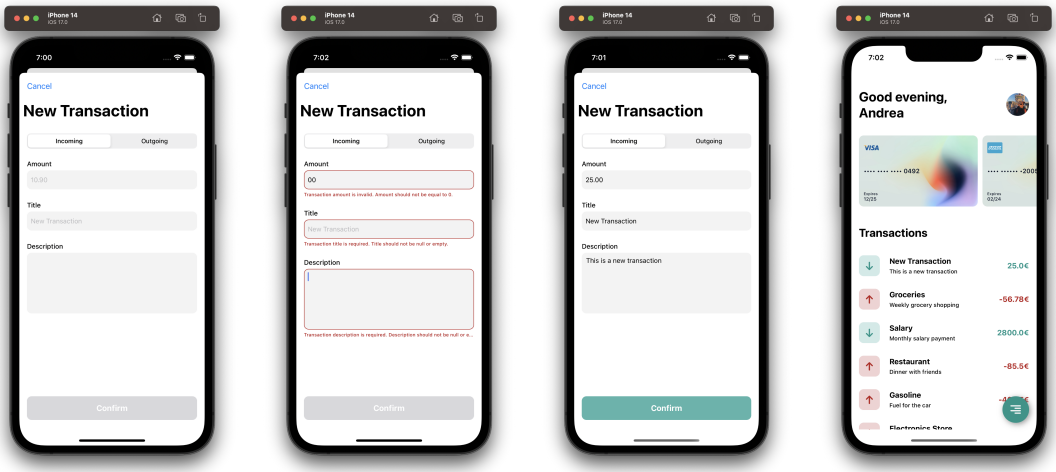

Figura 4.3: Flusso di inserimento di una nuova transazione

4.1.3 Inserire un nuovo metodo di pagamento

Nella schermata principale, spostandosi verso la fine della lista di metodi di pagamento, appare un bottone che permette di inserire una nuova carta. Il *form* presentato richiede di inserire il PAN, la data di scadenza, il CVV e il titolare della carta. Per poter inserire una nuovo metodo di pagamento è necessario che l'utente inserisca tutti i campi richiesti (Fig. 4.4). In particolare, per la validazione del PAN è stato attenzionato il meccanismo che regola la sua struttura e sono state aggiunte delle verifiche addizionali. Il PAN di una carta di pagamento è, infatti, composto da un numero che conta tra le 8 e le 19 cifre. La sua struttura segue solitamente lo standard ISO/IEC 7812[1]. Esso è composto da due parti: la prima regola il formato

[1]Per approfondimenti si veda ISO/IEC 7812, *Identification cards – Identification of issuers.*

dell'IIN (*Iusser Identification Number*) e del PAN, mentre la seconda determina le procedure per la registrazione a livello internazionale degli IIN.

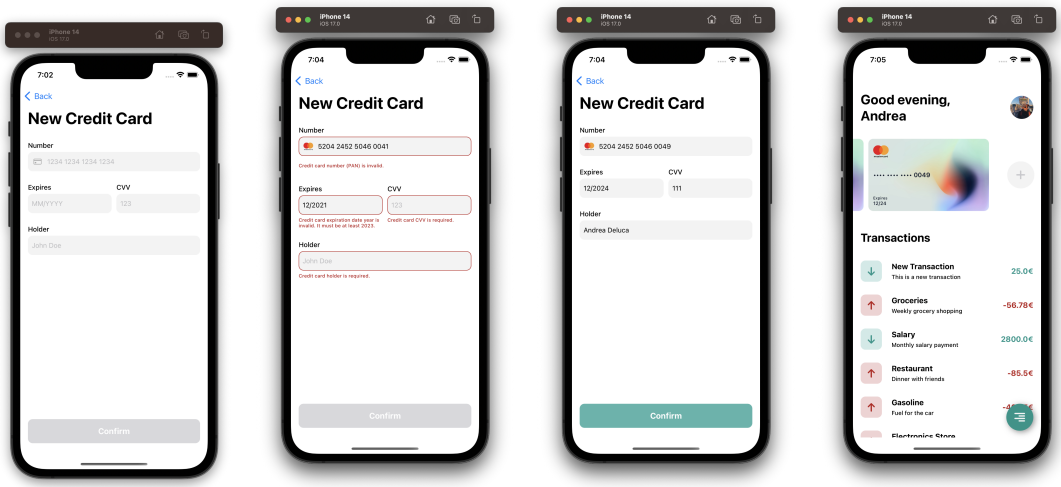

Figura 4.4: Flusso di inserimento di un nuovo metodo di pagamento

Ad oggi, l'IIN è solitamente composto da 6 cifre[2], dove la prima cifra dell'IIN a partire da sinistra rappresenta il cosiddetto *Major Industry Identifier* (MII). L'IIN viene abbinato ad un *Individual Account Identifier*, ossia un numero di identificazione del conto di lunghezza variabile (fino a 12 cifre), e ad un *checksum* a singola cifra, ossia una cifra di controllo calcolata mediante algoritmo di Luhn.

Algoritmo di Luhn

La formula di Luhn, conosciuta anche come *Modulo 10*, descrive un algoritmo che consente di generare e verificare la validità di numeri identificativi. Questa procedura prevede di sommare, da sinistra a destra, tutte le cifre in posizione pari al doppio della somma di quelle in posizione dispari. Si considera, quindi, il modulo rispetto a 10, ossia il resto della divisione per 10, del valore ottenuto e si determina la cifra di Luhn come segue:

[2]Nel 2015 sono iniziati i lavori per apportare una modifica ad ISO/IEC 7812 ed incrementare la lunghezza dell'IIN ad 8 cifre. La revisione del 2017 dello *standard* ha definito il nuovo IIN ad 8 cifre e un intervallo di tempo entro il quale verranno convertiti i precedenti IIN a 6 cifre nei nuovi IIN ad 8 cifre. DI conseguenza, anche l'intero PAN passerà da una lunghezza variabile tra 8 e 19 cifre ad una lunghezza tra 10 e 19 cifre.

- se il modulo è zero, ossia la somma ottenuta è divisibile per 10, allora la cifra di controllo sarà zero;
- altrimenti la cifra di Luhn sarà la differenza tra 10 ed il modulo.

Di conseguenza, per verificare un numero contenente la cifra di Luhn si può procedere come segue:

1. partendo da destra verso sinistra, si calcola la somma del doppio di tutte le cifre in posizione pari e, laddove una moltiplicazione ha dato come risultato un numero a due cifre, esse si sommano per ottenerne una sola;
2. si sommano tutte le cifre in posizione dispari al risultato ottenuto nel punto precedente;
3. se la somma è divisibile per 10, allora la carta è valida.

Validazione del PAN in *app*

Durante la digitazione del PAN da parte dell'utente viene riconosciuto l'IIN della carta. Nello specifico, è stato sviluppato un componente di validazione che accetta esclusivamente i metodi di pagamento con le specifiche riportate in Tab. 4.1.

Circuito	IIN	Lunghezza del PAN	Validazione
American Express	34 oppure 37	15 cifre	Algoritmo di Luhn
MasterCard	51-55	16 cifre	
Visa	4		

Tabella 4.1: Specifiche per le carte di pagamento accettate in *app*

A seconda dell'IIN individuato viene mostrata l'icona del circuito associato. Durante la digitazione, il PAN viene formattato attraverso degli spazi inseriti automaticamente, per rendere più facile ed immediata la sua lettura. Raggiunta la lunghezza indicata, la digitazione di un'ulteriore cifra non apporterà alcuna modifica al testo già inserito, bloccando dunque l'inserimento in questione. Infine, come si può osservare, tutte le tipologie di carte di pagamento accettate ammettono l'algoritmo di Luhn come metodo di validazione. Di conseguenza, è stata prevista la verifica della cifra di controllo del PAN inserito attraverso la procedura descritta in precedenza.

4.2 API di sistema per l'utilizzo della NFC

Per integrare funzionalità NFC all'interno di un'applicazione iOS è necessario analizzare quali possibilità vengono offerte agli sviluppatori per accedere alle

potenzialità della comunicazione di prossimità. Apple mette a disposizione degli sviluppatori iOS un *framework* chiamato CoreNFC, fondamentale per implementare funzionalità riguardanti la tecnologia in esame.

4.2.1 CoreNFC

Il *framework* offre agli sviluppatori la capacità di utilizzare la tecnologia NFC in modalità *reader/writer*[3] per individuare, leggere e scrivere *tag* NFC. Un'applicazione che sfrutta CoreNFC è in grado di leggere un *tag* per fornire all'utente maggiori informazioni sull'ambiente circostante e sugli oggetti del mondo reale che in esso si trovano. Ad esempio, l'utente, semplicemente avviando l'applicazione e avvicinando il suo iPhone a un *tag*, potrebbe ricevere dettagli rilevanti sui prodotti di un supermercato oppure sulle opere che si trovano in un museo. In particolare, CoreNFC permette non solo di leggere e scrivere *tag* NFC che contengono dati in formato NDEF, ma anche di interagire con *tag* che aderiscono a specifici protocolli, come *tag* ISO/IEC 7816, ISO/IEC 15693, FeliCa e MIFARE [10].

Sviluppare un lettore di *tag* mediante CoreNFC

Sviluppare una piccola applicazione che consente di leggere *tag* NFC utilizzando CoreNFC risulta piuttosto semplice.

```
@IBAction func beginScanning(_ sender: Any) {
    guard NFCNDEFReaderSession.readingAvailable else {
        let alertController = UIAlertController(
            title: "Scanning Not Supported",
            message: "This device doesn't support tag scanning.",
            preferredStyle: .alert
        )
        alertController.addAction(
            UIAlertAction(
                title: "OK",
                style: .default,
                handler: nil
            )
        )
        self.present(alertController, animated: true)
        return
    }

    session = NFCNDEFReaderSession(
        delegate: self,
        queue: nil,
        invalidateAfterFirstRead: true
```

[3]A seguito dell'entrata in vigore del *Digital Market Act* (DMA) varato dall'UE, con l'aggiornamento iOS 17.4 del 5 marzo 2024, Apple ha ampliato le capacità di CoreNFC, consentendo anche l'utilizzo del chip NFC in modalità *card emulation*.

```
23      )
24      session?.alertMessage = "Hold your iPhone near the item to learn more"
25      session?.begin()
26  }
```

Codice 4.1: Inizializzazione di una sessione di lettura NFC con CoreNFC [11]

Il Codice 4.1 mostra l'inizializzazione di una sessione `NFCNDEFReaderSession` per la lettura di un *tag* NDEF. La funzione `beginScanning` può essere impostata come azione da invocare al *tap* su un bottone. La prima operazione che troviamo nel flusso della funzione è il controllo della disponibilità di un chip NFC in grado di eseguire la lettura di un *tag* NDEF da parte del dispositivo. Nel caso in cui ci fosse un riscontro negativo, viene presentato un messaggio di errore tramite un *alert* e la funzione termina (righe 2-17). Se, invece, si è constatato che è possibile effettuare la lettura NFC, viene stabilita una sessione (righe 19-23). La classe `NFCNDEFReaderSession` ammette, in particolare, il parametro di inizializzazione `invalidateAfterFirstRead`, ossia un *flag* di configurazione che stabilisce se invalidare la sessione al termine della prima lettura. Di conseguenza, se tale impostazione viene abilitata, si dovrà richiamare la *routine* per avviare una nuova lettura NFC. Il codice termina impostando un messaggio di istruzione da mostrare all'utente ed effettuando la chiamata a funzione `session?.begin()` (righe 24-26). Questa abilita il *polling* delle radiofrequenze sul dispositivo ed inizia la scansione per la ricerca dei *tag*.

Solitamente, la funzione appena descritta viene inclusa all'interno di un *ViewController*, ossia la classe che gestisce la visualizzazione di una schermata, così da associare il comportamento esposto ad una qualche interazione dell'utente con l'applicazione. La gestione degli eventi che possono verificarsi dopo aver avviato la ricerca di un *tag* è delegata ad un componente conforme al protocollo `NFCNDEFReaderSessionDelegate`, passato come argomento di inizializzazione della sessione. In questo semplice esempio, esso è ancora rappresentato dal *ViewController* (Fig. 4.2).

```
1  extension MyViewController: NFCNDEFReaderSessionDelegate {
2      // ...
3  }
```

Codice 4.2: Estensione al protocollo `NFCNDEFReaderSessionDelegate`

In particolare, il protocollo `NFCNDEFReaderSessionDelegate` consente all'oggetto delegato di ricevere delle notifiche da parte della sessione quando viene letto un messaggio NDEF e quando la sessione viene invalidata, sia per la normale conclusione dell'operazione di lettura sia a causa di un errore. Ogni volta che la sessione riceve un nuovo messaggio NDEF, esso viene inviato al delegato attraverso

il metodo `readerSession(_:didDetectNDEFs:)`. Il delegato, essendo conforme al protocollo, deve prevedere un'implementazione per questa funzione. Nell'esempio (Codice 4.3) i messaggi ricevuti vengono visualizzati a schermo.

```
1 func readerSession(_ session: NFCNDEFReaderSession, didDetectNDEFs messages: [
      NFCNDEFMessage]) {
2     DispatchQueue.main.async {
3         // Process detected NFCNDEFMessage objects.
4         self.detectedMessages.append(contentsOf: messages)
5         self.tableView.reloadData()
6     }
7 }
```

Codice 4.3: Esempio di metodo per gestire la lettura di un messaggio [11]

Quando la sessione si conclude, essa notifica il delegato attraverso il metodo `readerSession(_:didInvalidateWithError:)`. Questa funzione prevede un parametro che indica la ragione per cui la sessione è stata interrotta. Quest'ultima viene invalidata anche a causa della corretta conclusione dell'operazione.

- Un errore di tipo `readerSessionInvalidationErrorFirstNDEFTagRead` viene fornito al delegato nel caso in cui è stata impostata la terminazione della sessione dopo la lettura del primo *tag*.
- Un errore di tipo `readerSessionInvalidationErrorUserCanceled` viene fornito al delegato se, invece, l'utente termina l'operazione oppure l'applicazione richiama la funzione `invalidate()`.

Nell'esempio 4.4, se l'errore ricevuto è diverso dai due appena citati, viene mostrato un *alert* con un messaggio che descrive la motivazione per cui la sessione è terminata (righe 2-23). Se, invece, la sessione termina correttamente, l'errore viene ignorato e si procede direttamente con la de-inizializzazione della sessione (riga 25).

```
1  func readerSession(_ session: NFCNDEFReaderSession, didInvalidateWithError error:
       Error) {
2      if let readerError = error as? NFCReaderError {
3          if (readerError.code != .readerSessionInvalidationErrorFirstNDEFTagRead)
4          && (readerError.code != .readerSessionInvalidationErrorUserCanceled) {
5
6              let alertController = UIAlertController(
7                  title: "Session Invalidated",
8                  message: error.localizedDescription,
9                  preferredStyle: .alert
10             )
11             alertController.addAction(
12                 UIAlertAction(
13                     title: "OK",
14                     style: .default,
15                     handler: nil
16                 )
```

```
17              )
18
19              DispatchQueue.main.async {
20                  self.present(alertController, animated: true)
21              }
22          }
23      }
24
25      self.session = nil
26  }
```

Codice 4.4: Esempio di metodo per gestire una sessione invalidata [11]

4.2.2 ProximityReader

Data la rapida e crescente adozione dei pagamenti *contactless*, Apple ha recentemente messo a disposizione degli sviluppatori un nuovo *framework* basato sulla NFC per uno scopo ben preciso. Il modulo si chiama ProximityReader e nasce a supporto della nuova funzionalità *Tap to Pay on iPhone*. Quest'ultima deriva dall'idea di abilitare gli esercenti ad accettare pagamenti con un semplice tocco sul proprio iPhone. Infatti, attraverso l'utilizzo di ProximityReader, gli sviluppatori sono in grado di integrare all'interno delle proprie applicazioni iOS la possibilità di accettare, direttamente su iPhone e senza l'utilizzo di *hardware* addizionale, pagamenti *contactless* eseguiti mediante carte di credito e di debito abilitate, Apple Pay, Apple Watch e *smartphone* con altri *wallet* digitali [12]. Insomma, ProximityReader abilita l'utilizzo di iPhone come terminale POS. In aggiunta, ProximityReader consente di leggere le carte di pagamento senza effettuare un addebito per memorizzare la carta per pagamenti futuri oppure per effettuare la ricerca di un pagamento precedente. Infine, con *Tap to Pay on iPhone* viene abilitata anche la lettura di carte fedeltà e carte sconto abilitate alla NFC presenti in Apple Wallet, sia indipendentemente che contemporaneamente rispetto alla lettura delle carte di pagamento [12].

