ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DISI

INGEGNERIA INFORMATICA

TESI DI LAUREA

in
TECNOLOGIE WEB T

APPLICAZIONI ANDROID PER ACCESSO A PERSONAL HEALTH RECORD

CANDIDATO
Pasquale Carlo Maiorano Picone

RELATORE:
Chiar.mo Prof. Ing. Paolo Bellavista

CORRELATORE:
Dott. Ing. Luca Foschini

Anno Accademico 2012/2013

Sessione I

Sommario

Introduzione 7
Capitolo 1 – Android 9
1.1 – Profilo del Sistema Operativo 9
1.1.1 – Breve storia dello sviluppo con accenni al rilascio delle versioni più significative 9
1.1.2 – Caratteristiche Principali 11
1.2 – Architettura Android 12
1.2.1 – Panoramica 13
1.2.2– Application Framework e Libraries 14
1.2.3 – Android Runtime Environment 16
1.2.4 – Inizializzazione della piattaforma 17
1.2.5 – La sicurezza 18
1.3 – Struttura delle applicazioni Android 18
1.3.1 - Activity e Intent 19
1.3.2 - Services 25
1.3.3– Broadcast Receiver 28
1.3.4 – Content Provider 30
1.3.5 – Manifest 35
1.3.6 – Application Package e firma digitale 38
1.4 – Processo di sviluppo per applicazioni Android 39
1.4.1 – Android Software Development Kit ed Android Virtual Device 39
1.4.2 – Android Debug Bridge 42
1.4.3 – Eclipse 44
Capitolo 2 – E-Health 45
2.1- Introduzione all'e-Health 45
2.1.1 – Il termine 45
2.1.2 – Le dieci regole 47
2.1.3 –I modelli principali delle applicazioni nel campo dell'E-Health 49
2.2 – Da E-Health a M-Health 51
2.2.1 – I requisiti 51
2.2.2 – I problemi 52
2.2.3 – Linee guida per una integrazione di successo 55
2.2.4 – I possibili scenari per un sistema M-Health (un particolare caso di studio riguardante le malattie cardiovascolari) 56
2.2.5 – Il Fascicolo Sanitario Elettronico 62

2.3 – Android,M-Health e E-Health 69
2.3.1 – Le applicazioni Android per E-Health. 70
2.3.2 – Android e M-Health 77
Capitolo 3 – Lo stato dell'arte della crittografia su Android 81
3.1 – Introduzione alla Crittografia 81
3.1.1 – Tipi di Crittografia 81
3.2 – La Crittografia nei linguaggi di programmazione 85
3.2.1 – Crypto++ 85
3.2.2 – Stanford Javascript Crypto Library 86
3.2.3 – Il Package javax.crypto 87
3.2.4 – Bouncy Castle 88
3.3 – La Crittografia nei sistemi mobili 89
3.3.1 – Un caso di studio: LPKI in ambiente mobile. 90
3.4 – Lo stato dell'arte della Crittografa sul sistema operativo mobile Android 97
3.4.1 – Il porting di javax.crypto 97
3.4.2 – Spongy Castle 100
3.4.3 – ULTRA Android Library 101
3.4.4 – La crittografia a blocchi implementata con NDK, un caso di studio 102
3.5 – Le applicazioni che fanno uso della crittografia sul Play Store 104
3.5.1 – Crypto 104
3.5.2 – Crypto SMS 105
3.5.3 – mSecure Password Manager 106
Capitolo 4 – Droidcare, un esempio di accesso a Personal Health Record 108
4.1 – Introduzione al progetto 108
4.1.1 – I requisiti 108
4.1.2 - Descrizione 109
4.1.2– La struttura del progetto 110
4.1.3– Gli strumenti utilizzati 111
4.2 – I package com.tesi.crypto.* 115
4.2.1 – Il motore 115
4.2.2 – Il Controller 117
4.2.3 – Gli algoritmi ed i meccanismi utilizzati 117
4.3 – HL7 e il relativo package 121
4.3.1 – Il modello di funzionamento del protocollo 121

4.3.2 – Le API di comunicazione 123
4.4 – L'organizzazione dell'applicazione, il package *com.tesi.droidcare* 128
4.4.1 – MainActivity 131
4.4.2 - SetupActivity 132
4.4.3 - LoginActivity 133
4.4.4 - SyncActivity 134
4.4.5 - DTRenderActivity 136
Conclusioni 138
Bibliografia 140
Ringraziamenti 143

Introduzione

Questo lavoro di tesi ha lo scopo di creare un'applicazione per la piattaforma mobile Android, orientata all'accesso a Personal Health Record. Il PHR a tutt'oggi sono informazioni non digitalizzate, il che impone ad un paziente di conservare delle copie cartacee di tutti gli esami, cure e patologie a cui si deve sottoporre; inoltre per avere accesso a queste informazioni sono necessari tempi burocratici molto lunghi. La digitalizzazione ed il conseguente accesso da dispositivi mobili, rappresenta una soluzione che annullerebbe completamente i tempi burocratici; inoltre risolve i problemi derivanti dall'eccessivo uso di materiale cartaceo, quali smarrimenti, deterioramento ed ingombro oltre che la protezione dei dati completamente assente. Il software inoltre andrebbe ad ovviare al problema di dover ritirare ricette, aggiornamenti delle cure e tutto ciò che non richiede un'interazione diretta con il personale sanitario. Portare tutti i dati clinici in formato digitale su un dispositivo mobile porta comunque dei problemi a livello di sicurezza, in quanto la cartella clinica è classificata come un dato estremamente sensibile. Il problema verrà risolto con l'uso della crittografia per la protezione dei dati, mentre si è pensato di utilizzare un meccanismo di verifica all'apertura dell'applicazione tramite connettività NFC.

La sincronizzazione dei dati è un argomento estremamente importante, i dati vanno scambiati secondo una grammatica ben precisa ed il messaggio deve essere validabile altrimenti c'è il rischio che vengano presi dati inconsistenti. La soluzione che possiamo adottare è l'utilizzo di uno standard americano per lo scambio di messaggi tra applicazioni mediche di livello enterprise: HL7. Questo standard definisce un modello di dati ed un protocollo di scambio di messaggi; il protocollo si adatta molto bene a qualsiasi tipo di applicazione, sia essa un semplice server http o TCP/IP o un'applicazione enterprise con un server JMS. Lo standard non prevede che il modello dei dati sia facilmente interpretabile dall'utente, pertanto sarà necessario mostrare all'utente un'interfaccia grafica estremamente semplice ed efficace; questo verrà implementato sfruttando le ampie capacità espressive, in termini di interfacce grafiche, messe a disposizione della piattaforma Android. La presentazione

fatta all'utente si differenzierà in base al tipo di dato che ci verrà fornito dal server remoto, quest'ultima cosa è molto importante per aumentare la capacità espressiva dell'applicativo e permettere all'utente di avere una visione specifica per ogni tipo di risorsa presente nella sua cartella clinica.

La trattazione presenta una panoramica della piattaforma di riferimento, scendendo successivamente nei particolari per quanto riguarda i componenti fondamentali che la caratterizzano. Il capitolo successivo ci illustra il contesto in cui si colloca l'applicativo, andando poi ad approfondire con casi di studio e presentando la situazione attuale delle applicazioni disponibili per la piattaforma Android.

La sezione dopo ci introduce alla crittografia, in un primo tempo, per poi andare a presentare la situazione corrente delle librerie e degli algoritmi crittografici che possiamo utilizzare sulla piattaforma di riferimento. L'ultima parte invece tratta dell'implementazione vera e propria del progetto, spiegando tutte le scelte progettuali fatte e motivandole.