Per completezza, si riporta qui di seguito un esempio di codice che consente di avviare la lettura di una carta di pagamento mediante l'utilizzo del *framework* ProximityReader (Codice 4.5).

```
1  public class WrapperClass {
2      var reader: PaymentCardReader?
3      var session: PaymentCardReaderSession?
4
5      public func readCard(for amount: Decimal) async throws {
6          let request = PaymentCardTransactionRequest(
7              amount: amount,
8              currencyCode: "USD",
9              for: .purchase
10         )
11
12         guard let reader = self.reader else { return }
```

```
13          let events = reader.events
14          do {
15              Task {
16                  for await event in events {
17                      // Handle events that happen while the sheet is up.
18                  }
19              }
20              let result = try await session?.readPaymentCard(request)
21              // Send result.paymentCardData to your payment service provider.
22          } catch {
23              // Handle any errors that occur during read
24              // (see PaymentCardReaderSession.ReadError).
25          }
26      }
27  }
```

Codice 4.5: Lettura di una carta di pagamento con ProximityReader [13]

La funzionalità *Tap to Pay on iPhone* è attualmente disponibile solo in alcuni Paesi del mondo e non tutti i fornitori di servizi di pagamento la supportano. In particolare, in Italia non è stata ancora rilasciata. Tuttavia, la recente introduzione della Francia tra la lista di Paesi che supportano il nuovo metodo di accettazione di pagamenti *contactless* fa ben sperare in un rilascio italiano nel futuro prossimo. Inoltre, grandi fornitori di servizi di pagamento come Square, SumUp, Stripe e Revolut hanno già abilitato il supporto a *Tap to Pay on iPhone* all'interno delle loro applicazioni iOS.

4.3 Risultati ottenuti

Combinando lo studio dei *framework* offerti da Apple con le verifiche effettuate durante le fasi di ricerca di questa tesi, si è in grado di esporre i primi risultati sullo sviluppo di nuove esperienze interattive tramite NFC, e più precisamente in merito all'utilizzo di iPhone come terminale POS. A partire dall'applicazione presentata precedentemente, si è, infatti, tentata l'integrazione di una funzionalità che prevedesse la lettura NFC di una carta di pagamento *contactless* mediante CoreNFC. In particolare, a differenza degli esempi di codice riportati, è stata istanziata una sessione di tipo `NFCTagReaderSession`. Se `NFCNDEFReaderSession` consente di leggere esclusivamente *tag* NDEF, l'alternativa ora utilizzata permette di individuare ed interagire con *tag* conformi agli standard ISO/IEC 7816 ed ISO/IEC 14443. Tuttavia, sebbene attraverso l'implementazione è stato possibile individuare la presenza di una *contactless smart card*, una volta riconosciuta essere una carta di pagamento, è stata riscontrata la terminazione inaspettata del processo. La motivazione di questo comportamento viene fornita dalla documentazione ufficiale di CoreNFC. Essa avvisa gli sviluppatori sull'incompatibilità del *framework* con i *tag* associati ad *Application Identifier* (AID) relativi ai metodi di pagamento.

Un AID è un codice che identifica un'applicazione presente all'interno della *smart card*. Attraverso l'invio di uno specifico comando APDU è possibile selezionare l'applicazione desiderata attraverso il corrispettivo AID. Questo comando viene inviato automaticamente da CoreNFC durante l'inizializzazione di una sessione, per ogni AID specificato in fase di configurazione del *framework*. Emerge, dunque, che l'interazione con le *contactless smart card* di pagamento non è supportata dal *framework* CoreNFC.

In questo scenario, ProximityReader assume un ruolo cruciale. Esso nasce proprio con lo scopo di abilitare le applicazioni iOS ad accettare pagamenti *contactless* direttamente su iPhone. Sebbene attraverso l'utilizzo di CoreNFC non sia stato possibile soddisfare tale funzionalità, ProximityReader e *Tap to Pay on iPhone* rappresentano effettivamente uno sviluppo concreto, e già affermato, di nuove esperienze interattive mediante NFC. Infatti, gli esercenti che utilizzano questa funzionalità possono offrire ai propri clienti un'esperienza di pagamento coerente e affidabile. La schermata mostrata ai clienti indica chiaramente l'importo dell'addebito, il nome dell'esercente e l'icona della categoria (Fig. 4.5). Dunque, ProximityReader supera le limitazioni di CoreNFC in tale contesto e, facendo uso della tecnologia NFC, offre un nuovo modo di interagire con il mondo digitale.

Figura 4.5: Richiesta di pagamento *contactless* con *Tap to Pay on iPhone*

Capitolo 5

Documenti d'identità elettronici

Un processo di identificazione elettronica rappresenta una soluzione digitale che ha il compito di identificare univocamente una persona o un'organizzazione. Utilizza dati di autenticazione personale in forma elettronica e assicura il riconoscimento dell'utente elettronico. Processi di questo tipo vengono utilizzati non solo da autorità governative, ma anche da banche ed aziende private, per abilitare l'accesso a particolari benefici o servizi, nonché per firmare un documento elettronico mediante firma digitale. La carta d'identità elettronica rappresenta uno degli strumenti più comuni per l'identificazione elettronica. Un cittadino può utilizzare la propria carta d'identità elettronica come mezzo di identificazione ed autenticazione personale, sia online che offline.

In questo capitolo verrà presentata la carta d'identità elettronica italiana. Attraverso un'analisi delle sue caratteristiche tecniche, nonché tramite lo studio delle raccomandazioni internazionali ICAO sui documenti di viaggio elettronici, sarà possibile affermare che la nuova carta d'identità italiana è una *contactless smart card* abilitata alla lettura NFC. Di conseguenza, quanto esposto in questo capitolo porrà le basi per lo sviluppo di un *framework* in grado di individuare e recuperare le informazioni presenti all'interno del documento tramite tecnologia NFC.

5.1 In Italia

La Carta d'Identità Elettronica (CIE) è il documento d'identità dei cittadini italiani emesso dal Ministero dell'Interno. Esso sostituisce il precedente formato cartaceo ed introduce elevati livelli di sicurezza e meccanismi internazionali di anticlonazione ed anticontraffazione. Permette l'identificazione e l'autenticazione elettronica per l'accesso ai servizi online delle Pubbliche Amministrazioni, sia in

Italia che nei Paesi dell'Unione Europea. Oltre ad accertare l'identità del titolare, la CIE consente di rappresentare il cittadino nel mondo digitale attraverso la sua componente elettronica. Inoltre, la CIE può essere utilizzata come strumento di firma elettronica avanzata, consentendo dunque ai cittadini di firmare digitalmente i propri documenti elettronici [14].

5.1.1 Cenni storici

Prima dell'effettiva adozione della CIE, si presentarono diverse sfide legate a problematiche sia di tipo organizzativo che implementativo. In anticipo rispetto agli altri stati europei, il progetto della CIE era già stato previsto nel 1997 dalle leggi Bassanini sulla riforma della Pubblica Amministrazione e sulla semplificazione amministrativa. Nel 2001, con lo scopo di individuare possibili problemi tecnici, fu avviata la fase di emissione del primo modello sperimentale di CIE in 83 comuni. Nel 2004 fu introdotta la CIE 2.0: un secondo modello, anch'esso sperimentale. Quest'ultima ha fatto da versione pilota in vista della futura emissione su scala nazionale. Dopo due anni, dal 1 gennaio del 2006, si iniziò a sostituire la carta d'identità cartacea con la variante elettronica. Alla fine del 2009 erano stata emesse circa 1.9 milioni di CIE da 153 comuni.

Nonostante ciò, la versione pilota presentava diversi problemi. Paolo Aielli, amministratore delegato dell'istituto Poligrafico e Zecca dello Stato dal 2014 al 2021, racconta: «Non solo i materiali non erano adatti, ma la realizzazione della carta ruotava attorno alla tecnologia proprietaria di un'azienda privata, la Laser Memory Card, nonostante fosse un progetto dello Stato, con tutte le pesanti limitazioni di sicurezza e gestione» [15]. Dunque, alla fine del 2015 si decise di istituire un unico polo produttivo: la Zecca di Roma. In aggiunta, furono specificate le caratteristiche della nuova CIE 3.0 e con il D.L. 78/2015, poi convertito in legge 6 agosto 2015, n. 125, la CIE fu ufficialmente definita come documento d'identità sostitutivo alla versione cartacea. Il 4 luglio 2016, si avviò la fase di sostituzione dal formato cartaceo con quello elettronico, inizialmente operativa solo in 199 comuni. Gradualmente, anche i restanti comuni furono abilitati all'emissione della nuova tessera e nel 2017, ai primi 199, si aggiunsero altri 350 comuni abilitati, assicurando così la copertura del 50% della popolazione. Tutti i comuni italiani furono abilitati entro la metà del 2018 e da settembre 2019 iniziò il rilascio di CIE anche ai cittadini italiani residenti all'estero.

5.1.2 Layout della CIE

Una delle grandi novità tecnologiche introdotte con l'adozione della CIE è la possibilità di poter scansionare la tessera attraverso un dispositivo di riconoscimento ottico dei caratteri (dall'inglese *optical character recognition*, abbreviato OCR).

Alcuni elementi della tessera sono stampati con un carattere che consente ad un lettore ottico di individuare queste informazioni. La tessera viene idealmente suddivisa in due zone: la *Visual Inspection Zone* (VIZ) ha lo scopo di facilitare l'ispezione visiva della tessera, mentre la *Machine Readable Zone* (MRZ) rappresenta i dati personali in modo da poter essere riconosciuti e decodificati da un dispositivo OCR.

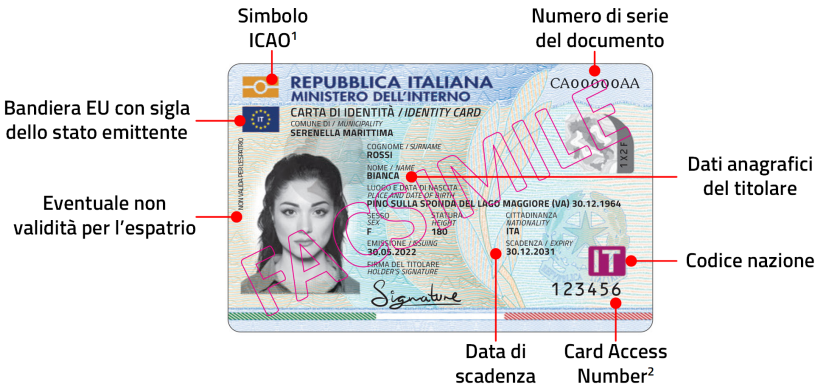

Figura 5.1: Fronte della CIE in uso dal 29 settembre 2022 [16]

La nuova CIE rappresenta i dati personali in una tessera dalle dimensioni di una carta di credito. Sul fronte della tessera (Fig. 5.1) sono riportati i classici dati anagrafici del titolare, la data di scadenza e il numero di serie del documento. Inoltre, sono presenti anche nuovi elementi rispetto al precedente formato cartaceo. Si evidenzia, ad esempio, la presenza del simbolo ICAO e del *Card Access Number* (CAN). Quest'ultimo è «un codice numerico che viene utilizzato dalle autorità di controllo per la lettura dei dati memorizzati nel *microchip*» della CIE.

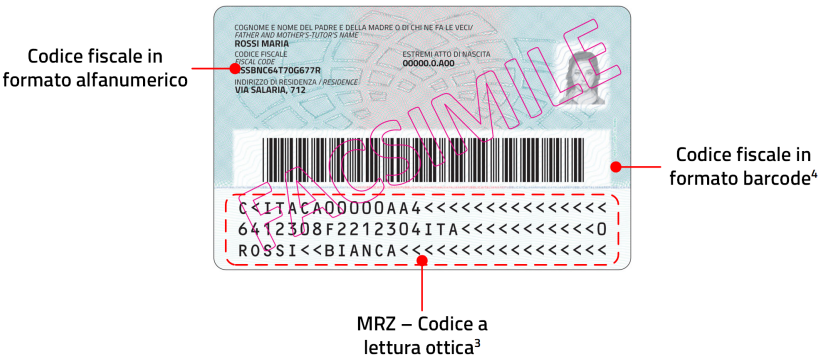

Figura 5.2: Retro della CIE in uso dal 29 settembre 2022 [16]

Sul retro della tessera (Fig. 5.2) vengono rappresentate altre informazioni personali, tra cui il codice fiscale sia in formato alfanumerico che in formato *barcode*. Inoltre, si può osservare la presenza della MRZ. Quest'ultima porzione di tessera «permette di leggere, decodificare e verificare in automatico, con strumenti a lettura ottica (OCR), le informazioni contenute nel documento».

5.1.3 La CIE come smart card

La lettura automatizzata della CIE non avviene esclusivamente mediante l'utilizzo della tecnologia OCR. La tessera è, infatti, dotata di un *microchip*. Esso memorizza al suo interno dati personali e biometrici, come foto e impronte digitali, nonché le informazioni che abilitano le procedure di autenticazione elettronica a servizi online erogati da pubbliche amministrazioni ed imprese. Questa caratteristica rende la tessera uno strumento unico e sicuro per l'identificazione elettronica dell'utente. In aggiunta, l'IC aderisce ai protocolli che consentono la propria lettura attraverso uno *smartphone* con interfaccia NFC, conferendo alla CIE tutte le specifiche di una vera e propria *contactless smart card*. In particolare, la CIE e il suo circuito integrato sono conformi alle raccomandazioni internazionali ICAO Doc 9303. Per questo motivo, è possibile osservare la presenza del simbolo ICAO, anche detto simbolo *Chip Inside*, sul fronte della tessera [16].

5.2 ICAO Doc 9303

Il Ministero degli Affari Esteri e della Cooperazione Internazionale riporta: «L'Organizzazione Internazionale per l'Aviazione Civile (ICAO) è un'agenzia specializzata delle Nazioni Unite con competenza primaria in materia di regolamentazione e sviluppo dell'aviazione civile. L'ICAO, fondata nel 1947, promuove l'elaborazione e l'adozione di norme internazionali e convenzioni in materia di navigazione aerea, trasporto di passeggeri e merci, sicurezza del trasporto aereo» [17].

Nel 1968 l'Organizzazione inizia il lavoro sui *Machine Readable Travel Document* (MRTD), ossia sui documenti di viaggio a lettura ottica[1], per la realizzazione di uno standard internazionale che regolasse le norme per i passaporti e per la loro scansione ottica affinché si accelerasse il processo di controllo dei documenti da parte delle autorità competenti. ICAO pubblica la prima edizione del Doc 9303 nel 1980. Al suo interno vengono raccolte le specifiche e le linee guida per lo sviluppo di un passaporto abilitato alla tecnologia OCR, ponendo le fondamenta per l'emissione

[1]Per MRTD si intende un qualsiasi documento di viaggio con la capacità di essere scansionato mediante lettore ottico. Si includono passaporti, visti e altri documenti di viaggio ufficiali, come le carte d'identità, conformi al Doc 9303 di ICAO.

dei primi *Machine Readable Passport* (MRP) in Australia, Canada e Stati Uniti. In seguito, il documento fu ampliato per includere il materiale di riferimento non solo per i visti, detti *Machine Readable Visa* (MRV), ma anche per quei documenti, come le carte d'identità, che possono essere utilizzati come documenti di viaggio ufficiali, nominati *Machine Readable Official Travel Document* (MROTD). Nel 1998 iniziano le analisi di ICAO per stabilire sia il sistema di identificazione biometrica più efficace sia i relativi mezzi di memorizzazione dei dati. Dunque, tra il 2006 e il 2008 introduce le specifiche per l'utilizzo della tecnologia delle *smart card* senza contatto [18].

5.2.1 Introduzione ai MRTD

Secondo le specifiche ICAO, un qualsiasi MRTD deve garantire la presenza sia della VIZ sia della MRZ, entrambe introdotte in 5.1.2. Più precisamente, la VIZ di un MRTD comprende le informazioni obbligatorie e facoltative pensate per l'ispezione visiva del documento. Essa, attraverso l'applicazione delle convenzioni ICAO sul formato degli elementi, deve contemporaneamente soddisfare sia i requisiti imposti dallo Stato emittente sia un livello sufficientemente alto di uniformità ed interoperabilità globale. D'altra parte, la MRZ facilita ed accelera il processo d'ispezione del documento, oltreché offrire indirettamente una rappresentazione alternativa ed una strategia di verifica della VIZ [19].