Capitolo 1 – Android

Android è un insieme di software open-source per dispositivi mobili che include un sistema operativo, componenti applicativi fondamentali standard e strumenti middleware che eseguono funzioni d'intermediazione tra diverse applicazioni. Fu inizialmente sviluppato dalla Startup Android, acquisita successivamente da Google. I cofondatori di Android iniziarono a lavorare per Google e svilupparono una piattaforma basata su kernel Linux. Il 5 novembre 2007 il consorzio di produttori Open Handset Alliance (di cui Google è capofila) presentò pubblicamente Android alla stampa. Il 12 novembre 2007 l'OHA ha reso disponibile il *Software Development Kit* (SDK) che fornisce gli strumenti e le API (le *Application Programming Interface* sono funzioni e classi messe a disposizione dei programmatori) necessarie per iniziare a sviluppare applicazioni sulla piattaforma Android utilizzando il linguaggio di programmazione Java. Esso include: gli strumenti di sviluppo, le librerie, un emulatore del dispositivo, la documentazione, alcuni progetti di esempio, tutorial e altro. È installabile su qualsiasi computer che usi come sistema operativo Windows, Mac OS X o Linux. L'ambiente di sviluppo integrato (*Integrated Development Environment* - IDE) ufficialmente supportato è Eclipse, per il quale è fornito un plug-in (nelle versioni più recenti dell'SDK è incluso direttamente l'IDE preconfigurato). Si potrebbe identificare l'utilizzo di Android come il solo utilizzo di tale sistema operativo all'interno di telefoni cellulari, ma un impiego molto interessante si ha anche in dispositivi che non sono nemmeno telefoni, come lettori di e-book, netbook e lettori multimediali.

1.1 – Profilo del Sistema Operativo

1.1.1 – Breve storia dello sviluppo con accenni al rilascio delle versioni più significative

Il 23 settembre del 2008 viene rilasciata la versione 1 di Android, ma ne vedremo la sua reale applicazione solo il successivo 22 ottobre con il lancio sul mercato americano del dispositivo T-Mobile G1, il primo in assoluto a essere venduto con il sopracitato sistema. Pur cercando di non dilungarmi molto sui dettagli di tutte le singole versioni, ritengo significativo sottolineare qualche

curiosità, come il fatto che alcuni rilasci siano strettamente collegati a un modello specifico di telefono. Pur mantenendo una certa retro compatibilità con le vecchie versioni, Android ha subito molteplici aggiornamenti e revisioni, alle più caratteristiche delle quali, è stato attribuito un secondo nome; ad esempio la versione 1.5 uscita il 13 aprile 2009 è nota come "CupCake". La successiva versione 1.6 del 16 settembre 2009, soprannominata "Donut", è un'evoluzione della 1.5 e supporta reti CDMA (protocollo di accesso multiplo a canale condiviso di comunicazione, molto diffuso nelle reti wireless) e risoluzioni grafiche multiple. Rilevanti cambiamenti si hanno nelle versioni 2 e 2.1 della fine del 2009, che vanno sotto il nome di "Eclair" che presentano, tra le novità, il supporto per più account e un meccanismo di riconoscimento vocale, ultimamente preso in considerazione anche dalla concorrenza. Il 20 maggio 2010 è stata rilasciata la versione 2.2 con il nome in codice di "Froyo" che, oltre ad aggiornamenti molto rilevanti, segna la scelta dei produttori di entrare nel mercato delle piattaforme aziendali, oltre che in quello consumer; mentre il successivo aggiornamento "Gingerbread", nome in codice della versione 2.3, servirà soprattutto come porta d'ingresso per il mercato dei giochi, ma anche per mettersi in pari con la concorrenza in qualche aspetto non del tutto secondario, come ad esempio la gestione del copia-incolla.

Nel gennaio 2011 viene rilasciata la versione 3 del sistema, ovvero "Honeycomb" che segna il tentativo di sbarco sui dispositivi tablet PC (questa versione di Android infatti è stata rilasciata unicamente per quel segmento di dispositivi), con risultati probabilmente inferiori alle aspettative, ma porta anche qualche cambiamento nell'estetica, cambiamento che verrà ripreso nella versione 4.0, quella attuale, soprannominata "Ice-Cream Sandwich", presentata il 19 ottobre 2011 congiuntamente al cellulare "Samsung Galaxy Nexus", primo con la nuova versione preinstallata. Quest'ultima versione è stata rilasciata con l'intento di "uniformare" tablet e smartphone, porta significativi cambiamenti sia nella grafica visuale che nel comparto di nuove e moderne applicazioni software; presenta anche a livello di sviluppo un SDK "unificato" e rende lo sviluppo di applicazioni per entrambi i segmenti di dispositivi molto più agevole.

1.1.2 – Caratteristiche Principali

Android è basato su una versione del kernel **Linux 2.6 (Android 4.x è basato su kernel Linux 3.x)** che, come nel caso del Sistema Operativo Desktop, rappresenta il livello di astrazione tra l'insieme di applicativi software e l'hardware sottostante. L'essere costruito a partire da software open-source, a differenza di altri sistemi simili ad esempio iOS di Apple, ne fa un punto di forza a favore dell'enorme comunità di sviluppatori che ha a disposizione lo stesso codice, aperto al pubblico, su cui lavorare e confrontarsi e apportare miglioramenti. L'apertura di Android è molto apprezzata anche dai costruttori di hardware che, non a caso, hanno scelto di formare tra loro delle alleanze che, sotto forma di consorzi tra produttori di tablet e smartphone, operatori nel campo Mobile e fornitori di software, permettono loro di condividere informazioni e di concordare o definire alcuni standard di base. Anche lo store di Google, che rappresenta la risorsa in rete più immediata dalla quale reperire le applicazioni di proprio gradimento, è più aperto di quello di Apple. Mentre il secondo definisce procedure molto precise per il caricamento di nuove applicazioni, Google permette agli sviluppatori di creare e caricare tutte le applicazioni che vogliono, comprese quelle non ancora complete o in una fase di testing non molto avanzata. Non si fa differenza tra le applicazioni principali, fornite all'acquisto del dispositivo, e quelle prodotte da terze parti, le quali hanno comunque accesso a tutte le funzionalità delle precedenti e possono utilizzare API di tutti i livelli. In questo modo viene data la possibilità a tutti gli sviluppatori di creare applicazioni nuove e innovative e di utilizzare le risorse in maniera più ottimizzata. Ad esempio, si possono combinare informazioni prelevate dal web con dati presenti sul telefono cellulare, come i contatti, l'agenda o la collocazione geografica, per fornire all'utente un'esperienza ancora più accattivante. Abbiamo a disposizione un browser basato sul motore di ricerca open-source WebKit; grafiche ottimizzate basate su stabili librerie sia 2D che 3D (openGL); supporto per la gestione dei Database e di una moltitudine di altri contenuti. A seconda delle caratteristiche del dispositivo, abbiamo supporto per reti di telefonia GSM e di comunicazione Bluetooth, EDGE, 3G, LTE e WiFi; e, se presenti, per la gestione di strumenti

hardware quali fotocamera, GPS, bussola, accelerometro e giroscopio. Il sistema operativo Android non limita il processore ad eseguire una singola applicazione alla volta. Il sistema **multi processo** gestisce interazioni e priorità di vari agenti all'interno di una singola applicazione e dell'intera struttura in generale, di modo che alcune operazioni possono essere eseguite in background mentre l'utente interagisce con il processo in primo piano. Una caratteristica interessante dell'esecuzione contemporanea di più processi, viene alla luce utilizzando le comuni **App Widgets**, piccole applicazioni che possono essere incorporate in altre (come la schermata principale o la barra delle notifiche), per interagire con l'utente mentre altre applicazioni sono in esecuzione, così da permettere di elaborare dei semplici eventi come l'aggiornamento della temperatura esterna o l'avvio di un brano musicale.