Tipologie di MRTD

Data la generica definizione di MRTD, è possibile categorizzare e specificare i documenti di viaggio ICAO secondo le seguenti caratteristiche:

- per MRP si intende generalmente un passaporto conforme alle raccomandazioni ICAO, in particolare ad ICAO Doc 9303-4;
- per MROTD si intende un documento conforme alle raccomandazioni ICAO, in particolare ad ICAO Doc 9303-5 oppure ICAO Doc 9303-6, che può essere utilizzato, in specifici contesti, in sostituzione ad un MRP, come la carta d'identità;
- per MRV si intende un visto conforme alle raccomandazioni ICAO, in particolare ad ICAO Doc 9303-7;
- per *Emergency Travel Document* (ETD) si intende un documento di viaggio conforme alle raccomandazioni ICAO, in particolare ad ICAO Doc 9303-8, che può essere utilizzato in caso di emergenza.

Si specifica che, da qui in avanti, con il termine MRTD si intenderà un documento che risulta essere indifferentemente un MRP oppure un MROTD. Le altre due

tipologie di documenti vanno oltre gli interessi di questa ricerca e pertanto non saranno ulteriormente trattati.

Un MRTD è conforme allo standard internazionale ISO/IEC 7810, precedentemente introdotto in 2.2.3 durante le trattazioni sugli standard tecnici a cui le *smart card* aderiscono. Sulla base di ciò, ICAO definisce tre tipologie di *Travel Document* (TD) a seconda delle dimensioni dello specifico MRTD:

- con TD1 si definisce un MROTD conforme al formato ID-1 di ISO/IEC 7810, le cui dimensioni corrispondo a quelle di una classica carta d'identità elettronica, come la CIE [20];
- con TD2 si definisce un MROTD conforme al formato ID-2 di ISO/IEC 7810, eccetto per lo spessore [21];
- con TD3 si definisce un MRP conforme al formato ID-3 di ISO/IEC 7810, eccetto per lo spessore [22].

Tuttavia, riconoscendo la posizione presa da parte degli Stati di uniformarsi al formato TD1, le specifiche in merito a TD2 non saranno ulteriormente mantenute in futuro, a partire dalla prossima edizione del documento di ICAO [21].

5.2.2 Integrazione della tecnologia *contactless*

Per documento di viaggio elettronico (eMRTD) si intende un MRTD dotato di un circuito integrato *contactless*, capace di memorizzare dati biometrici. L'IC permette di identificare il titolare con un livello di sicurezza e di veridicità elevato in quanto mantiene al suo interno dati biometrici che possono essere utilizzati nei sistemi di riconoscimento facciale, delle impronte digitali e dell'iride. In aggiunta, l'IC memorizza un duplicato della MRZ e lo Stato emittente ha la possibilità di inserire ulteriori informazioni opzionali. In accordo con la documentazione ICAO, le informazioni obbligatoriamente presenti nell'IC includono: la MRZ, i dati biometrici per il riconoscimento facciale e alcuni elementi necessari per garantire la sicurezza dei dati [23].

L'IC è conforme agli standard ISO/IEC 14443 e ISO/IEC 7816-4, entrambi precedentemente introdotti in 2.2.3. Ciò consente di sfruttare effettivamente un eMRTD come una *contacless smart card*, nonché di recuperare le informazioni memorizzate nell'IC tramite tecnologia NFC. D'altra parte, l'introduzione di un IC senza contatto espone gli eMRTD a nuovi rischi e possibili problematiche di sicurezza informatica. Per questo motivo, l'IC implementa tecniche di protezione che non solo garantiscono l'accesso ai dati esclusivamente ad una persona o organizzazione autorizzata, ma allo stesso tempo impediscono la manomissione delle informazioni contenute nel *chip* e permettono la verifica della loro autenticità [23].

Capitolo 6

Sviluppo del *framework* per la lettura NFC di un MRTD

Le analisi condotte nello scorso capitolo consentono di affermare che la CIE rappresenta un'implementazione concreta di eMRTD. Come tale, essa è una *smart card* dotata di IC *contactless*, conforme agli standard ISO/IEC 14443 e ISO/IEC 7816-4, e abilitata alla lettura tramite tecnologia NFC. Queste considerazioni, in combinazione con lo studio della documentazione ICAO, in particolare di ICAO Doc 9303-10 ed ICAO Doc 9303-11, descrivono le basi su cui si pone la realizzazione di un *framework* capace di recuperare le informazioni personali e biometriche memorizzate all'interno del documento d'identità. D'altra parte, questi risultati rappresenteranno gli strumenti necessari per lo sviluppo di nuove esperienze interattive mediante NFC per un'applicazione iOS.

In questo capitolo verrà fornita una panoramica sull'implementazione del *framework* per l'estrazione e la decodifica delle informazioni contenute in un eMRTD. In particolare, dopo aver analizzato il formato di codifica dei dati, verrà affrontata prima un'analisi sulla comunicazione con l'IC e successivamente sulla lettura e sulla decodifica delle informazioni. Infine, si presenteranno i meccanismi di sicurezza incaricati di proteggere l'IC da accessi non autorizzati e da tentativi di sostituzione o manomissione.

6.1 Codifica dei dati

L'analisi della strategia di codifica dei dati di un eMRTD è essenziale per comprendere la struttura delle informazioni estratte dall'IC. Gli elementi nel *chip* sono codificati utilizzando il formato TLV (abbreviazione di *Type Length Value*) specificato dallo standard *Abstract Syntax Notation One* (ASN.1) [24].

6.1.1 Concetti essenziali di ASN.1

Nata con lo scopo di definire uno schema formale e robusto per i messaggi scambiati nei protocolli di comunicazione, ASN.1 è una notazione standardizzata[1] ed internazionale che consente di specificare il formato delle strutture dati a un alto livello di astrazione, indipendentemente dall'implementazione, dalla piattaforma e dal linguaggio di programmazione utilizzato [25]. In ASN.1 vengono definiti sia tipi primitivi in grado di descrivere oggetti semplici, come numeri interi, sia tipi più complessi che permettono di modellare anche strutture dati annidate. Si riportano alcuni esempi qui di seguito:

- `INTEGER` rappresenta valori interi;
- `BIT STRING` modella una sequenza di bit di lunghezza variabile;
- `OCTECT STRING` modella una sequenza di byte;
- `OBJECT IDENTIFIER` definisce un identificatore globale, univoco e registrato da ISO;
- `SEQUENCE` specifica una collezione di elementi;
- `SET` specifica, come il precedente, una collezione di elementi ma, in questo caso, gli oggetti all'interno della lista potrebbero non essere ordinati.

Tramite l'utilizzo dei vincoli, ASN.1 offre la possibilità di limitare l'insieme di possibili valori che un certo dato può assumere e di specificare richieste aggiuntive sulla dimensione o sulla lunghezza di una collezione di elementi.

L'esempio 6.1 mostra come sia effettivamente possibile definire uno schema, anche complesso, in ASN.1 ed evidenzia non solo la possibilità di annidamento delle strutture dati, ma anche la flessibilità data dall'utilizzo dei tipi in combinazione con i vincoli.

```
1 PurchaseOrder ::= SEQUENCE {
2     dateOfOrder DATE,
3     customer    CustomerInfo,
4     items       ListOfItems
5 }
6
7 CustomerInfo ::= SEQUENCE {
8     firstname   VisibleString (SIZE (3..50)),
9     lastname    VisibleString (SIZE (3..50))
```

[1] Per approfondimenti si veda ISO/IEC 8824-1:2021, *Information technology. Abstract Syntax Notation One (ASN.1) – Part 1: Specification of basic notation* oppure le raccomandazioni ITU-T X.680.

```
10 }
11
12 ListOfItems ::= SEQUENCE (SIZE (1..100)) OF Item
13
14 Item ::= SEQUENCE {
15     itemCode         INTEGER (1..99999),
16     color            VisibleString ("Black" | "Blue" | "Brown"),
17     deliveryTime     INTEGER (8..12 | 14..19),
18     quantity         INTEGER (1..1000),
19     unitPrice        REAL (1.00 .. 9999.00),
20     isTaxable        BOOLEAN
21 }
```

Codice 6.1: Esempio di strutture dati definite in ASN.1

Per poter convertire dati ASN.1 in un flusso di byte è necessario definire delle regole che specificano la strategia di codifica dei messaggi per la trasmissione. A seconda dell'ambiente e dal contesto di utilizzo, è possibile scegliere l'insieme di regole più appropriato, le cui caratteristiche possono offrire un risultato più o meno compresso, piuttosto che una strategia di decodifica più veloce [26].

6.1.2 Codificare dati ASN.1 mediante BER

Le BER (*Basic Encoding Rules*)[2] utilizzano il formato TLV per la codifica dei dati, con T, L e V costituiti da una serie di byte [27]: il tipo, o *tag*, T indica la tipologia del dato, la lunghezza L rappresenta la dimensione in byte del dato e il valore V contiene il dato vero e proprio [28]. Quest'ultimo non solo può essere un dato primitivo, come un numero intero o una sequenza di bit, ma può anche rappresentare un dato *constructed*, ossia può a sua volta contenere uno o più dati TLV [28]. Ad esempio, un dato di tipo `SEQUENCE` oppure `SET` è un dato *constructed*.

Figura 6.1: Struttura del formato TLV [28]

[1]Per approfondimenti si veda ISO/IEC 8824-1:2021, *Information technology. ASN.1 encoding rules – Part 1: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)* oppure le raccomandazioni ITU-T X.209 e X.690.

Codifica del tag

Il *tag* T rappresenta un vero e proprio identificatore di codifica e racchiude diverse informazioni particolarmente utili durante la ricostruzione del dato codificato. È possibile dividere il *tag* in tre parti (Fig. 6.2) [29].

- I due bit più significativi indicano la la classe del tipo, dove:
  - 00_2 rappresenta la classe *universal*, assegnata a tutti i tipi definiti nello standard ASN.1;
  - 01_2 rappresenta la classe *application*, che identifica un tipo all'interno di una particolare applicazione;
  - 10_2 rappresenta la classe *context-specific*, che identifica un tipo all'interno di uno specifico contesto ben definito.
  - 11_2 rappresenta la classe *private*, che può essere utilizzata, ad esempio, da un'azienda per identificare un tipo all'interno di alcune delle loro applicazioni comuni;
- Il bit successivo indica la *form* del dato codificato: se esso è 0, allora il tipo è *primitive*, altrimenti il dato è *constructed*.
- I restanti bit indicano il *number*, ossia l'effettivo identificatore dei tipo del dato codificato.

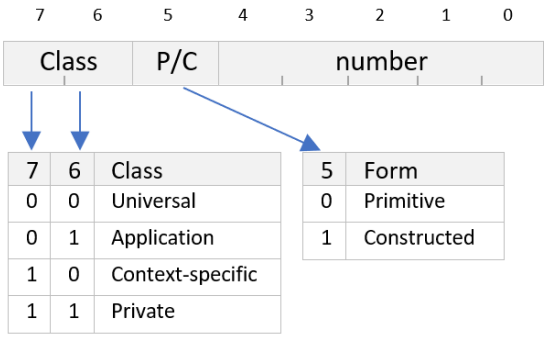

Figura 6.2: Struttura della codifica BER del *tag* [29]

Le seguenti regole permettono la codifica del *number* [28]:

- se *number* è compreso tra 0 e 30 (inclusi), allora il *tag* è codificato in un solo byte e *number* occupa i restanti 5 bit, senza alcuna particolare accortezza (Fig. 6.3);

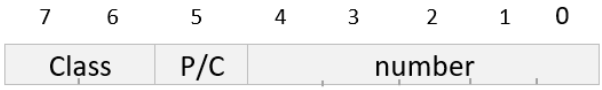

Figura 6.3: Rappresentazione di un *tag* BER codificato in 1 byte [28]

- se *number* è compreso tra 31 e 127 (inclusi), allora il *tag* è codificato in due byte, dove i restanti 5 bit del primo byte sono tutti impostati ad 1, mentre il secondo byte è composto dal bit più significativo impostato a 0, seguito da *number* espresso nei restanti 7 bit (Fig. 6.4);

Figura 6.4: Rappresentazione di un *tag* BER codificato in 2 byte [28]

- se *number* è maggiore di 127, il *tag* è codificato in n byte, dove il primo segue il formato del caso precedente, dal secondo byte all'*n-1*-esimo si imposta il bit più significativo ad 1 e i restanti 7 con una porzione di *number*, in ordine da quella più significativa, e, infine, l'n-esimo byte ha il bit più significativo a 0, seguito dalla porzione meno significativa di *number* (Fig. 6.5).

Figura 6.5: Rappresentazione di un *tag* BER codificato in n byte [28]

Codifica della lunghezza

Per quanto riguarda la codifica della lunghezza L è necessario tenere in considerazione le tre forme di codifica riportate qui di seguito [28].

- *Short form*: se L è compresa tra 0 e 127 byte (inclusi), essa può essere codifica in un solo byte, dove il bit più significativo viene impostato a 0 e i restanti bit rappresentano L (Fig. 6.6).

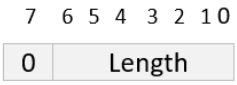

Figura 6.6: Codifica BER della lunghezza in *short form* [28]

- *Long form*: se L è compresa tra 0 e 2^{1008} byte (inclusi), essa può essere codificata in una serie di byte, dove il primo ha il bit più significativo impostato ad 1 seguito dalla lunghezza in byte di L, mentre i restanti n byte rappresentano L (Fig. 6.7).

Figura 6.7: Codifica BER della lunghezza in *long form* [28]

- *Indefinite form*: se in fase di codifica non si conosce l'effettiva lunghezza in byte del dato, o semplicemente se si ritiene conveniente tracciare l'inizio e la fine del dato codificato invece di specificarne l'esatta dimensione, è possibile racchiudere V tra un byte contenente il valore esadecimale `0x80` (10000000_2) e due byte speciali di *end-of-content*, entrambi rappresentanti il valore `0x00` (Fig. 6.8).

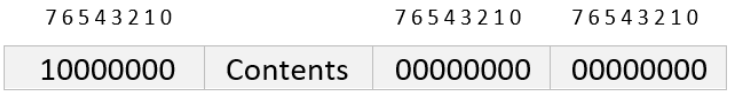

Figura 6.8: Codifica BER della lunghezza in *indefinite form* [28]

6.1.3 Sottoinsiemi di BER

Le regole descritte finora costituiscono una vista d'insieme sufficientemente dettagliata per comprendere i meccanismi fondamentali della codifica dei dati ASN.1

mediante BER. Tuttavia, sono stati definiti anche dei sottoinsiemi delle regole BER. In particolare, le *Canonical Encoding Rules* (CER) e le *Distinguished Encoding Rules* (DER) rappresentano due specializzazioni delle BER, che eliminano parte della flessibilità fornita da quest'ultime. Di conseguenza, un *decoder* BER è in grado di decodificare correttamente una qualsiasi codifica CER o DER, ma non è possibile il viceversa [27].

Le DER garantiscono che per ogni messaggio esista una ed una sola codifica possibile. Questo insieme di regole viene spesso utilizzato nelle applicazioni relative alla sicurezza dei sistemi informativi, come i certificati digitali X.509 [30]. Tra le restrizioni applicate dalle DER si evidenzia il vincolo per cui la lunghezza L deve essere codificata nel minor numero di byte e in una *definite form*, negando dunque l'utilizzo della *indefinite form* [27]. Al contrario, le CER consentono la codifica della lunghezza solo nella *indefinite form* [27].

6.1.4 Sviluppo del modulo ASN.1

Ponendo nuovamente l'attenzione sul *framework*, dalle analisi svolte durante gli sviluppi è emerso che le informazioni memorizzate nell'IC sono codificate in formato TLV e che le regole di codifica utilizzate coincidono con le DER. Pertanto, a supporto della decodifica dei dati, è stato sviluppato un modulo in grado di riconoscere correttamente il formato TLV delle strutture dati presenti nell'IC e di identificarne i corrispettivi campi[3].