1.2 – Architettura Android

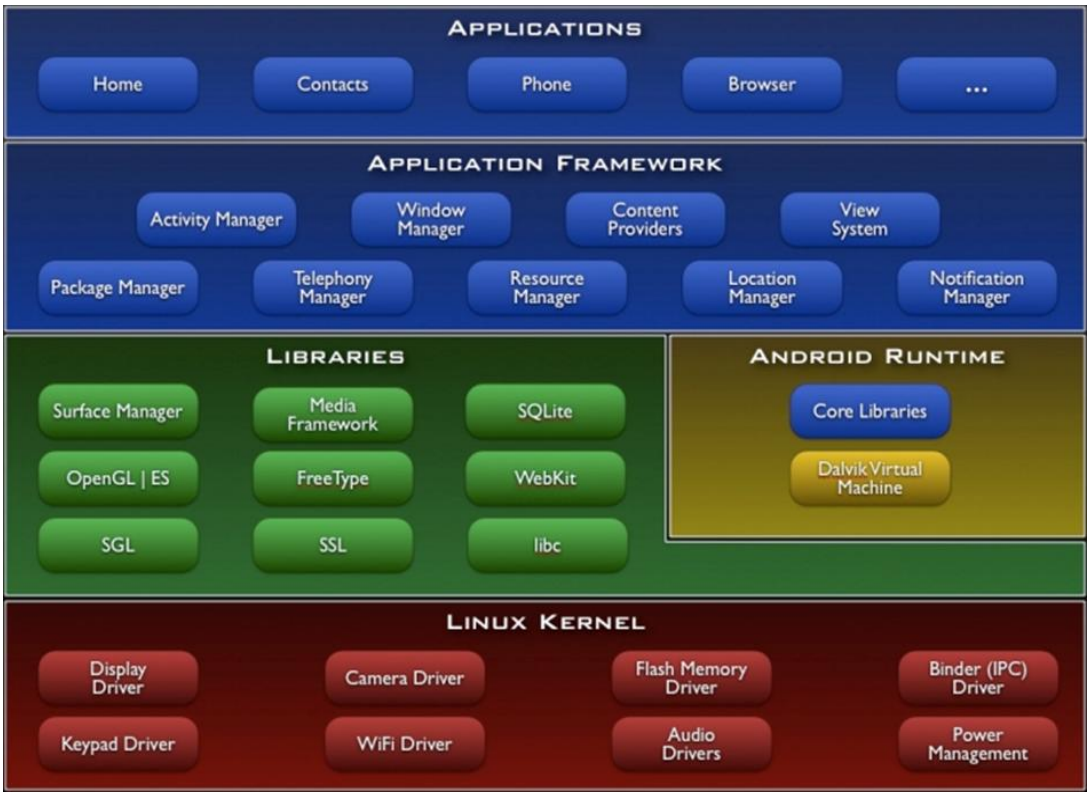

Figura 1 - L'architettura del sistema operativo

1.2.1 – Panoramica

Android è fornito con un set di **applicazioni** di base, che comprende programmi per gestire e-mail, SMS, calendario, mappe, browser, contatti e altro. Tutte le applicazioni sono scritte in linguaggio Java. Grazie al particolare **Application Framework**, accennato in precedenza, gli sviluppatori hanno pieno accesso all'insieme di API messe a disposizione dal sistema e usate dalle applicazioni di base. Questa particolare architettura è progettata per semplificare il riutilizzo dei componenti che ogni applicazione può rendere pubblici a beneficio di altre che possono beneficiarne (il tutto è naturalmente regolato da politiche di privacy e di permessi che garantiscono sicurezza alla condivisione dei dati). Questo meccanismo inoltre consente all'utente di sostituire i componenti standard con versioni personalizzate. Le peculiarità alla base delle applicazioni e messe a disposizione dei progettisti sono molteplici: un ricco gruppo di *View* estendibili e personalizzabili che comprendono caselle di testo, pulsanti, liste e quant'altro; dei *Content Provider* che permettono alle applicazioni di accedere a dati da altre applicazioni (come i Contatti), o di condividere i propri dati; un *Resource Manager* che consente l'accesso e l'integrazione di risorse non in forma di codice ma di stringhe localizzate piuttosto che di immagini, grafica o file di layout; un *Notification Manager* che permette a tutte le applicazioni di mostrare notifiche nella barra di stato; un *Activity Manager* che gestisce il ciclo di vita della Attività, componenti fondamentali delle applicazioni. Android comprende inoltre un completo set di **Libraries** (in C/C++) usate dai propri componenti. Tra queste possiamo notare le *Media Libraries* multimediali che abilitano alla riproduzione di molti popolari formati audio e video (MPEG4, H.264, MP3, AAC, AMR, JPG, e PNG). Le *3D Libraries* che usano API OpenGL e accelerazione grafica hardware se disponibile. L'*SQLite* invece è una versione leggera e dedicata a sistemi mobili del più famoso SQL (*Structured Query Language*), in grado di interrogare, modificare e gestire dati memorizzati in un sistema di Database relazionali. Sono invece le *Core Libraries* a fornire la maggior parte delle funzionalità disponibili nelle librerie di base del linguaggio di programmazione Java. Questa architettura necessita di una macchina virtuale che esegua il

ByteCode (linguaggio intermedio più astratto del linguaggio macchina) generato dalla compilazione di codice Java (come buona parte delle architetture basate su questo linguaggio di programmazione); per questo scopo è stata adattata la tradizionale JVM (Java Virtual Machine) alla più leggera *Dalvik Virtual Machine*, ottimizzata per sfruttare la poca memoria presente nei dispositivi mobili. Questa consente di far girare diverse istanze della macchina virtuale contemporaneamente ed in maniera efficiente nascondendo al sistema operativo sottostante l'amministrazione della memoria e dei *thread* (sono la più piccola unità di elaborazione gestibile dal sistema operativo; tutte le applicazioni ne hanno almeno uno).

1.2.2– Application Framework e Libraries

A occupare il livello immediatamente sottostante a quello delle applicazioni native, troviamo l'Application Framework, un insieme di blocchi di alto livello utili alla creazione di nuove applicazioni. L'Application Framework è preinstallato nei dispositivi Android e consiste nelle seguenti componenti:

- **Activity Manager** provvede al ciclo di vita delle applicazioni e mantiene uno Stack condiviso delle attività per permettere la navigazione tra di esse.
- **Content Provider** incapsulano i dati per poterli scambiare tra le diverse applicazione.
- **Location Manager** informa il dispositivo della propria locazione fisica e gestisce tutte le risorse quali il GPS.
- **Notification Manager** notifica importanti informazioni provenienti dalle applicazioni installate sul dispositivo (come ad esempio l'arrivo di un messaggio di testo) senza tuttavia interrompere ciò che in quel momento l'utente sta facendo.
- **Package Manager** permette alle applicazioni di conoscere quali altri package sono installati sullo stesso dispositivo.
- **Resource Manager** gestisce l'accesso delle applicazioni alle proprie risorse dati e multimediali.
- **Telephony Manager** istruisce le applicazioni sui servizi di telefonia dei quali è dotato il dispositivo e le abilita ad effettuare e ricevere chiamate.

- **View System** si occupa degli elementi presenti nell'interfaccia utente di un'applicazione, della loro posizione e degli eventi eventualmente generati dall'interazione dell'utente con essi; si occupa inoltre di popolare le view all'avvio dell'applicazione come specificato nel codice dell'applicazione.
- **Window Manager** è responsabile dell'organizzazione grafica delle finestre sullo schermo, ne gestisce le dimensioni e gli assegna una superficie di azione.

  Le componenti dell'Application Framework, per adempiere ai propri compiti, si basano su delle librerie native di sistema, alle quali gli sviluppatori possono accedere tramite l'uso delle API a loro disposizione oppure tramite il *Native Development Kit* (NDK). Le librerie sono sviluppate in linguaggio nativo C/C++ e consistono nelle seguenti componenti:

- **Free Type** supporta il rendering di bitmap e vector font.
- **Libc** è una libreria BSD (*Berkeley Software Distribution* indica una variante originaria di Unix sviluppata presso l'Università di Berkeley) che implementa la versione standard C di sistema, ma adattata e messa a punto per dispositivi embedded Linux.
- **LibWebCore** fornisce supporto al motore di ricerca preinstallato in Android e alla visualizzazione delle pagine web. È la medesima usata dai noti browser "Google Chrome" e "Apple Safari".
- **Media Framework** è di supporto alla registrazione e riproduzione di molti popolari formati audio e video. I formati supportati sono MPEG4, H.264, MP3, AAC, AMR, JPEG e PNG e derivati.
- **OpenGL/ES** utilizzano Accelerazione 3D (se disponibile come hardware del dispositivo) e sistemi software per le rappresentazioni grafiche in 3D. Sono un sottoinsieme di API ricavate del noto progetto open source OpenGL, pensate appositamente per dispositivi dalle limitate risorse elaborative avendo un consumo massimo di memoria di 64 MB.
- **SGL** rappresentano il motore grafico 2D.

- **SQLite** fornisce il supporto per un moderno, potente e leggero database relazionale interrogabile da tutte le applicazioni. È la medesima usata "Mozilla Firefox" e "Apple IPhone" per la memorizzazione persistente.
- **SSL** si occupa della sicurezza per la comunicazione cifrate in rete, fornendo un supporto basato su *Secure Sockets Layer*.
- **Surface Manager** per la gestione della visualizzazione grafica.

1.2.3 – Android Runtime Environment

L'ambiente di esecuzione in Android prevede delle **Core Libraries** che implementano una versione ridotta delle librerie Java 5, e la **Dalvik Virtual Machine** una macchina virtuale differente dalla *Oracle Java Virtual Machine* e basata sui registri del processore invece che sullo stack, con impiego di minore memoria (in termini di stack), ma al contempo con istruzioni più lunghe del normale (il tutto è stato progettato per adattarsi ad una differente architettura sottostante, basata su processori di tipo RISC invece che di tipo CISC). Un'importante differenza tra le due macchine virtuali è il codice eseguito: la Dalvik VM esegue file nel formato Dalvik Executable (.dex), ottimizzato per usare al minimo lo spazio disponibile in memoria. Bisogna notare per l'appunto che la Dalvik Virtual Machine essendo una macchina virtuale non-java, non manipola direttamente bytecode Java, ma la trasformazione in formato DEX eseguibile viene mutuata da uno strumento interno ad Android chiamato "**dx**". Il compatto formato eseguibile (circa 30% in meno di un *jar* non compresso, il formato eseguibile Java) è progettato per essere adatto a sistemi con limitate risorse in termini di memoria e di velocità del processore.

La scelta, operata da parte dei progettisti Android, di utilizzare linguaggio Java *puro*, è nettamente controcorrente a quelle fatte dai diretti concorrenti nell'ambito dei dispositivi mobili; per progettare applicazioni mobili per Apple, è necessario il sistema operativo Mac OS X e il linguaggio da utilizzare è l'*Objective-C*, un'estensione ad oggetti del linguaggio C; per sviluppare un'applicazione Nokia è necessario impiegare un sotto-linguaggio del C++ (Qt) mentre per sviluppare su Windows Phone è neccessario ricorrere a tutto il

campionario di tecnologie proprietarie Microsoft. Il vantaggio di questa decisione è rappresentato soprattutto dalla presenza nativa di elementi da non dover ridefinire, come, ad esempio, le specifiche del linguaggio, che saranno quelle rilasciate dalla Sun, il compilatore e gli strumenti di redazione del codice; in questo modo, il processo di sviluppo di una applicazione Android mostra molti punti in comune con la realizzazione di applicazioni desktop in Java.

1.2.4 – Inizializzazione della piattaforma

Per quanto il suo hardware sia fortemente ridotto, un sistema di elaborazione mobile presenta forti analogie con il più complesso sistema desktop e anche se i sistemi operativi delle piattaforme mobili sono numerosi e spesso proprietari, il processo che porta all'avvio del kernel del sistema è molto simile indipendentemente dalla piattaforma che il sistema operativo deve controllare. La procedura di start up del kernel di Android è molto simile a quella seguita da un kernel Linux su un sistema desktop, o in generale su altri sistemi. L'inizializzazione dello spazio utente avviene tramite il processo **init** che nell'albero delle cartelle di Android è situato in *system/core/init*; si tratta del primo processo a girare nello spazio utente. Il processo iniziale imposta tutto ciò che è necessario a eseguire il sistema: vengono avviate le protezioni della memoria, le memorie cache, gli algoritmi di scheduling e i **daemon** che, in background, senza il controllo diretto dell'utente, gestiscono autonomamente le interfacce hardware di basso livello (usb, adb, debugger, radio). Viene, inoltre, creato **Zygote**, un processo di sistema che si fa carico di configurare e avviare un'istanza della Dalvik Virtual Machine utile ad eseguire il **System Server**. Il processo **Runtime** inizializza il **Service Manager** che comprende alcuni componenti fondamentali dell'Application Framework visti in precedenza utili per la comunicazione tra le vari parti del sistema. Dopo l'avvio dei primi due processi che gestiscono rispettivamente grafica e audio, tutti gli altri componenti del sistema vengono avviati.

1.2.5 – La sicurezza

L'architettura Android include un modello di sicurezza che impedisce alle applicazioni, agenti per utenti specifici, di compiere operazioni che potrebbero arrecare danni o involontariamente modificare parametri di altre applicazioni, del kernel Linux o di altri utenti. Questo modello amplia quello preesistente nell'architettura Linux, basato sugli ID (codici identificativi) degli utenti e dei gruppi. Come impostazione predefinita, le applicazioni non possono leggere o scrivere informazioni riguardante gli utenti, come ad esempio contatti, messaggi o e-mail piuttosto che accedere alla videocamera o a Internet. Le applicazioni che necessitano di questo tipo di dati sensibili possono pervenire ad essi, ma solo se gli sono stati forniti i giusti permessi (da approvare in fase di installazione dell'applicazione).

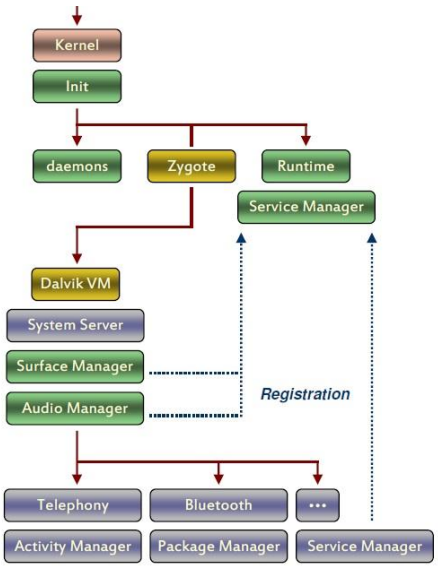

Figura 2 - Lo schema dei processi

Android può gestire la sicurezza in vari modi, di solito automaticamente basandosi sull'acquisizione di particolari certificati, o con l'ausilio dell'utente che decide quali e se concedere o revocare permessi alle singole applicazioni.

1.3 – Struttura delle applicazioni Android

L'architettura di un'applicazione Android è differente da quella adottata per le normali applicazioni in esecuzione su un classico Personal Computer.

L'approccio architetturale basato sul framework limita la libertà degli sviluppatori, imponendo loro talvolta determinate modalità d'azione per molteplici casi di utilizzo. Ogni applicazione è un agglomerato di componenti che comunicano tra di loro scambiandosi *intenti* che saranno descritti, insieme agli altri elementi, in un particolare file che rappresenta il *manifesto* dell'intera applicazione la quale sarà racchiusa in un unico pacchetto software. Le diverse componenti di un'applicazione (Activities, Services, Content Provider, and Broadcast Receiver) sono eseguite in un processo Linux e gestite da Android che ne coordina le interazioni. Esse non sono tutte indispensabili (un'applicazione potrebbe essere composta solo da attività o da attività e servizi), ma possono condividere tra di loro un set di risorse che include Database, preferenze, file (che possono essere di sistema e non) e processi. Questo approccio basato sulle componenti, permette un facile riutilizzo, da parte delle nuove applicazioni, di risorse esistenti, riducendo in tal modo l'impiego della memoria complessiva di per sé limitata nei dispositivi mobili. Inoltre, il Sistema Operativo può istanziare la componente richiesta solo quando diventa strettamente necessaria per quella particolare applicazione in esecuzione.