Per modellare un singolo elemento TLV è stata implementata la struttura `ASN1Node` (Codice 6.2). Essa mantiene al suo interno le informazioni che caratterizzano un nodo TLV, nonché il flusso di byte originale.

```
1  internal struct ASN1Node {
2      private(set) var tag: ASN1Tag
3      private(set) var content: Content
4      private(set) var encodedBytes: ArraySlice<UInt8>
5
6      internal var children: ASN1NodeCollection? {
7          if case .constructed(let children) = content {
8              return children
9          } else { return nil }
10     }
11
12     internal var length: Int {
13         switch content {
14         case .constructed(let nodes): nodes.reduce(0) { (prev, node) in
```

[3]La decodifica vera e propria delle strutture dati in informazioni di senso compiuto sarà descritta successivamente. Il modulo qui presentato ha il compito di effettuare correttamente il *parsing* dei dati in formato TLV, in modo tale da suddividere con successo un flusso di byte recuperato dall'IC.

```
15              prev + node.length
16          }
17          case .primitive(let data): data.count
18          }
19      }
20
21      internal init(tag: ASN1Tag, content: Content, encodedBytes: ArraySlice<UInt8>)
         {
22          self.tag = tag
23          self.content = content
24          self.encodedBytes = encodedBytes
25      }
26
27      // ...
28  }
```

Codice 6.2: Implementazione della struttura `ASN1Node`

Come specificato in precedenza, un dato TLV può contenere a sua volta un'ulteriore informazione in formato TLV. Tale comportamento è stato implementato attraverso il tipo enumerazione `ASN1Node.Content` (Codice 6.3). In questo modo è possibile mantenere nel contenuto del nodo corrente un dato *primitive* oppure *constructed*. In quest'ultimo caso, il contenuto comprende una collezione di altri nodi.

```
1  internal struct ASN1Node {
2
3      // ...
4
5      internal enum Content {
6          case constructed(ASN1NodeCollection)
7          case primitive(ArraySlice<UInt8>)
8      }
9  }
```

Codice 6.3: Implementazione del tipo enumerazione `ASN1Node.Content`

Essendo il *tag* di un nodo TLV particolarmente importante per la corretta comprensione e decodifica del flusso di byte recuperato, per modellare efficientemente il tipo di un dato TLV è stata sviluppata la struttura dati `ASN1Tag` (Codice 6.4). Inoltre, a supporto di essa sono stati implementati i tipi enumerazione `ASN1TagClass` e `ASN1TagForm` (Codice 6.5).

```
1  internal final class ASN1Tag: ExpressibleByIntegerLiteral {
2      private(set) var rawValue: Int
3
4      internal var `class`: ASN1TagClass {
5          // Initialize ASN.1 tag class...
6      }
7
8      internal var form: ASN1TagForm {
9          // Initialize ASN.1 tag form...
10     }
```

```
11
12      internal var number: Int {
13          self.rawValue & 0x001F
14      }
15
16      // ...
17  }
```

Codice 6.4: Implementazione della classe `ASN1Tag`

```
1   enum ASN1TagClass: UInt8, Hashable, Sendable {
2       case universal = 0x00
3       case application = 0x01
4       case context = 0x02
5       case `private` = 0x03
6
7       init(tag: UInt8) {
8           switch tag >> 6 {
9           case 0x00: self = .universal
10          case 0x01: self = .application
11          case 0x02: self = .context
12          case 0x03: self = .private
13          default: fatalError("Unreachable")
14          }
15      }
16
17      init(tag: [UInt8]) {
18          self.init(tag: tag[0])
19      }
20  }
21
22  enum ASN1TagForm: Hashable, Sendable {
23      case primitive
24      case constructed
25
26      init(tag: UInt8) {
27          switch (tag >> 5) & 0x01 {
28          case 0x00: self = .primitive
29          case 0x01: self = .constructed
30          default: fatalError("Unreachable")
31          }
32      }
33
34      init(tag: [UInt8]) {
35          self.init(tag: tag[0])
36      }
37  }
```

Codice 6.5: Tipi enumerazione di supporto `ASN1TagClass` e `ASN1TagForm`

Il componente incaricato di effettuare il *parsing* di un flusso di byte è implementato attraverso `ASN1Parser`. Esso espone il metodo `parse(_:)` (Codice 6.6) che consente di analizzare un flusso di byte recuperato dall'IC e di produrre un `ASN1Node` come risultato del *parsing*.

```swift
internal struct ASN1Parser {

    // ...

    internal static func parse(_ data: ArraySlice<UInt8>) throws -> ASN1Node {
        var result = try ASN1ParseResult.parse(data)
        let firstNode = result.nodes.removeFirst()
        let rootNode: ASN1Node
        let content: ASN1Node.Content

        if firstNode.isConstructed {
            let nodeCollection = result.nodes.prefix {
                $0.depth > firstNode.depth
            }
            result.nodes = result.nodes.dropFirst(nodeCollection.count)
            content = .constructed(ASN1NodeCollection(
                nodes: nodeCollection, depth: firstNode.depth)
            )
        } else { content = .primitive(firstNode.dataBytes ?? []) }

        rootNode = ASN1Node(
            tag: firstNode.tag,
            content: content,
            encodedBytes: firstNode.encodedBytes
        )

        precondition(
            result.nodes.count == 0,
            "ASN1ParseResult unexpectedly allowed multiple root nodes"
        )

        return rootNode
    }
}
```

Codice 6.6: Metodo `parse(_:)` di `ASN1Parser`

L'effettiva logica di analisi del flusso di byte codificato è implementata dalla funzione `parse(_:)` di `ASN1ParseResult`, di cui si omette l'implementazione. Tale metodo costruisce un risultato di tipo `ASN1ParseResult`. Esso mantiene al suo interno una collezione di `ASN1NodeParser`, ossia un'astrazione più semplice ed essenziale di `ASN1Node`, il cui utilizzo è vantaggioso durante la sola fase di *parsing*. Di conseguenza, `ANS1Parser` richiama questa procedura di analisi con il successivo compito di rappresentarne il risultato attraverso il modello più sofisticato specificato da `ASN1Node`.

6.2 Comunicazione con l'IC

Essendo conforme allo standard ISO/IEC 14443, un eMRTD implementa la tecnologia delle *proximity smart card*. Da ciò deriva la compatibilità con gli standard NFC, nonché l'integrazione delle interfacce che ne permettono la lettura attraverso sia lettori compatibili che *smartphone* con supporto alla NFC [31].

6.2.1 Inizializzazione della sessione NFC

Il punto di ingresso del *framework* è stato implementato attraverso il metodo `readPassport(mrzKey:)` di `NFCPassportReader` (Codice 6.7)[4].

```
1  public final class NFCPassportReader: NSObject {
2      private typealias NFCCheckContinuation =
3          CheckedContinuation<NFCPassportModel, Error>
4
5      private var mrzKey: String?
6      private var readerSession: NFCTagReaderSession?
7
8      private var continuation: NFCCheckContinuation?
9      private var passport: NFCPassportModel?
10
11     // ...
12
13     public func readPassport(mrzKey: String) async throws -> NFCPassportModel {
14         self.passport = NFCPassportModel()
15         self.mrzKey = mrzKey
16
17         guard NFCNDEFReaderSession.readingAvailable else {
18             throw NFCPassportReaderError.NFCNotSupported
19         }
20
21         readerSession = NFCTagReaderSession(
22             pollingOption: .iso14443,
23             delegate: self,
24             queue: nil
25         )
26         self.updateReaderSessionMessage(alertMessage: .requestPresentPassport)
27         readerSession?.begin()
28
29         return try await withCheckedThrowingContinuation {
30             (continuation: NFCCheckContinuation) in
31             self.continuation = continuation
32         }
33     }
34 }
```

Codice 6.7: Metodo `readPassport(mrzKey:)` di `NFCPassportReader`

Il metodo ha il compito di inizializzare una sessione NFC attraverso la quale effettuare la lettura dell'IC. Il risultato che viene prodotto è un modello che rappresenta le informazioni recuperate. A riga 21 si può notare l'utilizzo di CoreNFC per la creazione di un'istanza di `NFCTagReaderSession`. Essa è configurata, tramite il parametro `pollingOption`, per interagire con un *tag* ISO/IEC 14443. A riga 27

[4]Per il momento, si tralasci l'argomento passato in ingresso alla funzione. Il suo scopo sarà maggiormente comprensibile compresi i meccanismi di sicurezza per l'accesso all'IC, descritti in 6.4.1.

avviene l'effettivo inizio della sessione di lettura. Rilevato l'IC, viene invocata la funzione `tagReaderSession(_:didDetect:)` (Codice 6.8).

```
extension NFCPassportReader: NFCTagReaderSessionDelegate {

    // ...

    public func tagReaderSession(
        _ session: NFCTagReaderSession,
        didDetect tags: [NFCTag]
    ) {
        if tags.count > 1 {
            self.invalidateSession(error: .MoreThanOneTagFound)
            return
        }

        let passportTag: NFCISO7816Tag
        if case .iso7816(let nFCISO7816Tag) = tags.first {
            passportTag = nFCISO7816Tag
        } else {
            self.invalidateSession(error: .TagNotValid)
            return
        }

        Task { [passportTag] in
            do {
                try await session.connect(to: tags.first!)
                // ...
                let tagReader = TagReader(tag: passportTag)
                // ...
                let passportModel = try await self.startReading(
                    tagReader: tagReader
                )
                continuation?.resume(returning: passportModel!)
                continuation = nil
            } catch let error as NFCPassportReaderError {
                self.invalidateSession(error: error)
            } catch let error {
                self.invalidateSession(error: .ConnectionError)
                continuation?.resume(throwing: error)
                continuation = nil
            }
        }
    }

    // ...
}
```

Codice 6.8: Metodo `tagReaderSession(_:didDetect:)` di `NFCPassportReader`

L'implementazione individua un *tag* di tipo `NFCISO7816Tag`, un'interfaccia fornita da CoreNFC per interagire con *tag* conformi allo standard ISO/IEC 7816. Stabilita una connessione con quest'ultimo, avvia la lettura dei dati memorizzati nell'IC. La procedura di lettura delle informazioni, nonché la produzione della struttura dati `NFCPassportModel` risultante, avviene attraverso il metodo `startReading(tagReader:)`. Quest'ultimo riceve come argomento un oggetto di

tipo `TagReader`, a cui viene associato, in fase di inizializzazione, il *tag* ISO/IEC 7816.

6.2.2 Comandi APDU

La comunicazione con l'IC, essendo esso conforme ad ISO/IEC 7816-4, avviene mediante lo scambio di messaggi APDU, come discusso in 2.2.3. Il primo comando APDU viene automaticamente inviato da CoreNFC durante l'inizializzazione della sessione di lettura NFC, come già riportato in 4.3. In particolare, quando la sessione riconosce il *tag*, essa invia un comando `SELECT`. Se si riceve una risposta positiva, viene invocata la funzione `tagReaderSession(_:didDetect:)`, vista sopra. In questo caso, l'AID che consente di selezionare l'applicazione eMRTD di interesse per il *framework* è `A00000024710`01[5], registrato da ISO secondo le specifiche in ISO/IEC 7816-5.

Il componente responsabile della successiva comunicazione con l'IC e del corretto scambio di messaggi con il *tag* viene individuato in `TagReader`. Esso espone il metodo `send(cmd:)`, attraverso il quale è possibile inviare un comando APDU al *tag* (Codice 6.9)[6].

```
1  internal final class TagReader {
2      private var tag: NFCISO7816Tag
3      internal var secureSession: NFCSecureSession
4
5      // ...
6
7      internal func send(cmd: NFCISO7816APDU) async throws -> APDUResponse {
8          var command = cmd
9
10         if secureSession.isSecureSessionEstablished {
11             command = try secureSession
12                 .secureMessaging!
13                 .protect(apdu: cmd)
14         }
15
16         let (data, sw1, sw2) = try await tag.sendCommand(apdu: command)
17         var response = APDUResponse(data: [UInt8](data), sw1: sw1, sw2: sw2)
18
19         if secureSession.isSecureSessionEstablished {
20             response = try secureSession
21                 .secureMessaging!
22                 .unprotect(rapdu: response)
23         }
24
```

[5]Questo AID permette di selezionare l'applicazione LDS1 di un eMRTD. Ulteriori informazioni in merito ad LDS1 verranno esposte in 6.3.

[6]I meccanismi di *secure session* e di protezione dei messaggi riportati nel codice saranno discussi in 6.4.2.

```
25          if let error = response.error {
26              throw NFCPassportReaderError.ResponseError(
27                  error: error,
28                  reason: error.description,
29                  sw1: BytesRepresentationConverter
30                      .convertToHexRepresentation(from: response.sw1),
31                  sw2: BytesRepresentationConverter
32                      .convertToHexRepresentation(from: response.sw2)
33              )
34          }
35
36          return response
37      }
38
39      // ...
40  }
```

Codice 6.9: Metodo `send(cmd:)` di `TagReader`

Qui si vuole evidenziare l'utilizzo della funzione `sendCommand(apdu:)` esposta da `NFCISO7816Tag` ed utilizzata a riga 16. Tale procedura di CoreNFC permette di inviare un comando APDU, modellato da `NFCISO7816APDU`, e ricevere una risposta APDU. Ovviamente, sia il comando che la risposta seguono rispettivamente le strutture riportate nelle tabelle 2.1 e 2.2.

Sebbene il metodo `send(cmd:)` di `TagReader` non abbia visibilità privata, effettivamente esso non viene richiamato dai componenti diversi da `TagReader`. A seconda del comando APDU da inviare è, infatti, possibile richiamare una specifica funzione esposta da `TagReader`, la quale non solo invoca la procedura descritta finora ma spesso esegue ulteriori operazioni, preliminari o di elaborazione successiva. Un semplice esempio è dato dal metodo `selectPassportApplication()` (Codice 6.10)[7].

```
1   internal extension TagReader {
2       // ...
3
4       func selectPassportApplication() async throws -> APDUResponse {
5           try await send(cmd: APDUCommand.SELECT_PASSPORT_APPLICATION)
6       }
7
8       // ...
9   }
```

Codice 6.10: Metodo `selectPassportApplication()` di `TagReader`

Per concludere, si vogliono introdurre le implementazioni di `APDUCommand`, oltreché `APDUResponse`. La prima è una struttura di utilità che racchiude la definizione

[7]Il comando inviato attraverso questo metodo coincide con il comando `SELECT` inviato da CoreNFC per selezionare l'applicazione eMRTD tramite AID.

di tutti i comandi APDU utilizzati dal *framework* affinché si possano agevolmente richiamare quando necessario(Codice 6.11).

```
internal struct APDUCommand {
    private static let instructionClass: UInt8 = 0x00
    private static let commandChainingInstructionClass: UInt8 = 0x10

    // ...

    internal static var SELECT_PASSPORT_APPLICATION: NFCISO7816APDU {
        .init(
            instructionClass: 0x00,
            instructionCode: 0xA4,
            p1Parameter: 0x04,
            p2Parameter: 0x0C,
            data: Data([0xA0, 0x00, 0x00, 0x02, 0x47, 0x10, 0x01]),
            expectedResponseLength: -1
        )
    }

    // ...
}
```

Codice 6.11: Esempio di comando APDU definito in `APDUCommand`

D'altra parte, la struttura `APDUResponse` permette di accedere efficientemente ad una risposta APDU. A supporto di ciò, sono state implementate delle *shortcut* per verificare il successo del comando o, in alternativa, per decodificare il codice di errore restituito in un messaggio descrittivo (Codice 6.12).

```
internal struct APDUResponse {
    private(set) var data: [UInt8]
    private(set) var sw1: UInt8
    private(set) var sw2: UInt8

    internal var isSuccess: Bool { [sw1, sw2] == [0x90, 0x00] }

    internal var isError: Bool { !self.isSuccess }

    internal var error: APDUResponseError? {
        if self.isError {
            return APDUResponseErrorDecoder.decode(response: self)
        } else { return nil }
    }

    internal func discardResponse() {}
}
```

Codice 6.12: Implementazione della struttura `APDUResponse`

6.3 Lettura e decodifica dei dati

Attraverso determinati comandi APDU è possibile richiedere informazioni all'IC. Esso risponderà con i dati codificati secondo le regole viste in precedenza. Successivamente, quest'ultimi dovranno essere decodificati affinché possano rappresentare un'informazione utile. Per recuperare i dati desiderati è innanzitutto necessario conoscere la loro posizione all'interno dell'IC. Infatti, esso implementa effettivamente una sorta di *file system*. La sua struttura prevede la presenza di un *Master File* (MF) come radice del sistema, al cui interno si trovano eventuali *Elementary File* (EF) e le applicazioni che sono state installate dallo Stato emittente. Secondo le raccomandazioni ICAO, un eMRTD deve supportare almeno l'applicazione LDS1, la quale definisce la struttura logica dei dati (*Logical Data Structure*, abbreviato LDS) conservati nell'IC [24]. In particolare, qui di seguito sono riportate le applicazioni che possono essere installate su un eMRTD.

- L'applicazione LDS1 è obbligatoria e memorizza i dati registrati dallo Stato emittente in 16 gruppi, chiamati *Data Group* (DG). In LDS1 viene conservato anche il *Document Security Object* (SOD). Quest'ultimo mantiene una serie di dati necessari per soddisfare i requisiti di sicurezza imposti dall'IC.
- In aggiunta, un eMRTD può ulteriormente implementare l'applicazione LDS2. Essa comprende funzionalità specifiche per la memorizzazione dei dati inerenti a viaggi, visti elettronici ed informazioni biometriche addizionali.
- Infine, ogni Stato emittente potrebbe voler aggiungere ulteriori applicazioni *ad hoc* per specifiche funzionalità.