1.3.1 - Activity e Intent

Un'**Activity** è un componente con il quale l'utente può direttamente interagire tramite un'interfaccia grafica. Ad esempio l'applicazione Contatti contiene delle attività per visualizzare o creare un nuovo contatto; l'applicazione Telefono include un'attività per comporre un numero, così come l'applicazione Messaggi ne comprende una per comporre un messaggio di testo. Le diverse Activity sono descritte da sottoclassi della classe **android.app.Activity**; quest'ultima è una sottoclasse indiretta di *android.content.Context*, classe astratta i cui metodi consentono alle applicazioni di accedere alle informazioni inerenti al loro ambiente e permettono di eseguire operazioni contestuali come il lancio di un'attività, di un nuovo servizio, di un *Broadcast Intent* o l'apertura di file privati. Le Activity di solito sovrascrivono (@Override) diversi metodi della classe principale. Questi metodi, che esamineremo a breve, prendono il nome di *lifecycle callback methods*, che Android usa per controllare le

interazioni dell'applicazione con la corrente Activity, durante il ciclo di vita di quest'ultima:

- **onCreate(Bundle)** viene chiamata quando l'Attività è istanziata per la prima volta. Inizializza l'interfaccia utente, gli eventuali thread e parametri globali. Gli viene passato un oggetto *android.os.Bundle* nel caso esista uno stato precedentemente congelato. Questo metodo è sempre seguito da *onStart()*.
- **onStart()** è chiamata poco prima che l'Attività diventi visibile all'utente. Android invoca *onResume()* se l'Attività sta per tornare in primo piano (foreground), *onStop()* se sta per essere nascosta (background).
- **onRestart()** viene chiamata dopo che l'Attività è stata arrestata e prima che sia riavviata. È seguita dalla *onStart()*.
- **onResume()** viene chiamata quando l'Attività inizia l'interazione con l'utente. A questo punto l'Attività è in cima allo stack delle Attività e l'input dell'utente ha effetto su di essa. Se l'Attività dovesse essere messa in pausa, Android chiamerebbe successivamente la *onPause()*.
- **onPause()** viene chiamata quando Android sta per riprendere un'Attività precedente o quando il dispositivo viene messo in standby. Questo metodo è generalmente utilizzato per rendere persistenti le modifiche non salvate o interrompere operazioni onerose in termini di consumi. Logicamente il codice da eseguire deve essere leggero perché l'Attività da recuperare non sarà ripresa finché questo metodo non termina i propri compiti. È seguito da *onResume()* se l'Attività deve ritornare in foreground, da *onStop()* se deve diventare invisibile all'utente.
- **onStop()** è chiamata quando l'Attività deve passare in background e non essere più visibile all'utente perché ne abbiamo appena ripreso (o lanciato) un'altra che ha soppiantato la precedente o perché quella corrente è stata distrutta. Android chiama *onRestart()* se l'Attività torna ad interagire con l'utente o *onDestroy()* se l'Attività non è più necessaria così da poter essere distrutta.
- **onDestroy()** è l'ultima chiamata che riceviamo prima che la nostra Attività venga definitivamente distrutta. Questo può avvenire se l'Attività ha terminato il proprio corso regolarmente (è stata riscontrata la chiamata

finish()) o se il sistema la sta distruggendo per recuperare spazio e le risorse. Con questo metodo, l'Attività rilascerà tutte le risorse allocate rendendole disponibili al sistema.

L'intero ciclo di vita di un'Attività è rappresentato da tutto ciò che c'è tra la chiamata onCreate(Bundle) e la onDestroy(). La durata visibile di un'Attività, invece, è l'intervallo che intercorre tra le chiamate onStart() e onStop(). Durante questa fase l'utente può comunque vedere l'Attività sullo schermo anche se non dovesse poter direttamente interagire in primo piano con essa. Questi ultimi due metodi possono essere invocati più volte alternando la visibilità o meno dell'Attività. Il ciclo di vita in foreground dura dalla chiamata onResume() fino alla onPause(). In questo stato, l'Attività è visibile in primo piano all'utente che sta interagendo con essa. Anche applicazioni molto semplici potrebbero richiedere più di un'Attività per soddisfare funzionalità diverse. Al lancio dell'applicazione viene lanciata l'Attività principale (main_activity) che può avviare altre Attività, di solito in seguito a un'interazione con l'utente, che metteranno quella principale in pausa. Il modo più pratico per attivare altri componenti dell'applicazione, facendo in modo che essi si avviino con determinati parametri, è quello di utilizzare gli Intenti.

Gli **Intent** sono stringhe che identificano l'operazione da eseguire o, in caso di trasmissioni broadcast, forniscono la descrizione dell'evento verificatosi. Questi messaggi sono implementati come istanze della classe **android.content.Intent**. Gli oggetti Intent forniscono informazioni di interesse al componente che riceve l'intento (ad esempio le azioni da intraprendere e i dati su cui agire), ma anche ad Android. Principalmente può contenere i seguenti elementi:

- **Component name** è il nome del componente che dovrà gestire l'intento. Il nome del componente è impostato da *setComponent(), setClass(),* o *setClassName()* e letto da *getComponent()*. Questo è un parametro opzionale; se presente, l'oggetto Intent è consegnato a un'istanza della classe designata;

altrimenti Android utilizzerà altre informazioni dell'oggetto Intent per individuare un ricevente adatto (o istanziarne uno se necessario).

- **Action** è una stringa che denomina l'azione da eseguire o, in caso di broadcast, viene riportata l'azione occorsa; tipicamente un verbo. La classe Intent definisce un certo numero di azioni standard:

Constant	**Target component**	**Action**
ACTION_CALL	activity	Initiate a phone call.
ACTION_EDIT	activity	Display data for the user to edit.
ACTION_MAIN	activity	Start up as the initial activity.
ACTION_SYNC	activity	Synchronize data on a server with data on the mobile device.
ACTION_BATTERY_LOW	broadcast receiver	A warning that the battery is low.
ACTION_HEADSET_PLUG	broadcast receiver	A headset has been plugged into the device, or unplugged from it.

ACTION_SCREEN_ON	broadcast receiver	The screen has been turned on.
ACTION_TIMEZONE_CHANGED	broadcast receiver	Changed time zone, change time.

Nella maggior parte dei casi, l'azione determina altri importanti parametri come in particolare dati e campi extra. In un oggetto Intent, l'azione è impostata tramite il metodo *setAction()* e letta con *getAction()*.

- **Data** contiene informazioni riguardanti i dati con cui si andrà ad interagire come, ad esempio, l'URI e il tipo MIME (*Multipurpose Internet Mail Extensions*). Azioni diverse sono accoppiate con specifici tipi di dati: se l'azione fosse *ACTION_EDIT*, il campo dati conterrebbe la URI del documento da visualizzare per l'editing; se fosse *ACTION_CALL*, il campo dati sarebbe contatto telefonico; in egual modo se l'azione fosse *ACTION_VIEW*, il campo di dati sarebbe un URI http (un indirizzo Internet). Il tipo di dati può anche essere impostato in modo esplicito nell'oggetto Intent tramite i metodi *setData()* che specifica i dati solo a livello di URI, *setType()* che ne specifica il tipo MIME e *setDataAndType()* che specifica entrambi. L'URI e il tipo del dato possono essere letti rispettivamente grazie ai metodi *getData()* e *getType()*.
- **Category** è una stinga contenente informazioni aggiuntive sul tipo di componente che è chiamato a gestire l'evento. Vediamo alcuni esempi di categorie standard:

Constant	**Meaning**
CATEGORY_BROWSABLE	The target activity can be safely invoked by the browser to display data referenced by a link — for example, an e-mail message.

CATEGORY_GADGET	The activity can be embedded inside of another activity that hosts gadgets.
CATEGORY_HOME	The activity displays the home screen, the first screen the user sees when the device is turned on or when the *Home* button is pressed.
CATEGORY_LAUNCHER	The activity can be the initial activity of a task and is listed in the top-level application launcher.
CATEGORY_PREFERENCE	The target activity is a preference panel.

Per aggiungere o eliminare una categoria si usano rispettivamente i metodi *addCategory()* e *removeCategory(),* per ottenere il set di categorie di un oggetto invece si usa *getCategories().*

- **Extras** sono un set di coppie chiave-valore che aggiungono informazioni aggiuntive e sono da consegnare al componente che gestisce l'Intent. È logico che anch'essi siano abbinati a determinati tipi di azioni: con un intento del tipo *ACTION_TIMEZONE_CHANGED*, avremo un extra composto da "tempo-zona" che identifica il nuovo fuso orario; con un altro Intent del tipo *ACTION_HEADSET_PLUG* avremo uno "stato" extra che ci informa se l'auricolare è collegato o meno e un "nome" che ci mette al corrente del tipo di auricolare usato. Gli extras vengono inseriti e prelevati tramite metodi *put…()* e *get…()* secondo il tipo di dato trattato. Per trattare tutto il set di extras insieme si possono usare i metodi *putExtras()* and *getExtras().*
- **Flags** informano Android su come avviare un'attività e trattarla dopo averla lanciata. I flags sono definiti all'interno della classe Intent.

  Un aspetto molto interessante da sottolineare quando parliamo di Intenti, è l'uso della classe **Intent Filter**. Un Intent Filter è utilizzato per informare il

sistema riguardo a quali Intent un'Activity, un Service o un Broadcast Receiver può ricevere e gestire e come li gestisce. Può essere inserito dinamicamente tramite la classe *IntentFilter* o preferibilmente in maniera statica, nel *manifesto* xml dell'applicazione (utilizzando i tag <*intent-filter*>). È così chiamato perché filtra gli Intent impliciti lanciati, lasciando passare solo quelli voluti.

Un Intent *esplicito* è sempre consegnato alla classe obiettivo e non prevede la consultazione degli Intent Filter. Un Intent *implicito* invece, viene consegnato ad un componente se uno dei suoi Intent Filter, confrontato con l'Intent, restituisce esito positivo. Sono tre i campi di un oggetto Intent che possono essere consultati per stabilire se il confronto con un Intent Filter ha avuto successo: action, category e data. Se un Intent supera con successo il confronto con gli Intent Filter di più Activity, all'utente può essere chiesto quale componente attivare, mentre se non vi è alcun elemento trovato viene sollevata un'eccezione.

1.3.2 - Services

Un **Servizio** è un componente eseguito in background senza una scadenza temporale; non fornisce all'utente un'interfaccia grafica con cui interagire ed è invisibile a quest'ultimo. Ci sono Servizi a livello locale, che sono eseguiti nello stesso processo dell'applicazione che l'ha richiesto, Servizi remoti che vengono eseguiti in un thread separato, utili per l'implementazione delle comunicazioni inter-processo. Un Servizio non è un processo separato dall'applicazione, né un thread distinto, ma esso permette alle applicazioni di esplicare cosa ha intenzione di fare anche quando l'utente non interagisce direttamente con essa, e di comunicare le sue peculiarità ad altre applicazioni. Possiamo addure come esempio un Servizio che riproduce brani musicali, è naturale che l'utente si aspetti che la musica continui a suonare anche dopo che l'applicazione non è più in primo piano; o un Servizio di aggiornamento delle applicazioni in background, molto più discreto di un'Attività con cui interagire. Un Servizio in sostanza può trovarsi in due stati diversi: *avviato*

quando viene chiamata una particolare funzione di start(*startService()*), esso viene eseguito in background per un tempo indefinito e continua la sua esecuzione anche se la componente che l'ha generato viene distrutta; *vincolato* (bound) quando un componente (o più) di un'applicazione si lega ad esso chiamando una particolare funzione di bind, in questo caso i componenti interagiscono con il Servizio inviando richieste e ottenendo risultati, ma questo funzionerà solo fino a quando ci sarà un componente legato ad esso.

I Servizi sono descritti da sottoclassi della classe **android.app.Service** che è una sottoclasse indiretta di *Context.* I Servizi, come le attività, sovrascrivono diversi *lifecycle callback methods* che Android userà durante la vita del Servizio:

- **onStartCommand()** viene chiamata quando un altro componente, ad esempio un'attività, richiede che il Servizio venga avviato chiamando *startService().* Implementando questa funzione diventa nostra responsabilità interrompere il Servizio quando ha terminato il suo incarico invocando *stopService()* o *stopSelf()*. Se si desidera che il Servizio sia vincolato non è necessario ridefinire questo metodo.
- **onBind()** viene chiamata quando un componente desidera legarsi al Servizio chiamando *bindService()*. Per questo metodo è necessario fornire un'interfaccia di ritorno (*IBinder*) che i clienti possano utilizzare per comunicare con il Servizio. È obbligatorio implementare questo metodo ma se non si vuole che il Servizio sia legato (o legabile) ad alcun componente, esso deve restituire un valore "null".
- **onCreate()** è chiamata esclusivamente quando il Servizio viene creato, per permetterne le procedure di setup, viene effettuata prima delle chiamate *onStartCommand()* oppure *onBind()*.
- **onDestroy()** viene chiamata quando il Servizio non viene più utilizzato e sta per essere distrutto. Dobbiamo implementare questo metodo per essere certi di distruggere tutte le risorse allocate dal servizio (ad esempio listeners, threads, receivers, etc).

Il Sistema Operativo distrugge forzatamente il Servizio nel caso necessiti di risorse per tenere in vita l'attività con la quale l'utente sta interagendo in quel momento.
Naturalmente se il Servizio è legato a un'attività in primo piano le probabilità che esso sia distrutto sono molto basse, in caso contrario, se esso è stato avviato precedentemente, con lo scorrere del tempo esso perderà di importanza per il sistema che alla fine sarà più prepenso a terminarlo. In ogni caso Android riattiva il Servizio, che ha autonomamente chiuso, appena le risorse di memoria tornano di nuovo disponibili e adeguate all'esecuzione.
I Servizi locali, di solito, sono avviati da un Context con **ComponentName startService(Intent intent)** metodo che ritorna un *android.component.ComponentName* se il Servizio è stato correttamente avviato, o null se si sono riscontrati problemi durante la creazione.

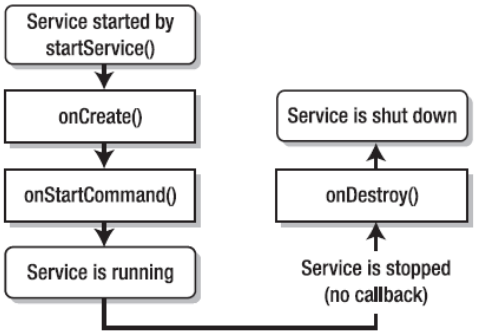

Figura 3 - Il ciclo di vita di un servizio

Nel ciclo di vita mostrato in figura, la chiamata *startService(Intent)*, si riduce in una chiamata *onCreate()* e in una successiva **int onStartCommand(Intent intent, int flags, int startId)** i cui argomenti rappresentano rispettivamente l'Intento passato a *startService(Intent)*, dati addizionali e un intero univoco che descrive questa specifica chiamata di avvio, che volendo si può usare per chiudere il Servizio passandolo come argomento a *boolean stopSelfResult(int startId)*.

I Servizi remoti vengono avviati da un Context tramite il metodo **boolean bindService(Intent service, ServiceConnection conn, int flags)**.

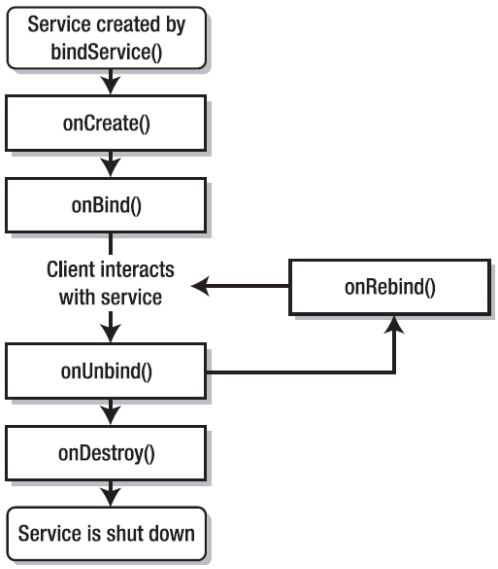

Figura 4 - Il bind di un servizio

La chiamata *bindService(Intent, ServiceConnection, int)*, si traduce in una chiamata *onCreate()* seguita da una chiamata *onBind(Intent)* che restituisce il vero canale di comunicazione (un'istanza di una classe che implementa *android.os.IBinder*).