Il *framework* sviluppato è in grado di recuperare informazioni esclusivamente dall'applicazione LDS1, selezionata attraverso il comando `SELECT` e il suo AID (`A0000002471001`), come riportato in 6.2.2.

Gli EF che si possono trovare sotto la radice MF sono riportati qui di seguito.

- EF.ART/INFO è richiesto se è installata un'applicazione LDS2, altrimenti opzionale. Contiene specifiche informazioni implementative, come il massimo numero di byte per un comando APDU oppure il numero massimo di byte che ci si può aspettare da una risposta APDU.
- EF.DIR, anche questo richiesto se è presente una qualsiasi applicazione LDS2, ma comunque raccomandato se sono presenti altre applicazioni oltre LDS1. Esso mantiene la lista delle applicazioni supportate dall'eMRTD.
- EF.CardAccess è richiesto se l'IC supporta il meccanismo di sicurezza PACE per il controllo dell'accesso. Contiene le relative informazioni di sicurezza e i parametri crittografici utilizzati.

- EF.CardSecurity contiene le informazioni di sicurezza e i parametri crittografici per i meccanismi di sicurezza *Terminal Authenitcation* e *Chip Authentication*, se supportati, nonché gli stessi dati contenuti in EF.CardAccess;

6.3.1 Struttura dei dati in LDS1

L'applicazione LDS1 offre all'IC uno spazio di memorizzazione per elementi di dati obbligatori e facoltativi, firmati digitalmente dallo Stato emittente ed utilizzati per associare il documento al relativo titolare. Per garantire la protezione delle informazioni personali e per individuare facilmente eventuali manomissioni del circuito integrato, solo lo Stato emittente ha accesso in scrittura ai gruppi di dati. Di conseguenza, prima dell'emissione del documento, l'IC viene bloccato, impedendo così la scrittura, la modifica o la cancellazione dei dati presenti in LDS1 [24]. Qui di seguito una rassegna dei gruppi di dati che è possibile recuperare in LDS1.

- EF.COM, obbligatoriamente presente, contiene le informazioni sulla versione di LDS1, sulla versione di Unicode[8] e la lista dei DG presenti all'interno dell'applicazione.
- EF.SOD, obbligatoriamente presente, è firmato digitalmente all'emissione del documento e, in particolare, contiene i *digest*[9] dei DG presenti in LDS1.
- EF.DG1, obbligatoriamente presente, riflette l'intero contenuto della MRZ, i cui dettagli implementativi dipendono dal formato di eMRTD (TD1, TD2 oppure TD3).
- EF.DG2, obbligatoriamente presente, mantiene la codifica dell'immagine del volto del titolare per essere utilizzata nei sistemi di riconoscimento facciale.
- EF.DG3, se presente, memorizza l'impronta digitale del titolare, la quale può essere utilizzata nei corrispettivi sistemi di riconoscimento biometrico, mentre EF.DG4, se presente, contiene i dati biometrici per il riconoscimento dell'iride.
- EF.DG5 ed EF.DG7, se presenti, mantengono rispettivamente le codifiche del ritratto e della firma del titolare visualizzati sul documento, mentre EF.DG6 è riservato per usi futuri.

[8]Unicode è un sistema di codifica che assegna un numero univoco ad ogni carattere usato per la scrittura di testi, in maniera indipendente dalla lingua, dalla piattaforma informatica e dal programma utilizzato.

[9]Una funzione di hash produce una sequenza di bit detta *digest*, strettamente correlata con i dati in ingresso. Il *digest* di un dato è una rappresentazione unica e compatta delle informazioni originali, nonché la cosiddetta impronta digitale dell'informazione.

- EF.DG8, EF.DG9 ed EF.DG10, se presenti, memorizzano ulteriori informazioni per specifiche funzionalità, non di interesse ai fini di questa tesi.
- EF.DG11 ed EF.DG12, se presenti, contengono rispettivamente ulteriori informazioni personali, come il codice fiscale e l'indirizzo di residenza, e dettagli aggiuntivi sul documento, ad esempio l'autorità emittente e la data di emissione.
- EF.DG13, se presente, è utilizzato per memorizzare ulteriori dettagli a discrezione dello Stato emittente.
- EF.DG14, richiesto se l'IC implementa il meccanismo di sicurezza *Chip Authentication* o la strategia di accesso sicuro PACE, ed EF.DG15, richiesto se è implementato il meccanismo *Active Authentication*, contengono le informazioni di sicurezza necessarie per il corrispettivo meccanismo di protezione.
- EF.DG16, se presente, contiene una lista di informazioni sui contatti di emergenza comunicati dal titolare.

La struttura interna dei dati implementata da un eMRTD con LDS1 può essere rappresentata come in Fig. 6.9.

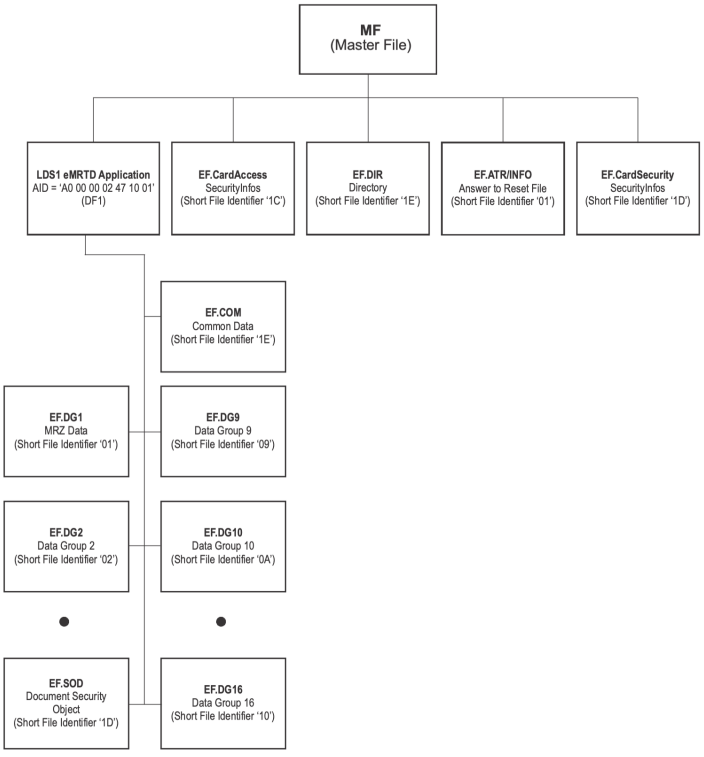

Figura 6.9: Struttura e contenuto di un eMRTD con LDS1 [24]

6.3.2 Implementazione del processo di decodifica

Il processo di lettura e decodifica delle informazioni presenti all'interno dell'IC ha inizio attraverso il metodo `startReading(tagReader:)` di `NFCPassportReader` (Codice 6.13)[10].

```
private extension NFCPassportReader {
    private func startReading(tagReader: TagReader) async throws ->
    NFCPassportModel? {
        let cardAccessData = try await tagReader.readCardAccess()
        let cardAccess = try CardAccess(ASN1Parser.parse(cardAccessData))
        passport?.cardAccess = cardAccess

        try await self.doPACEAuthentication(tagReader: tagReader)
        try await tagReader.selectPassportApplication().discardResponse()
        if passport?.PACEStatus != .success {
            try await doBACAuthentication(tagReader: tagReader)
        }

        try await readDataGroups(tagReader: tagReader)
        // ...
        readerSession?.invalidate()
        return self.passport
    }

    // ...
}
```

Codice 6.13: Metodo `startReading(tagReader:)` di `NFCPassportReader`

Il metodo `readDataGroups(tagReader:)` ha il compito di orchestrare la lettura sistematica dei DG presenti nel circuito integrato (Codice 6.14).

```
private extension NFCPassportReader {
  // ...

  private func readDataGroups(tagReader: TagReader) async throws {
    if let com = try await readDataGroup(tagReader: tagReader, dgTag: .COM)
    as? COM {
        self.passport?.addDataGroup(.COM, dataGroup: com)

        if com.availableDataGroups.contains(.DG14),
           let dg14 = try await readDataGroup(tagReader: tagReader, dgTag: .DG14)
           as? DataGroup14 {
            self.passport?.addDataGroup(.DG14, dataGroup: dg14)
            // ...
            try await doChipAuthentication(tagReader: tagReader, dg14: dg14)
        } else { self.passport?.CAStatus = .notSupported }

```

[10]Si tralascino le operazioni preliminari di lettura di EF.CardAccess e di esecuzione dei meccanismi di autenticazione. Maggiori dettagli riguardanti l'accesso all'IC saranno forniti in 6.4.1.

```
17          for dgTag in com.availableDataGroups.filter({ $0 != .DG14 }) {
18              if let dataGroup = try await readDataGroup(tagReader: tagReader,
19              dgTag: dgTag) {
20                  self.passport?.addDataGroup(dgTag, dataGroup: dataGroup)
21              }
22          }
23
24          if let sod = try await readDataGroup(tagReader: tagReader, dgTag: .SOD)
25          as? SOD {
26              self.passport?.addDataGroup(.SOD, dataGroup: sod)
27              try doPassiveAuthentication(sod: sod)
28          }
29      }
30    }
31
32    // ...
33  }
```

Codice 6.14: Metodo `readDataGroups(tagReader:)` di `NFCPassportReader`

Il primo gruppo di dati recuperato è EF.COM. Attraverso la sua lettura è possibile ottenere la lista dei DG presenti nell'IC. Se presente, viene letto EF.DG14, affinché si possano eseguire le corrispettive operazioni di sicurezza, le quali saranno discusse in 6.4.4. Successivamente, si esegue la lettura di tutti gli altri DG presenti nell'IC. Infine, viene recuperato EF.SOD per verificare la correttezza e l'integrità dei dati recuperati, come si vedrà in 6.4.3.

Per la lettura di ogni singolo gruppo di dati viene richiamata la funzione `readDataGroup(tagReader:dgTag:)` (Codice 6.15).

```
1   private extension NFCPassportReader {
2       // ...
3
4       private func readDataGroup(tagReader: TagReader, dgTag: DGTag) async throws ->
         DataGroup? {
5           // ...
6           let response = try await tagReader.readDataGroup(dgTag)
7           return try DGDecoder.decode(data: response)
8           // ...
9       }
10
11      // ...
12  }
```

Codice 6.15: Metodo `readDataGroup(tagReader:dgTag:)` di `NFCPassportReader`

Essa è incaricata di richiamare `TagReader` per comunicare con l'IC attraverso i relativi comandi APDU e recuperare un flusso di byte dall'IC (Codice 6.16).

```
1   internal extension TagReader {
2       // ...
```

```
3
4       func readDataGroup(_ dataGroup: DGTag?) async throws -> [UInt8] {
5           guard let fileId = dataGroup?.EFIdentifier else {
6               throw NFCPassportReaderError.UnsupportedDataGroup
7           }
8
9           try await selectFile(file: fileId).discardResponse()
10          return try await readFile()
11      }
12
13      // ...
14  }
```

Codice 6.16: Metodo `readDataGroup(tagReader:dgTag:)` di `TagReader`

Per selezionare il corretto EF in LDS1, viene inviata una variante del comando APDU `SELECT`. Ogni DG, oltre ad essere associato ad un *tag* ASN.1 utile in fase di decodifica, è identificato da un codice, detto *EF Identifier*, e dalla sua versione ridotta *Short EF Identifier*. Di conseguenza, al posto di un AID, in questo caso, viene inviato l'identificatore dell'EF. Tale informazione è mantenuta dal tipo enumerazione `DGTag`.

Selezionato quindi l'EF desiderato, si procede con l'effettiva lettura del DG. Il `TagReader` invia uno o più comandi APDU `READ BINARY` all'IC a seconda della dimensione totale del gruppo di dati. Di conseguenza, l'IC risponde, ad ogni comando, con un flusso di byte (Codice 6.17).

```
1   internal extension TagReader {
2       // ...
3
4       func readFile() async throws -> [UInt8] {
5           // ...
6           while remaining > 0 {
7               // ...
8               let cmd = APDUCommand.READ_BINARY(
9                   offset: offset,
10                  expectedResponseLength: readAmount
11              )
12              response = try await send(cmd: cmd)
13
14              data += response.data
15              remaining -= response.data.count
16              amountRead += response.data.count
17          }
18
19          return data
20      }
21  }
```

Codice 6.17: Metodo `readFile()` di `TagReader`

Successivamente, `TagReader` ritorna il risultato prodotto dalla lettura del DG a `NFCPassportReader`. Esso ne avvia la decodifica tramite il metodo `decode(data:)` di `DGDecoder` (Codice 6.18).

```
internal final class DGDecoder {
    private static let classes: [DGTag:DataGroup.Type] = [
        .COM: COM.self,
        .DG1: DataGroup1.self,
        .DG2: DataGroup2.self,
        .DG3: DGNotImplemented.self, // DG3 Not Implemented
        .DG4: DGNotImplemented.self, // DG4 Not Implemented
        .DG5: DGNotImplemented.self, // DG5 Not Implemented
        .DG6: DGNotImplemented.self, // DG6 Not Implemented
        .DG7: DataGroup7.self,
        .DG8: DGNotImplemented.self, // DG8 Not Implemented
        .DG9: DGNotImplemented.self, // DG9 Not Implemented
        .DG10: DGNotImplemented.self, // DG10 Not Implemented
        .DG11: DataGroup11.self,
        .DG12: DataGroup12.self,
        .DG13: DGNotImplemented.self, // DG13 Not Implemented
        .DG14: DataGroup14.self,
        .DG15: DGNotImplemented.self, // DG15 Not Implemented
        .DG16: DGNotImplemented.self, // DG16 Not Implemented
        .SOD: SOD.self
    ]

    internal static func decode(data: [UInt8]) throws -> DataGroup {
        let data = try ASN1Parser.parse(data)

        guard let tag = DGTag.decode(from: data.tag),
              let dataGroup = DGDecoder.classes[tag] else {
            throw NFCPassportReaderError.UnknownTag
        }

        return try dataGroup.init(data, identifier: tag)
    }
}
```

Codice 6.18: Implementazione della classe `DGDecoder`

Il componente `DGDecoder` mantiene la mappatura di tutti i possibili `DGTag` nella corrispettiva classe che implementa lo specifico DG[11]. Il metodo `decode(data:)` sfrutta `ASN1Parser`, descritto in 6.1.4, per eseguire il *parsing* del flusso di byte ricevuto in ingresso. Identificata la classe attraverso il relativo *tag*, si inizializza il gruppo di dati. Ogni specifica implementazione di un DG estende la classe astratta `DataGroup`. Essa garantisce la memorizzazione del *tag* e della struttura ASN.1 dell'elemento, nonché la presenza di un metodo `decode(_:)`. Quest'ultimo

[11]Come si evince dall'implementazione di `DGDecoder`, non tutti gli elementi LDS1 potenzialmente presenti nell'IC sono stati implementati. Il *framework* è in grado di decodificare correttamente esclusivamente COM, SOD, DG1, DG2, DG7, DG11, DG12 e DG14.

viene implementato da ogni DG concreto, in base alle proprie specifiche, e viene richiamato alla fine dell'inizializzazione (Codice 6.19).

```
internal class DataGroup {
    internal var identifier: DGTag!
    private(set) var data: ASN1Node!

    internal required init (_ data: ASN1Node, identifier: DGTag) throws {
        self.identifier = identifier
        self.data = data

        try decode(data)
    }

    internal func decode(_ data: ASN1Node) throws { }
}
```

Codice 6.19: Implementazione della classe `DataGroup`

Come esempio concreto di `DataGroup` si riporta l'implementazione del gruppo di dati `DataGroup1`, la cui decodifica permette di ottenere il contenuto della MRZ (Codice 6.20).

```
internal final class DataGroup1: DataGroup {
    private(set) var travelDocument: TravelDocument?

    internal required init(_ data: ASN1Node, identifier: DGTag) throws {
        try super.init(data, identifier: identifier)
    }

    override internal func decode(_ data: ASN1Node) throws {
        guard data.children?.firstChild?.tag == 0x5F1F else {
            throw NFCPassportReaderError.UnexpectedResponseStructure
        }

        guard case .primitive(let travelDocumentData) = data.children?.firstChild
    ?.content else {
            throw NFCPassportReaderError.UnexpectedResponseStructure
        }

        self.travelDocument = try TravelDocument(data: [UInt8](travelDocumentData)
    )
    }
}
```

Codice 6.20: Implementazione della classe `DataGroup1`

6.4 Meccanismi di sicurezza

Quando si ha a che fare con dati personali e sensibili è necessario porre una maggiore attenzione nei confronti della sicurezza informatica. In questo specifico

caso, non va assolutamente dimenticato che un eMRTD rappresenta di fatto un documento d'identità in formato elettronico che memorizza informazioni personali e dati biometrici. Per questo motivo è necessario garantire in ogni modo riservatezza e integrità informatica delle informazioni contenute nell'IC.