1.3.3– Broadcast Receiver

I **Broadcast Receiver** hanno il compito di intercettare e reagire ai messaggi broadcast, che rappresentano quelle informazioni generate da una unità trasmittente e rivolte a un insieme di sistemi riceventi non definito a priori. Molti di questi messaggi sono generati dal codice di sistema, come ad esempio il cambiamento del fuso orario o lo scarso livello di batteria o il cambiamento di connettività. I messaggi broadcast possono anche essere generati da applicazioni per monitorare aspetti generali del sistema o di altre applicazioni, come per esempio lo squillare del telefono o l'arrivo di un messaggio di testo. Se un'applicazione ha bisogno di intercettare questo tipo di eventi globali, deve necessariamente registrare dei Broadcast Receiver. Broadcast Receiver sono descritti da sottoclassi della classe astratta **android.content.BroadcastReceiver,** e sovrascrivono il metodo astratto **void**

onReceive(Context context, Intent intent). Vediamone subito un semplice esempio:

Possiamo registrare dinamicamente un Broadcast Receiver tramite la chiamata **Intent registerReceiver (BroadcastReceiver receiver, IntentFilter filter)**. In questo modo ad esso verrà notificato ogni messaggio (o altro) per ogni Intent che corrisponde all'IntentFilter passatogli come parametro. Con il metodo **void unregisterReceiver (BroadcastReceiver receiver)**, invece, annulliamo la precedente registrazione.

È possibile, in alternativa, dichiarare un Broadcast Receiver nel file xml Manifest (vedremo nei prossimi paragrafi com'è organizzato questo file) tramite l'elemento <**receiver**> nel quale descriveremo il nome della classe che implementa il Receiver ed esporremo gli IntentFilter; in questo modo l'applicazione non ha bisogno di essere in esecuzione per essere avviata, quando l'evento si verifica, l'applicazione viene aperta automaticamente.

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<application android:icon="@drawable/icon">
        <activity android:name=".Activity1" android:label="@string/app_name">
                <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
        </activity>
        <receiver android:name=".MyActivitySMS" >
                <intent-filter>
                        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
                </intent-filter>
        </receiver>
</application>
```

Figura 5 - Esempio di Broadcast Receiver

1.3.4 – Content Provider

Un'applicazione potrebbe dover gestire dati esterni o aver bisogno di esporne ad altre applicazioni; a questo scopo potrebbe utilizzare i **Content Provider**.

I Content Provider servono a rendere disponibili specifici set di dati tra diverse applicazioni e implementano un insieme di metodi classici per accedere a un archivio dati condiviso da un'applicazione diversa, o da un suo componente. È preferibile il loro utilizzo alla manipolazione di dati grezzi perché tramite i Content Provider otteniamo un disaccoppiamento del codice dai dati, che è preferibile perché previene malfunzionamenti nel caso il formato dei dati cambi. Le applicazioni possono accedere ai Content Provider tramite i rispettivi URI per depositare o prelevare informazioni di interesse usando l'oggetto **ContentResolver**. Questo oggetto dispone di metodi che chiamano procedure, del Provider interessato, aventi lo stesso nome (stessa firma). Di solito, l'accesso ai Content Provider è regolato da permessi. Il termine URI (*Uniform Recource Identifier*) è un acronimo del più generico URL (*Uniform Resource Locator*), riferito principalmente a Internet, che indica una stringa che identifica univocamente una risorsa generica che può essere un indirizzo Web, un documento, un'immagine, un file, un servizio, un indirizzo di posta elettronica, ecc.

Si può utilizzare qualsiasi meccanismo di archiviazione dati presente su piattaforma Android: file, database SQLite, XML, connessioni virtuali.

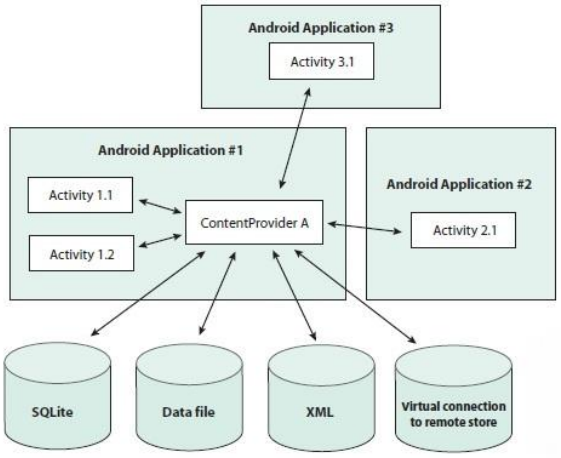

Figura 6 - Schema di un Content Provider

Un Content Provider espone i propri dati ad applicazioni esterne in una forma tabellare, molto simile a quella di un database relazionale, dove in ogni riga è rappresentata un'istanza di un certo tipo di dato che il Provider raccoglie. Uno dei provider incorporati nella piattaforma Android è il dizionario utente, che memorizza l'ortografia delle parole non standard che l'utente desidera mantenere. Nell'esempio seguente esaminiamo questo tipo di Provider, che casualmente ha una colonna soprannominata *_ID*, ma essa non è essenziale ai fini della memorizzazione dei record, salvo che non si voglia associare il proprio Provider a una *ListView* (che è un componente grafico che utilizza il Content Provider per avere un riferimento ai dati da esporre).

word	**app id**	**frequency**	**locale**	**_ID**
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

I Content Provider sono descritti dalle sottoclassi della classe astratta **android.content.ContentProvider** e sovrascrivono alcuni metodi utili a svolgere azioni tipiche riguardanti le banche dati, come creare, recuperare, cancellare e aggiornare campi di una tabella. Forniamo di seguito dei brevi chiarimenti sui più comuni metodi da ridefinire:

- **onCreate()** è chiamato per inizializzare il Provider ed eseguire particolari azioni iniziali. Questo metodo è chiamato al momento del lancio dell'applicazione, non deve quindi eseguire operazioni lunghe, o l'avvio

dell'applicazione potrebbe essere rallentato. Ci occuperemo delle operazioni più complesse in un altro punto dell'applicazione per evitare che problemi nell'avvio del Content Provider (ad esempio di disco rovinato) arrestino l'avvio dell'intera applicazione. Se stiamo utilizzando un database SQLite, *SQLiteOpenHelper* si dimostra una classe molto utile per la sua gestione per esempio rimanda automaticamente l'apertura fino a quando non è espressamente richiesta.

- **query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)** è il metodo più articolato, implementabile per gestire le richieste di dati da parte dei clienti; l'*uri* è il nome da interrogare; le *projection* sono le colonne da inserire nel cursore, se uguale a null saranno incluse tutte le colonne della tabella; *selection* (simile clausola WHERE del linguaggio SQL) rappresenta un criterio di selezione da applicare durante il filtraggio delle righe, se null tutte le righe saranno incluse; le stringhe in *selectionArgs* rappresentano le clausole del parametro precedente; *sortOrder* (SORT BY) è la clausola di ordinamento per le righe del cursore risultante.
- **insert (Uri uri, ContentValues values)** serve ad inserire nuovi dati nel Content Provider; gli si forniscono l'*uri* del richiedente e una serie di *values*, coppia di elementi nome-colonna e valore da aggiungere al database e ritorna l'URI per l'elemento appena inserito. La classe *ContentValue* utilizzata serve per formattare dati processabili da un ContentResolver.
- **update (Uri uri, ContentValues values, String selection, String[] selectionArgs)** da implementare per gestire le richieste di aggiornamento (simile all'UPDATE nel linguaggio SQL) per una o più righe; bisogna fornirgli l'*uri* da interrogare che potrebbe contenere l'identificativo di una riga se l'aggiornamento è da effettuare solo su quella specifica riga; una serie di *values* contenenti vecchi e nuovi nomi di colonna, che può essere anche null; una stringa *selection* che rappresenta un filtro opzionale per abbinare le righe da aggiornare; applicando delle clausole di selezione, tramite *selectionArgs*, è possibile incidere su più righe. Il metodo ritorna il numero di righe effettivamente aggiornate.

- **delete (Uri uri, String selection, String[] selectionArgs)** gestisce le richieste di eliminazione di una o più righe (se usiamo clausole di selezione in *selectionArgs*); bisogna fornirgli l'*uri* da interrogare includendo un identificativo di riga se dobbiamo eliminarne una specifica; una *selection* rappresentante una restrizione opzionale che è possibile applicare alle righe durante l'eliminazione. Il metodo restituisce il numero di righe interessate nell'operazione e può sollevare una *SQLException*.
- **getType (Uri uri)** restituisce il tipo MIME dei dati, presso l'*uri* specificato, che dovrebbe iniziare con *vnd.android.cursor.item*, nel caso di un singolo elemento, o con *vnd.android.cursor.dir/,* per elementi multipli.

  È buona norma per i metodi *insert* e *update* chiamare la funzione **notifyChange (Uri uri, ContentObserver observer)** dopo che hanno svolto i loro compiti. Questa funzione comunica al cliente *observer* registrato (tramite il metodo *registerContentObserver()*) che ci sono stati degli aggiornamenti. Similmente possiamo invocare la **notifyDelete()** dopo aver cancellato delle righe.