Le raccomandazioni internazionali ICAO prevedono che un eMRTD implementi diversi meccanismi di sicurezza per impedire:

- la lettura non autorizzata dei dati nell'IC;
- intercettazioni sulla comunicazione tra l'IC e il sistema di ispezione (IFD);
- la manomissione delle informazioni memorizzate;
- la sostituzione e/o la manomissione dell'IC.

6.4.1 Accesso all'IC

L'assenza di un meccanismo di controllo e di autorizzazione per l'accesso ai dati espone l'IC a due possibili rischi:

- i dati memorizzati nell'IC potrebbero essere letti senza alcuna autorizzazione;
- se non cifrata, la comunicazione tra l'IC e l'IFD potrebbe essere intercettata, con la conseguente lettura dei messaggi scambiati.

Risulta, dunque, inevitabile introdurre un meccanismo di protezione per l'accesso all'IC. Il titolare del documento deve essere consapevole della imminente lettura sicura dei dati e l'IC deve negare il recupero delle informazioni se l'IFD non è in grado di provare alcuna autorizzazione di accesso. Quest'ultima viene concessa attraverso un protocollo crittografico che si basa sulla conoscenza, da parte del lettore, di informazioni recuperate direttamente dal documento fisico. In questo modo, assumendo che non si possono ottenere informazioni dal documento fisico senza averne preso visione, si ammette che l'eMRTD sia stato consapevolmente fornito per l'ispezione. Inoltre, attraverso la cifratura del canale di comunicazione con l'IC, decifrare i messaggi scambiati, eventualmente intercettati, richiederebbe un costo computazionale troppo elevato [32].

Un eMRTD può implementare due diversi meccanismi di sicurezza per garantire l'accesso sicuro all'IC:

- *Basic Access Control* (BAC), basato sulla crittografia simmetrica;
- *Password Authenticated Connection Establishment* (PACE), basato sulla crittografia asimmetrica, la quale fornisce maggiore entropia alle chiavi di sessione.

Un eMRTD può essere configurato per implementare esclusivamente il meccanismo BAC oppure entrambe le strategie, ossia BAC e PACE. Tuttavia, la prima metodologia di accesso, in futuro, non sarà più supportata e, dunque, tutti gli eMRTD implementeranno esclusivamente PACE [32]. In generale, la procedura di accesso all'IC al fine di autenticare l'IFD si suddivide nelle operazione riportate qui di seguito.

1. Se il meccanismo PACE è supportato dall'eMRTD, allora è innanzitutto necessario eseguire la lettura di EF.CardAccess per recuperare i parametri crittografici supportati dall'IC, altrimenti si procede con il meccanismo BAC (punto 4).

2. Opzionalmente, è possibile effettuare la lettura di EF.DIR per recuperare la lista di applicazioni presenti sull'IC.

3. Se supportato, si procede con l'autenticazione dell'IFD mediante PACE. In caso di esito negativo, si tenta la procedura di autenticazione mediante BAC (punto 4).

4. Si procede con l'autenticazione del sistema di ispezione mediante BAC.

In caso di esito positivo al punto 3, o eventualmente al punto 4, si stabilisce una sessione sicura che abilita la lettura dei dati sensibili presenti nell'IC[12].

Basic Access Control

A monte della procedura di autenticazione, l'IFD deve ricavare alcune informazioni dalla MRZ del documento. In particolare, dall'ispezione dell'eMRTD fisico si ottengono il numero e la data di scadenza del documento, nonché la data di nascita del titolare[13]. Ciò permette all'IFD di generare, attraverso una *key derivation function* (KDF), le *Document Basic Access Key*, ossia due chiavi, K_{ENC} e K_{MAC}, utilizzate rispettivamente per cifrare ed autenticare i messaggi durante le operazioni di accesso (Codice 6.21). Inoltre, essendo BAC basato su crittografia simmetrica, le stesse chiavi saranno generate ed utilizzate anche dall'IC.

```
1 internal final class DocumentBasicAccessKeys {
2     private(set) var Kenc: [UInt8]
3     private(set) var Kmac: [UInt8]
4
```

[12]Il flusso appena descritto completa l'esposizione del metodo `startReading(tagReader:)` di `NFCPassportReader` (Codice 6.13), discusso in 6.3.2.

[13]L'IFD recupera i dati necessari insieme alle corrispettive cifre di controllo, anch'esse riportate nella MRZ.

```
    internal init(mrzKey: String) throws {
        let digest = try HashAlgorithm.hash(
            [UInt8](mrzKey.data(using: .utf8)!),
            with: .SHA1
        )
        let Kseed = [UInt8](digest[0..<16])

        let generator = SessionKeyGenerator(
            securityConfig: .init(encryptionAlgorithm: .DESEDE2)
        )
        self.Kenc = try generator.deriveKey(keySeed: Kseed, mode: .ENC_MODE)
        self.Kmac = try generator.deriveKey(keySeed: Kseed, mode: .MAC_MODE)
    }
}
```

Codice 6.21: Implementazione della classe `DocumentBasicAccessKeys`

Il meccanismo BAC (Codice 6.22) utilizza 3DES come algoritmo di cifratura a blocchi[14]ed implementa un protocollo *challenge-response* a tre passaggi per concordare un segreto.

1. Per avviare la procedura di autenticazione, l'IFD emette un comando APDU `GET CHALLENGE` all'IC. Questo comando richiede al circuito integrato di generare e restituire un *nonce*, che funge da sfida per l'autenticazione.

2. L'IFD genera un nuovo *nonce* e una chiave. Essi vengono inviati all'IC, insieme alla prima sfida, mediante un comando APDU `EXTERNAL AUTHENTICATE`.

3. Verificata la correttezza della sfida dell'IC, quest'ultimo genera un'ulteriore chiave, la sostituisce a quella inviata dall'IFD e risponde.

4. Se anche la sfida dell'IFD è verificata, allora i due dispositivi si sono, di fatto, scambiati le chiavi con successo. La loro combinazione produce un segreto condiviso.

Una volta che il segreto è stato concordato, vengono generate le chiavi di sessione, KS_{ENC} e KS_{MAC} mediante KDF.

```
internal final class BACHandler {
    private var rndic: [UInt8] = []
    private var kic: [UInt8] = []

    private var rndifd: [UInt8] = []
    private var kifd: [UInt8] = []

```

[14]In realtà viene utilizzato l'algoritmo DES-EDE2, ossia una variante di 3DES che esegue tre operazioni DES, in particolare *encrypt-decrypt-encrypt*, dove le chiavi usate nelle due operazioni di cifratura coincidono.

```
8       // ...
9
10      internal func performBAC(mrzKey: String) async throws {
11          tagReader.secureSession.clear()
12          tagReader.secureSession.configure(
13              with: SecurityConfiguration(encryptionAlgorithm: .DESEDE2)
14          )
15
16          let documentBasicAccessKeys = try DocumentBasicAccessKeys(mrzKey: mrzKey)
17
18          let challengeResponse = try await tagReader.getChallenge()
19          self.rndic = challengeResponse.data
20
21          let commandData = try computeMutualAuthenticationCommandData(
22              documentBasicAccessKeys: documentBasicAccessKeys
23          )
24          let mutualAuthenticationResponse = try await tagReader
25              .sendMutualAuthenticate(data: Data(commandData))
26
27          guard mutualAuthenticationResponse.data.count > 0 else {
28              throw NFCPassportReaderError.InvalidMRZKey
29          }
30
31          let eic = try tagReader.secureSession
32              .securityConfig!
33              .encryption
34              .decrypt(
35                  key: documentBasicAccessKeys.Kenc,
36                  message: [UInt8](mutualAuthenticationResponse.data[0..<32])
37              )
38
39          self.kic = [UInt8](eic[16..<32])
40
41          let sharedSecret = self.kic.xor(self.kifd)
42          let ssc = [UInt8](self.rndic.suffix(4) + self.rndifd.suffix(4))
43
44          try tagReader.secureSession.establish(
45              secret: sharedSecret,
46              sendSquenceCounter: ssc
47          )
48      }
49  }
```

Codice 6.22: Implementazione della classe `BACHandler`

Password Authenticated Connection Establishment

La procedura PACE utilizza *Diffie-Hellman* (DH) in forma anonima come protocollo di *key agreement* e una strategia di mappatura dei parametri di dominio per concordare il segreto condiviso e generare le chiavi di sessione. L'effettiva implementazione del meccanismo PACE può variare in base all'utilizzo non solo di DH basato su curve ellittiche (ECDH) o meno, ma anche del protocollo di *mapping* e dell'algoritmo di cifratura e di autenticazione dei messaggi. Qui si descriverà esclusivamente la variante PACE con DH, *Generic Mapping* (GM) e

3DES in quanto la configurazione corrisponde con quella messa in pratica dalla CIE e maggiormente analizzata in fase di sviluppo. Si specifica, inoltre, che un eMRTD può implementare più configurazioni contemporaneamente[15].

Anche in questo caso è necessario che l'IFD recuperi delle informazioni dal documento fisico. Per eseguire PACE è possibile recuperare dall'eMRTD sia le stesse informazioni utilizzate in BAC sia il CAN (si veda 5.1.2). Essendo quest'ultimo un codice di qualche cifra, ha il vantaggio di poter essere facilmente inserito nell'IFD, anche manualmente. La procedura PACE (Codice 6.23) ha inizio con il comando APDU `MSE:Set AT`, che inizializza il protocollo di accesso specificando l'informazione recuperata dall'eMRTD fisico e la configurazione crittografica scelta. Successivamente, si prosegue con la strategia di calcolo del segreto condiviso.

1. L'IFD invia un comando APDU `GENERAL AUTHENTICATE` per richiedere all'IC di generare un *nonce* s. Quest'ultimo viene prima cifrato con una chiave, prodotta sia dall'IC che dall'IFD attraverso l'informazione recuperata dal documento, e poi inviato all'IFD.

2. Decifrato s, si produce una coppia di chiavi effimere, pubblica e privata, che saranno utilizzate esclusivamente durante il processo di *mapping* dei parametri di dominio. Quindi si procede con una prima applicazione del protocollo DH: IFD ed IC si scambiano le chiavi pubbliche tramite `GENERAL AUTHENTICATE` e, in combinazione con la propria chiave privata, si genera un primo segreto condiviso h.

3. I parametri di dominio recuperati da EF.CardAccess includono un generatore crittografico g, usato per la generazione delle chiavi. La fase di *mapping*, nel caso di GM, prevede di trasformare tale generatore g in un nuovo generatore $\hat{g}$ attraverso una funzione $M : g \to \hat{g}$ tale che:

   $$\hat{g} = g^s \cdot h \mod p$$

   dove p è un numero primo, anch'esso incluso nei parametri di dominio.

4. A questo punto, IFD ed IC generano una nuova coppia di chiavi effimere, facendo uso del nuovo generatore. Infine, applicano nuovamente il protocollo DH per generare un nuovo segreto condiviso.

Anche in questo caso, concordato il segreto, si stabilisce una sessione protetta e si generano le chiavi di sessione, KS_{ENC} e KS_{MAC}, mediante KDF. In aggiunta, la conclusione del protocollo PACE prevede anche lo scambio e la verifica di un

[15]La lista delle configurazioni supportate dall'IC è ottenibile da EF.CardAccess.

authentication token. Esso è generato da IC ed IFD a partire dalla chiave KS_{MAC} e dalla chiave pubblica della controparte.

```
internal final class PACEHandler {
    private static let MRZ_PACE_KEY_REFERENCE: UInt8 = 0x01
    private static let CAN_PACE_KEY_REFERENCE: UInt8 = 0x02

    private var paceInfo: PACEInfo

    internal init(tagReader: TagReader, cardAccess: CardAccess) throws {/* ... */}

    internal func performPACE(mrzKey: String) async throws {
        guard isPACESupported else {
            throw NFCPassportReaderError.NotSupported("PACE not supported")
        }

        let paceKey = try derivePaceKey(from: mrzKey)
        let encodedPACEInfoOid = paceInfo.oid.encode(withTag: 0x80)
        let encodedPACEUsedKeyType = ASN1BasicEncoder.encode(
            tag: 0x83,
            data: [Self.MRZ_PACE_KEY_REFERENCE]
        )
        let encodedPACEParametersId = ASN1BasicEncoder.encode(
            tag: 0x84,
            data: [UInt8(PACEParametersDecoder.getParametersId(
                from: paceInfo.parameters!
            ) ?? 0)]
        )
        let encodedPACEInfoData = Data(
            encodedPACEInfoOid +
            encodedPACEUsedKeyType +
            encodedPACEParametersId
        )
        try await tagReader.sendMSESetAT(
            data: encodedPACEInfoData,
            for: .mutualAuthentication
        ).discardResponse()
        let decryptedNonce = try await computeDecryptedNonce(withKey: paceKey)
        let ephemeralParams = try await computeEphemeralParams(
            decryptedNonce: decryptedNonce
        )
        let sharedSecret = try await performKeyAgreement(
            ephemeralParams: ephemeralParams
        )

        tagReader.secureSession.clear()
        tagReader.secureSession.configure(
            with: SecurityConfiguration(
                encryptionAlgorithm: paceInfo
                    .securityProtocol
                    .usedEncryptionAlgorithm
            )
        )
        try tagReader.secureSession.establish(secret: sharedSecret)
    }
}
```

Codice 6.23: Implementazione della classe `PACEHandler`

6.4.2 Sessione sicura

Lo scopo finale dei meccanismi di accesso BAC e PACE, come si è constato, è quello di generare delle chiavi di sessione per garantire riservatezza e integrità dei messaggi scambiati durante la comunicazione tra IC ed IFD. Questa strategia consente di stabilire una sessione sicura e di garantire una comunicazione protetta (Codice 6.24).

```
internal final class NFCSecureSession {
    private(set) var securityConfig: SecurityConfiguration?
    private(set) var secureChannel: SecureChannel?
    private(set) var secureMessaging: SecureMessaging?

    internal var isSecureSessionEstablished: Bool {/*...*/}

    internal func configure(with config: SecurityConfiguration) {/*...*/}

    internal func establish(secret: [UInt8], sendSquenceCounter: [UInt8]? = nil)
    throws {
        guard let config = self.securityConfig else {
            throw NFCPassportReaderError.UnkownSecurityConfiguration
        }

        let generator = SessionKeyGenerator(securityConfig: config)
        let KSenc = try generator.deriveKey(keySeed: secret, mode: .ENC_MODE)
        let KSmac = try generator.deriveKey(keySeed: secret, mode: .MAC_MODE)
        let ssc = sendSquenceCounter ?? [UInt8](
            repeating: 0x00,
            count: config.encryption.params.blockSize
        )

        let channel = SecureChannel(KSenc: KSenc, KSmac: KSmac, ssc: ssc)
        self.secureChannel = channel

        self.secureMessaging = .init(
            secureChannel: channel,
            securityConfig: config
        )
    }

    internal func clear() {/*...*/}
}
```

Codice 6.24: Implementazione della classe `NFCSecureSession`

Le chiavi di sessione sono utilizzate nel protocollo *Secure Messaging* (Codice 6.25) definito in ISO/IEC 7816-4. Tale procedura prevede che, una volta stabilita una sessione, i dati inviati tramite comandi e risposte APDU vengano cifrati ed autenticati. In aggiunta, si introduce una strategia di numerazione dei messaggi inviati e ricevuti attraverso il *Send Sequence Counter* (SSC). Tale valore viene incrementato prima dell'invio di ogni comando e risposta APDU, oltreché essere incluso nel calcolo del MAC di ogni messaggio affinché se ne possa verificare la correttezza.

```swift
internal final class SecureMessaging {

    private var securityConfig: SecurityConfiguration
    private var secureChannel: SecureChannel

    // ...

    internal func protect(apdu: NFCISO7816APDU) throws -> NFCISO7816APDU {
        secureChannel.incrementSSC()

        // ...

        let N = DataPadder.pad(
            data: secureChannel.ssc + M,
            blockSize: securityConfig.encryption.params.blockSize
        )

        let CC = try securityConfig
            .encryption
            .mac(key: secureChannel.KSmac, message: N)
            .prefix(8)

        // ...

        return NFCISO7816APDU(data: Data(protectedAPDU))!
    }

    internal func unprotect(rapdu: APDUResponse) throws -> APDUResponse {
        secureChannel.incrementSSC()

        // ...

        return .init(data: decryptedData, sw1: sw1, sw2: sw2)
    }
}
```

Codice 6.25: Implementazione della classe `SecureMessaging`

6.4.3 Autenticazione dei dati

Per garantire che il contenuto di EF.SOD, e di conseguenza dell'interno contenuto di LDS1, sia autentico e inalterato, è necessario implementare un'ulteriore meccanismo di protezione. Esso è sviluppato attraverso la strategia di autenticazione dei dati *Passive Authentication* (PA), inclusa in ogni eMRTD.

Passive Authentication

Per verificare che i dati recuperati dall'IC siano autentici è sufficiente verificare la correttezza di EF.SOD. Come detto in 6.3.1, esso contiene i *digest* di tutti i DG presenti all'interno dell'applicazione LDS1. Inoltre, EF.SOD è firmato digitalmente dallo Stato all'emissione del documento. Di conseguenza, la PA prevede la verifica della firma di EF.SOD attraverso un'infrastruttura a chiave pubblica e un controllo

di uguaglianza tra i *digest* presenti all'interno dell'IC e quelli calcolati applicando una determinata funzione di *hash* ai DG recuperati [32]. Sebbene questa sia la procedura completa per attuare correttamente il protocollo PA, la fase di verifica della firma di EF.SOD non è stata approfondita durante gli sviluppi del *framework*. Pertanto, la logica implementativa riguardante l'autenticazione dei dati si limita alla verifica dei *digest* (Codice 6.26).

```
internal final class PassiveAuthenticationHandler {
    private var sod: SOD

    internal func performPassiveAuthentication(on dataGroups: [DataGroup]) throws{
        // ...

        try dataGroups.filter {
            $0.identifier != .COM && $0.identifier != .SOD
        }.forEach { dataGroup in
            guard let currentSodHash = sod
                .signedData
                .encapContentInfo[dataGroup.identifier]
            else {
                throw NFCPassportReaderError.PassiveAuthenticationFailed(
                    "Data group hash not found in SOD"
                )
            }

            let computedHash = try HashAlgorithm.hash(
                [UInt8](dataGroup.data.encodedBytes),
                with: sod.signedData.digestAlgorithm ?? .SHA1
            )

            if computedHash != currentSodHash {
                throw NFCPassportReaderError.PassiveAuthenticationFailed(
                    "\(String(describing: dataGroup.identifier)) hash not match"
                )
            }
        }
    }
}
```

Codice 6.26: Implementazione della classe `PassiveAuthenticationHandler`

Si specifica, infine, che l'esecuzione di tale meccanismo di sicurezza si ricollega alle operazione effettuate dopo la lettura di tutti i DG presenti nell'IC (Codice 6.14) descritto in 6.3.2.

6.4.4 Autenticazione dell'IC

Lo Stato emittente dell'eMRTD può scegliere se implementare anche una strategia di autenticazione dell'IC per garantire che quest'ultimo non sia stato sostituito e che le informazioni siano recuperate dal *chip* originale. In particolare, possono opzionalmente essere implementati tre diversi meccanismi [32].

- La presenza di EF.DG15 indica il supporto alla *Active Authentication*. Tale strategia prevede di autenticare l'IC attraverso la verifica della firma di una sfida inviata dall'IFD. Quest'ultima viene firmata dall'IC mediante una chiave privata memorizzata in modo sicuro e nota esclusivamente al circuito integrato. Affinché il controllo sia attendibile, è richiesta l'autenticazione dei dati tramite PA, la quale garantisce il recupero della corretta chiave pubblica da EF.DG15.
- La presenza di EF.DG14 indica il supporto alla *Chip Authentication* (CA), che, a differenza del precedente meccanismo, si basa sul protocollo DH.
- La presenza della corrispettiva configurazione in EF.CardAccess indica il supporto al meccanismo di accesso PACE con *Chip Authentication Mapping*. Esso estende il protocollo GM al fine di includere la strategia di autenticazione dell'IC nel processo di accesso al circuito.

Di seguito si descriverà esclusivamente il protocollo CA in quanto supportato dal *framework*.

Chip Authentication

Il meccanismo CA richiede innanzitutto che sia stato correttamente effettuato l'accesso all'IC affinché si possano recuperare le informazioni contenute in EF.DG14. Infatti, esso include non solo la specifica configurazione di CA supportata ma anche una chiave pubblica dell'IC, da utilizzare durante il processo di autenticazione. La corrispettiva chiave privata è, invece, memorizzata in modo sicuro all'interno della memoria del *chip*. Recuperata la chiave pubblica da EF.DG14, IFD ed IC concordano un segreto mediante il protocollo DH [32].

1. L'IFD genera una coppia di chiavi effimere, pubblica e privata, ed invia quella pubblica all'IC tramite un comando APDU.
2. Sia IFD che IC utilizzano la propria chiave privata e quella pubblica della controparte per applicare DH e generare un segreto condiviso.

Sulla base del segreto concordato, si generano due nuove chiavi di sessione, KS_{ENC} e KS_{MAC}, mediante KDF e si riavvia la sessione sicura con le nuove chiavi prodotte (Codice 6.27).

```
1 internal final class ChipAuthenticationHandler {
2
3     private var chipAuthenticationInfos: [KeyId : ChipAuthenticationInfo] = [:]
4     private var chipAuthenticationPublicKeyInfos:
5     [ChipAuthenticationPublicKeyInfo] = []
6
7     internal init(tagReader: TagReader, dg14: DataGroup14) {/*...*/}
8
```

```
    // ...
}

private extension ChipAuthenticationHandler {
    private func performCA(
        keyId: Int?,
        protocol caProtocol: ChipAuthenticationSecurityProtocol,
        subjectPublicKeyInfo: SubjectPublicKeyInfo
    ) async throws {
        let ephemeralKeyPair = try KeyAgreementAlgorithm
            .generateKeyPair(withParamsFrom: subjectPublicKeyInfo.publicKey)

        defer { EVP_PKEY_free(ephemeralKeyPair) }

        let ephemeralPublicKey = try KeyAgreementAlgorithm
            .extractPublicKey(from: ephemeralKeyPair)

        try await sendPublicKey(
            keyId: keyId,
            protocol: caProtocol,
            publicKey: ephemeralPublicKey
        )

        let sharedSecret = try caProtocol
            .usedKeyAgreementAlgorithm
            .computeSharedSecret(
                personalKeyPair: ephemeralKeyPair,
                externalPublicKey: subjectPublicKeyInfo.subjectPublicKeyBytes
            )

        defer { caProtocol.usedKeyAgreementAlgorithm.free(
            sharedSecret: sharedSecret
        )}

        let sharedSecretBytes = try caProtocol
            .usedKeyAgreementAlgorithm
            .convertToBytes(
                key: sharedSecret,
                keyPair: ephemeralKeyPair
            )

        try restartSecureSession(
            caProtocol: caProtocol,
            sharedSecret: sharedSecretBytes
        )
    }
}
```

Codice 6.27: Implementazione della classe `ChipAuthenticationHandler`

Si specifica, infine, che la descrizione di tale meccanismo di sicurezza si ricollega alle operazione effettuate dopo la lettura di EF.DG14 (Codice 6.14), completando di fatto la descrizione del relativo metodo descritto in 6.3.2.

Capitolo 7

Nuove esperienze interattive

Le analisi svolte sulle raccomandazioni internazionali ICAO in merito agli eMRTD hanno permesso la realizzazione di un *framework* capace di eseguire la corretta lettura NFC delle informazioni memorizzate nella CIE. Questo ha aperto la strada allo sviluppo di nuove esperienze interattive per le applicazioni iOS utilizzando la tecnologia NFC, partendo dalla fase di ricerca e progettazione, fino alla realizzazione del *framework* e alla sua integrazione in *app*.

Il corretto funzionamento del *framework* e il raggiungimento degli obiettivi prefissati sono stati verificati tramite l'integrazione di nuove funzionalità all'interno dell'applicazione introdotta nel capitolo 4. Di conseguenza, come risultato conclusivo della ricerca condotta per questa tesi, in questo capitolo verranno presentate le funzionalità NFC introdotte. In particolare, l'applicazione è stata estesa affinché prevedesse i seguenti casi d'uso:

- estensione del flusso di autenticazione per verificare l'identità del cliente attraverso il proprio documento d'identità elettronico;
- verifica addizionale dell'identità del cliente attraverso la propria CIE per confermare la richiesta di un'operazione bancaria ritenuta delicata, come un bonifico il cui importo è maggiore di una certa soglia.

Infine, è stata integrata anche una nuova sezione per la visualizzazione a schermo di tutte le informazioni recuperate dalla CIE, comprese le configurazioni di sicurezza.

7.1 Processo di autenticazione dell'utente

Rispetto alla versione presentata nel capitolo 4, all'interno dell'applicazione è stato inserito un processo di autenticazione del cliente. Come avviene nella maggior parte della applicazioni bancarie, il flusso richiede un identificativo associato all'utente, che in questo caso coincide con l'indirizzo email, e un codice PIN (Fig. 7.1).

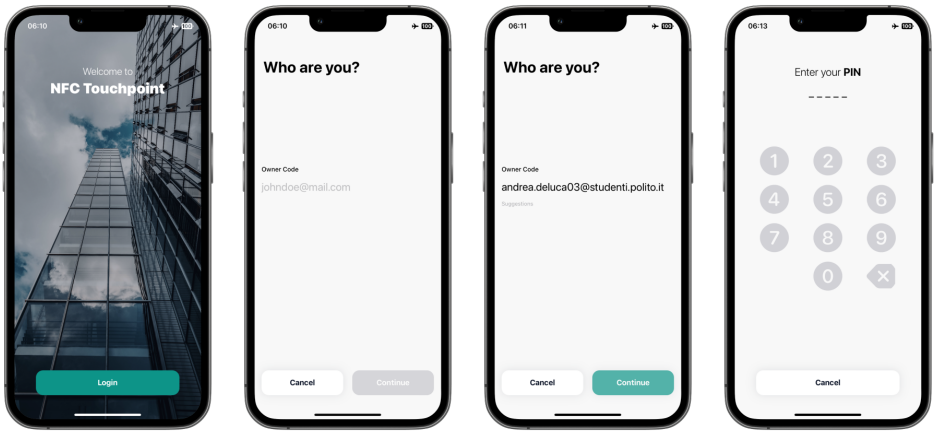

Figura 7.1: Flusso di autenticazione del cliente

Tale flusso di autenticazione è stato esteso al fine di prevedere la verifica dell'identità del cliente mediante l'utilizzo della tecnologia NFC, in combinazione con la propria CIE (Fig. 7.2). Ciò è possibile grazie all'utilizzo del *framework* sviluppato.

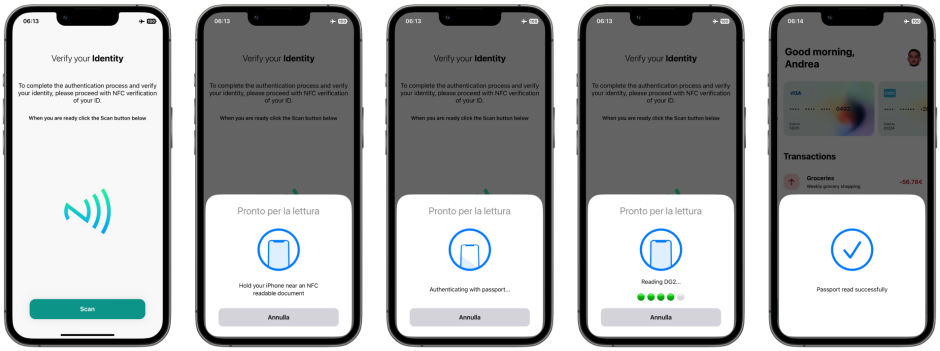

Figura 7.2: Estensione del flusso di autenticazione tramite NFC

Si specifica, inoltre, che per la realizzazione di questa funzionalità si è supposto che i dati necessari per effettuare la lettura della CIE siano stati memorizzati dal sistema durante la fase di *onboarding*. Si ipotizza, ad esempio, un processo di

registrazione del proprio documento di riconoscimento compatibile, anche differito rispetto alla creazione del proprio account, in cui viene chiesto all'utente di inserire manualmente il numero di serie della CIE, la data di scadenza del documento e la propria data di nascita, successivamente validati attraverso un tentativo di scansione NFC della carta d'identità. Alternativamente, si potrebbe pensare di integrare un modulo OCR per la lettura della MRZ.

7.2 Conferma delle operazioni bancarie

Verificata con successo l'identità dell'utente e concluso il processo di autenticazione, è possibile avviare un flusso per l'esecuzione di un bonifico bancario. Navigando nell'apposita sezione, viene prima richiesto di inserire il nome del beneficiario, il codice IBAN associato al conto corrente di quest'ultimo e un messaggio (Fig. 7.3).

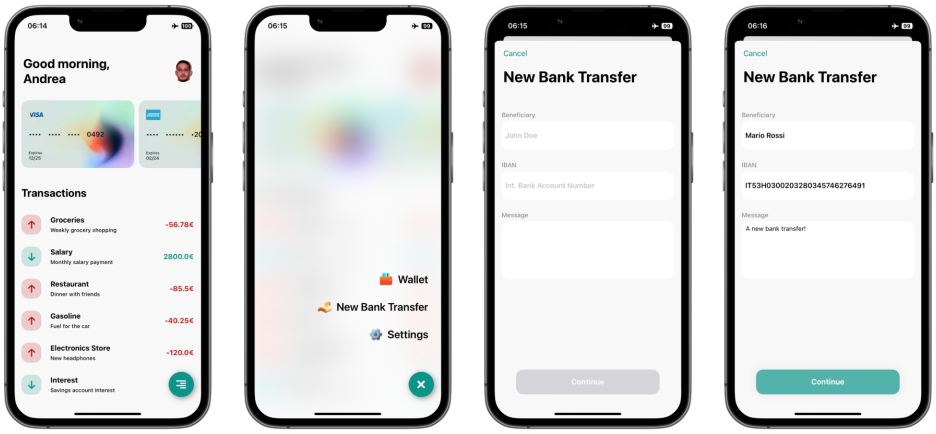

Figura 7.3: Richiesta di un nuovo bonifico bancario

Successivamente, viene richiesto l'importo da trasferire al conto corrente indicato e, alla conferma, si procede con l'operazione bancaria (Fig. 7.4).

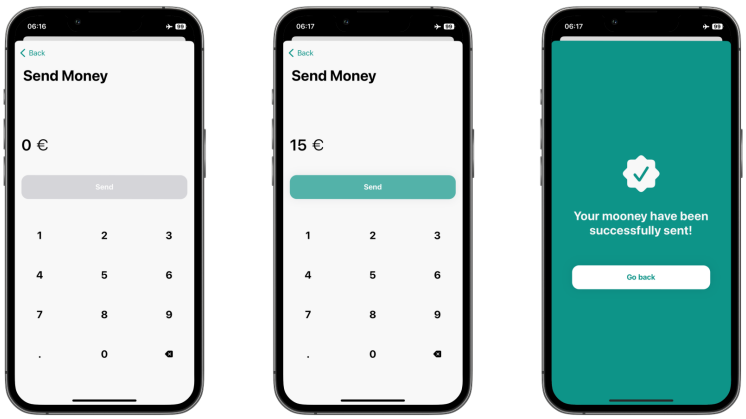

Figura 7.4: Invio di un bonifico bancario (non delicato)

Tuttavia, nel momento in cui l'importo risulta essere maggiore di una certa soglia l'operazione è considerata delicata, data l'elevata somma di denaro da trasferire. Pertanto, anche in questa circostanza, si è aggiunta una verifica addizionale dell'identità del cliente tramite la propria CIE (Fig. 7.5).

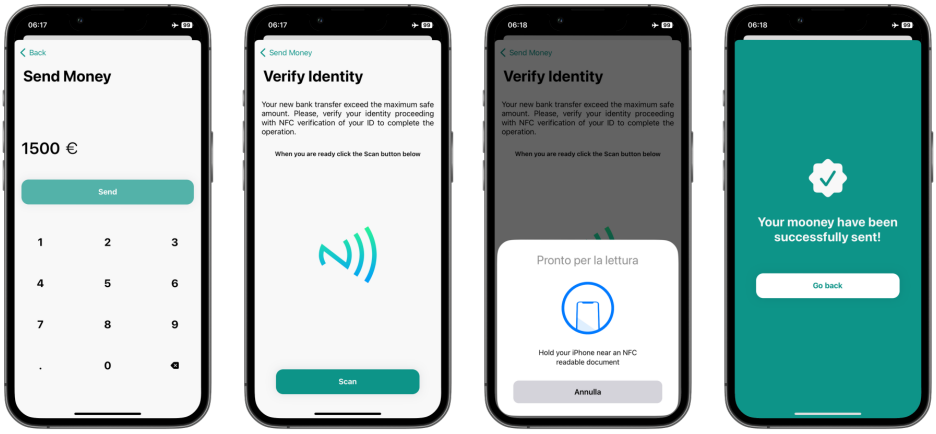

Figura 7.5: Invio di un bonifico bancario (delicato) con verifica tramite NFC

7.3 Visualizzazione delle informazioni recuperate

A scopo puramente dimostrativo e di analisi, all'interno dell'applicazione è stata integrata anche una sezione che permette di avere una visualizzazione delle informazioni recuperate dalla CIE. La schermata è raggiungibile tramite la foto profilo visibile nella schermata principale, in alto a destra. Essa coincide con la foto stampata sul documento d'identità e viene recuperata tramite lettura NFC durante l'accesso in *app*. All'interno della schermata viene ricreata una rappresentazione del documento scansionato e viene mostrata la lista di tutte le informazioni recuperate, tra cui i dettagli e le configurazioni in merito ai meccanismi di sicurezza, il contenuto di EF.SOD e la lista di DG recuperati, nonché i dettagli sul titolare e sul documento (Fig. 7.6).

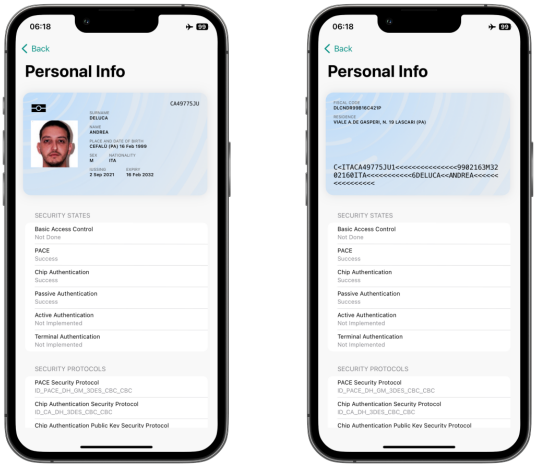

Figura 7.6: Visualizzazione dei dati recuperati dalla CIE

Capitolo 8

Conclusioni

Con lo studio affrontato in questa tesi si è voluto dimostrare il significativo potenziale della tecnologia NFC nello sviluppo di nuove esperienze interattive all'interno di applicazioni iOS. Trattare tutti i contesti in cui l'utilizzo della tecnologia in esame potrebbe rappresentare una soluzione innovativa ed efficiente risulta molto complesso a causa della sua versatilità e delle tante possibilità implementative che offre.

Per il raggiungimento degli obiettivi si è partiti dallo studio delle tecnologie di interesse, quali NFC, RFID e *smart card* senza contatto. Attraverso queste nozioni, siamo stati in grado di condurre delle analisi nel contesto delle transazioni digitali, nonché di comprendere le specifiche delle carte d'identità elettroniche, come la CIE. Grazie a questi studi, siamo stati in grado di raccogliere risultati e considerazioni.

- Abbiamo prima analizzato l'utilizzo di un iPhone come terminale POS. Anche se questa capacità non è stata sviluppata direttamente attraverso questa ricerca, i risultati raccolti dimostrano come sia di fatto possibile richiedere una transazione digitale attraverso il proprio *smartphone* senza l'ausilio di dispositivi terzi. Come abbiamo visto, Apple ha già rilasciato la funzionalità *Tap to pay on iPhone* in diversi Stati e fornisce agli sviluppatori ProximityReader come *framework* per l'integrazione della funzionalità in questione all'interno delle proprie applicazioni iOS.

- Per comprendere appieno il percorso verso l'adozione di nuove esperienze interattive tramite NFC ed implementare una soluzione innovativa partendo da zero, abbiamo successivamente approfondito il recupero delle informazioni memorizzate all'interno della CIE. Dai risultati ottenuti attraverso lo sviluppo di un *framework* a supporto di ciò e la conseguente integrazione di quest'ultimo all'interno di un'applicazione di *mobile banking*, è emerso come sia possibile implementare nuove funzionalità coinvolgenti, permettendo di migliorare non solo l'esperienza d'uso degli utenti, ma anche di creare valore aggiunto per gli sviluppatori e le aziende.

Tuttavia, è importante notare come la ricerca svolta evidenzia come ci siano ancora diverse sfide da affrontare. Ad esempio, durante lo sviluppo del *framework*, è stato riscontrato che i tanti meccanismi di sicurezza implementati all'interno dell'IC del documento di riconoscimento, per quanto sicuramente necessari, rendono lo sviluppo di soluzioni innovative piuttosto difficile. Inoltre, la possibilità di supportare una vasta gamma di configurazioni di sicurezza per ogni meccanismo aggiunge complessità implementativa considerevole. D'altra parte, sebbene non sia stato sviluppato all'interno del *framework* il supporto a tale variante, si ritiene molto interessante l'adozione del meccanismo PACE con *Chip Authentication Mapping* in quanto combina in un unico meccanismo di sicurezza la strategia di accesso al circuito integrato e il conseguente avvio di una sessione sicura con il protocollo di autenticazione dell'IC.

Nonostante il *framework* risulti sufficientemente completo affinché si possano recuperare informazioni dalla CIE, esso non supporta interamente le funzionalità offerte dalle raccomandazioni internazionali ICAO. Ciò determina sicuramente degli spunti per analisi, studi e possibili sviluppi futuri.

- Per prima cosa, non è stata implementata la decodifica di tutti i gruppi di dati possibilmente presenti, dunque un'integrazione completa potrebbe consentire la lettura di informazioni utili in determinati contesti.

- Per quanto siano state coperte altre configurazioni crittografiche, oltre a quelle implementate dalla CIE, il loro corretto funzionamento non è garantito a causa della mancanza di un supporto per la loro verifica. Ad esempio, è stato previsto anche l'utilizzo di AES (*Advanced Encryption Standard*), ECDH e diverse funzioni di hash, che potrebbero essere adoperati da altri eMRTD.

- Sebbene ci si è soffermati sullo sviluppo di un corposo insieme di meccanismi di sicurezza, il supporto non è completo. Lo dimostra, ad esempio, l'assenza della gestione del meccanismo di *Active Authentication*. Inoltre, anche alcuni dei meccanismi implementati richiederebbero ulteriori sviluppi, come la verifica della firma digitale applicata a EF.SOD per la *Passive Authentication* oppure il supporto di tutti i tipi di *mapping* possibili per il protocollo PACE.

In aggiunta, al di là dell'integrazione del *framework* in *app*, ulteriori studi andrebbero affrontati in merito alla sua adozione in un contesto reale, prendendo sicuramente in considerazione le possibili implicazioni e gli eventuali impatti sull'utilizzo e sulla protezione dei dati personali e biometrici recuperati dal documento. D'altra parte, potrebbe essere altrettanto interessante affrontare un'analisi più approfondita di ProximityReader e delle sue capacità al fine di fornire importanti spunti di riflessione su come la tecnologia NFC possa essere utilizzata in un'altra vasta gamma di contesti applicativi.

In conclusione, penso che la tecnologia NFC rappresenti un'opportunità concreta per rivoluzionare il modo in cui interagiamo con le applicazioni digitali. Il suo potenziale è già evidente nel modo in cui semplifica molte attività quotidiane, dalla gestione dei pagamenti elettronici alla condivisione di informazioni. Credo fortemente in una sua adozione sempre più pervasiva nei prossimi anni. Sempre più aziende e istituzioni si stanno spingendo verso una possibile integrazione della tecnologia NFC. Rimanendo ancora sul tema dell'identificazione personale, penso, ad esempio, al progetto europeo del *Digital Identity Wallet* e all'*IT Wallet*, ultimamente sempre più discusso dato l'imminente rilascio. In effetti, è facile considerare come l'integrazione della tecnologia NFC potrebbe essere un elemento chiave da includere in un *e-wallet* emesso dal governo italiano, in cui i cittadini possono conservare la propria patente di guida e CIE. D'altra parte, in un mondo sempre più interconnesso e vulnerabile agli attacchi informatici, considero i rischi legati alla protezione dei dati e alla privacy assolutamente non trascurabili e ritengo altrettanto fondamentale un'attenta analisi di questi temi. Pertanto, mentre ci entusiasmiamo per le possibilità offerte dall'NFC, dobbiamo anche essere pragmatici nel valutare e affrontare le sfide ancora aperte. Solo attraverso una gestione oculata e responsabile di questa tecnologia possiamo garantire che il suo impatto sia veramente positivo e duraturo nel tempo e che possa effettivamente introdurre nella nostra quotidianità valide, innovative e nuove esperienze coinvolgenti che abbiano la capacità di ridefinire la nostra interazione con il mondo digitale.

Bibliografia

[1] Vedat Coskun, Kerem Ok e Busra Ozdenizci. *Near Field Communication. From Theory to Practice (ebook)*. John Wiley & Sons, 2011, pp. 28–47.

[2] Maikii. *Che cos'è l'NFC? Alla scoperta di questa tecnologia e delle nuove potenzialità di Marketing*. 2023. URL: https://www.linkedin.com/pulse/che-cos%C3%A8-lnfc-alla-scoperta-di-questa-tecnologia-e-delle-nuove-/ (visitato il 15/03/2024).

[3] Fabio Meo. «Near Field Communication (NFC): interazione fisica-interazione virtuale». Università Ca' Foscari Venezia, 2012, pp. 8–25.

[4] International Organization for Standardization (ISO). *ISO/IEC 7816-4:2020 - Identification cards. Integrated circuit cards. Part 4: Organization, security and commands for interchange (Abstract)*. 2020. URL: https://www.iso.org/standard/77180.html (visitato il 27/03/2024).

[5] Paolo Talone e Giuseppe Russo. *RFId. Fondamenti di una tecnologia silenziosamente pervasiva*. 2ª ed. Vol. 1 - Introduzione agli RFId. Fondazione Ugo Bordoni, 2008. URL: http://www.rfid.fub.it/edizione_2/Parte_I.pdf (visitato il 18/03/2024).

[6] SumUp. *Pagamenti digitali: cosa sono e come sfruttarli al meglio per la tua attività*. URL: https://www.sumup.com/it-it/business-guide/pagamenti-digitali (visitato il 16/03/2024).

[7] SumUp. *Cos'è un pagamento POS? Una breve guida introduttiva sui pagamenti tramite POS*. URL: https://www.sumup.com/it-it/pagamento-pos/ (visitato il 16/03/2024).

[8] Dumindu Buddhika. «How Apple Pay Works Under the Hood». In: *Medium* (2018). URL: https://medium.com/free-code-camp/how-apple-pay-works-under-the-hood-8c3978238324 (visitato il 27/03/2024).

[9] GlobalPlatform. *Introduction to Secure Elements*. 2018. URL: https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Secure-Element-15May2018.pdf (visitato il 16/03/2024).

[10] Apple Inc. *Core NFC. Detect NFC tags, read messages that contain NDEF data, and save data to writable tags.* URL: https://developer.apple.com/documentation/corenfc (visitato il 17/03/2024).

[11] Apple Inc. *Building an NFC Tag-Reader App. Read NFC tags with NDEF messages in your app.* URL: https://developer.apple.com/documentation/corenfc/building_an_nfc_tag-reader_app (visitato il 17/03/2024).

[12] Apple Inc. *Tap to Pay on iPhone.* URL: https://developer.apple.com/tap-to-pay (visitato il 18/03/2024).

[13] Apple Inc. *Configuring Tap to Pay on iPhone. Set up all the necessary components so your app can begin using Tap to Pay on iPhone to read contactless payment cards.* URL: https://developer.apple.com/documentation/proximityreader/configuring-tap-to-pay-on-iphone (visitato il 18/03/2024).

[14] Ministero dell'Interno. *Cos'è la carta.* URL: https://www.cartaidentita.interno.gov.it/la-carta/ (visitato il 27/03/2024).

[15] Valentina Conte. «Carta d'identità elettronica: la rivoluzione non decolla, ce l'hanno solo in 300mila». In: *la Repubblica* (2017). URL: https://www.repubblica.it/tecnologia/2017/03/16/news/carta_d_identita_elettronica_la_rivoluzione_che_non_decolla-160653671 (visitato il 20/03/2024).

[16] Ministero dell'Interno. *Caratteristiche del documento.* URL: https://www.cartaidentita.interno.gov.it/categoria-vuota/caratteristiche-del-documento/ (visitato il 27/03/2024).

[17] Ministero degli Affari Esteri e della Cooperazione Internazionale. *ICAO - Organizzazione Internazionale per l'Aviazione Civile.* URL: https://www.esteri.it/it/politica-estera-e-cooperazione-allo-sviluppo/organizzazioni_internazionali/icao-organizzazione-internazionale-per-laviazione-civile (visitato il 10/02/2024).

[18] ICAO. *Doc 9303. Machine Readable Travel Documents.* 8ª ed. *Part 1: Introduction.* 2021. URL: https://www.icao.int/publications/Documents/9303_p1_cons_en.pdf (visitato il 11/02/2024).

[19] ICAO. *Doc 9303. Machine Readable Travel Documents.* 8ª ed. *Part 3: Specifications Common to all MRTDs.* 2021. URL: https://www.icao.int/publications/Documents/9303_p3_cons_en.pdf (visitato il 11/02/2024).

[20] ICAO. *Doc 9303. Machine Readable Travel Documents.* 8ª ed. *Part 5: Specifications for TD1 Size Machine Readable Official Travel Documents (MROTDs).* 2021. URL: https://www.icao.int/publications/Documents/9303_p5_cons_en.pdf (visitato il 12/02/2024).

[21] ICAO. *Doc 9303. Machine Readable Travel Documents*. 8ª ed. *Part 6: Specifications for TD2 Size Machine Readable Official Travel Documents (MROTDs)*. 2021. URL: https://www.icao.int/publications/Documents/9303_p6_cons_en.pdf (visitato il 12/02/2024).

[22] ICAO. *Doc 9303. Machine Readable Travel Documents*. 8ª ed. *Part 4: Specifications for Machine Readable Passports (MRPs) and other TD3 Size MRTDs*. 2021. URL: https://www.icao.int/publications/Documents/9303_p4_cons_en.pdf (visitato il 12/02/2024).

[23] ICAO. *Doc 9303. Machine Readable Travel Documents*. 8ª ed. *Part 9: Deployment of Biometric Identification and Electronic Storage of Data in eMRTDs*. 2021. URL: https://www.icao.int/publications/Documents/9303_p9_cons_en.pdf (visitato il 17/02/2024).

[24] ICAO. *Doc 9303. Machine Readable Travel Documents*. 8ª ed. *Part 10: Logical Data Structure (LDS) for Storage of Biometrics and Other Data in the Contactless Integrated Circuit (IC)*. 2021. URL: https://www.icao.int/publications/Documents/9303_p10_cons_en.pdf (visitato il 17/02/2024).

[25] John Larmouth. *ASN. 1 complete*. Morgan Kaufmann, 1999.

[26] OSS Nokalva. *ASN.1 Made Simple. Encoding Rules*. URL: https://www.oss.com/asn1/resources/asn1-made-simple/encoding-rules.html (visitato il 22/02/2024).

[27] Olivier Dubuisson. *ASN. 1 communication between heterogeneous systems*. Morgan Kaufmann, 2000.

[28] OSS Nokalva. *ASN.1 Made Simple. Quick ASN.1 Reference – Basic Encoding Rules*. URL: https://www.oss.com/asn1/resources/asn1-made-simple/asn1-quick-reference/basic-encoding-rules.html (visitato il 22/02/2024).

[29] OSS Nokalva. *ASN.1 Made Simple. Quick ASN.1 Reference – ASN.1 Tags*. URL: https://www.oss.com/asn1/resources/asn1-made-simple/asn1-quick-reference/asn1-tags.html (visitato il 22/02/2024).

[30] OSS Nokalva. *ASN.1 Made Simple. Quick ASN.1 Reference – Distinguished Encoding Rules*. URL: https://www.oss.com/asn1/resources/asn1-made-simple/asn1-quick-reference/distinguished-encoding-rules.html (visitato il 23/02/2024).

[31] Ministero dell'Interno. *Carta d'Identità Elettronica CIE 3.0. Specifiche Chip*. URL: https://www.cartaidentita.interno.gov.it/downloads/2021/03/cie_3.0_-_specifiche_chip.pdf (visitato il 25/02/2024).

[32] ICAO. *Doc 9303. Machine Readable Travel Documents*. 8ª ed. *Part 11: Deployment of Biometric Identification and Electronic Storage of Data in eMRTDs*. 2021. URL: https://www.icao.int/publications/Documents/9303_p11_cons_en.pdf (visitato il 17/02/2024).