```
public class SimpleContentProvider extends ContentProvider{
        @Override
        public int delete(Uri uri, String selection, String[] selectionArgs){
                System.out.println("delete(Uri, String, String[]) called");
                return 0;
        }
        @Override
        public String getType(Uri uri){
                System.out.println("getType(Uri) called");
                return null;
        }
        @Override
        public Uri insert(Uri uri, ContentValues values){
                System.out.println("insert(Uri, ContentValues) called");
                return null;
        }
        @Override
        public boolean onCreate(){
                System.out.println("onCreate() called");
                return false;
        }
        @Override
        public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder){
                System.out.println("query(Uri, String[], String, String[], String) called");
                return null;
        }
        @Override
        public int update(Uri uri, ContentValues values, String selection,
                        String[] selectionArgs){
                System.out.println("update(Uri, ContentValues, String, String[]) called");
                return 0;
        }
}
```

Figura 7 - Esempio di codice del Content Provider

1.3.5 – Manifest

Android apprende da quali componenti è composta l'applicazione tramite il file **Android Manifest.xml**. Riportiamo qui di seguito un esempio di file Manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.project" android:versionCode="1"
        android:versionName="1.0">
        <application android:label="@string/app_name" android:icon="@drawable/icon">
                <activity android:name=".MyActivity" android:label="@string/app_name">
                        <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category Android:name="android.intent.category.LAUNCHER" />
                        </intent-filter>
                </activity>
        </application>
</manifest>
```

Figura 8 - Un esempio di file Manifest che dichiara un'Acitivity

Il file inizia necessariamente con il prologo **<?xml version="1.0" encoding="utf-8"?>** che lo identifica come file XML versione 1.0 il cui contenuto è codificato con lo standard UTF-8. Il successivo tag **<manifest>** rappresenta l'elemento primario (root) di questo tipo di documento XML. Esso contiene degli attributi che specificano alcune caratteristiche del progetto: *android* identifica il namespace Android, *package* specifica il package Java dell'applicazione e *versionCode/versionName* informazioni sulla Versione. Al tag <manifest>, segue quello **<application>** che rappresenta il diretto genitore delle componenti dell'applicazione. I suoi attributi *label* e *icon* si riferiscono a delle risorse dalle quali Android attinge per visualizzare rispettivamente il titolo e l'icona dell'applicazione. Innestata nel tag <application> troviamo quello **<activity>** che descrive appunto un'attività specificandone il nome tramite l'attributo *name* e altre caratteristiche. All'interno del precedente tag troviamo invece **<intent-filter>** che dichiara alcune funzionalità del tag contenitore attraverso due suoi tag figli: **<action>** che identifica l'azione da compiere (all'attributo *android:name* è assegnato il valore "*android.intent.action.MAIN*" quando si vuole indicare che l'attività che si sta descrivendo è quella principale dell'applicazione); **<category>** invece identifica il tipo di componente (se stiamo dichiarando l'attività principale che deve essere avviata al lancio dell'applicazione, l'attributo *andorid:name* deve avere il valore "*android.intent.category.LAUNCHER*").

Il Manifest potrebbe contenere anche dei tag **<uses-permission>** che descrivono i permessi di cui l'applicazione ha bisogno. Essi appaiono come figli diretti del tag <manifest>, allo stesso livello del tag <application>. Per esempio, per un'applicazione che ha bisogno di usare la videocamera, bisogna specificare il suddetto tag con il seguente attributo *"<uses-permission android:name="android.permission.CAMERA"/>*". I permessi sono comunque regolati da autorizzazioni presenti nella firma digitale o da interazioni dirette con l'utente durante l'installazione (non ci saranno ulteriori controlli durante l'esecuzione dell'applicazione).

Vediamo ora come dichiarare altri importati elementi di un'applicazione nel file AndroidManifest.xml.

Usiamo il tag **<service>** per dichiarare un Servizio nel Manifest; esso è figlio del tag <application> e può contenere i tag <intent-filter> e <meta-data>.

```
<service android:enabled=["true" | "false"]
    android:exported=["true" | "false"]
    android:icon="drawable resource"
```

Figura 9 - La dichiarazione di un servizio

Usiamo il tag **<receiver>** per dichiarare un Broadcast Receiver; esso è figlio del tag <application> e può contenere i tag <intent-filter> e <meta-data>.

```
<receiver android:enabled=["true" | "false"]
    android:exported=["true" | "false"]
    android:icon="drawable resource"
    android:label="string resource"
    android:name="string"
    ... >
  . . .
</receiver>
```

Figura 10 - La dichiarazione di un receiver

Usiamo il tag **<provider>** per dichiarare un Content Provider; esso è figlio del tag <application> e può contenere vari tag tra cui <meta-data> ed altri

```
<provider android:authorities="list"
    android:enabled=["true" | "false"]
    android:exported=["true" | "false"]
    android:grantUriPermissions=["true" | "false"]
    android:icon="drawable resource"
    android:initOrder="integer"
    android:label="string resource"
    android:multiprocess=["true" | "false"]
    android:name="string"
    android:permission="string"
    android:process="string"
    android:readPermission="string"
    android:syncable=["true" | "false"]
    android:writePermission="string" >
  . . .
</provider>
```

Figura 11 - La dichiarazione di un Provider

1.3.6 – Application Package e firma digitale

L'insieme dei file contenenti il codice Java, i dati necessari e le risorse esterne sono raccolti in un pacchetto applicazione con estensione **.APK**. I pacchetti APK, anche se nel linguaggio comune vengono spesso usati come sinonimi, non sono da considerarsi vere e proprie applicazioni poiché potrebbero utilizzare componenti di altri pacchetti APK. Essi, tuttavia, sono usati per distribuire e installare le applicazioni sui dispositivi mobili.

Ogni APK deve essere *firmato* con un certificato che ne identifica l'autore. Il certificato, di proprietà dell'autore stesso (non occorre la firma di un'autorità di certificazione), è usato da Android, oltre che per reperire l'identità dell'autore, anche per stabilire determinate relazioni tra applicazioni con la stessa firma; ma il sistema operativo non discrimina in base a questo quali applicazioni possano essere o no installate sul dispositivo.

Un utente, potendo creare autonomamente una propria firma, deve comunque rispettarne la prassi. Una chiave deve rappresentare l'entità personale, aziendale o organizzativa che si vuole sia legata all'applicazione. Google raccomanda di impostare un periodo di validità superiore ai 25 anni, ma se si intende pubblicare l'applicazione sul *Google Play*, lo store online ufficiale di Google precedentemente soprannominato *Google Market*, bisogna comunque tenere presente che il periodo di validità della chiave deve superare il 22 Ottobre 2033. Inoltre non sono accettate chiavi prodotte dall'Android SDK durante le operazioni di test. Tutti i contenuti originali distribuiti in forma digitale, saranno automaticamente protetti da copyright in tutti i Paesi aderenti alla Convenzione di Berna. Le applicazioni possono essere depositate in maniera che siano gratuitamente scaricabili da parte degli utenti, in questo caso è comune trovarne con banner pubblicitari all'interno frutto di una negoziazione di sponsorizzazione con terzi, o a pagamento, o come versione gratuita e limitata di un'applicazione più ampia acquistabile.

1.4 – Processo di sviluppo per applicazioni Android

1.4.1 – Android Software Development Kit ed Android Virtual Device

Google mette a disposizione degli sviluppatori diverse versioni dell'**Android SDK 4.x** per vari sistemi operativi: Windows, Linux, Mac OS X. Osservando il contenuto del pacchetto software, possiamo notare che Google fornisce molteplici strumenti di supporto come add-on, driver USB, esempi e documentazione offline e anche SDK di terze parti (ad esempio gli SDK

proprietari di Sony e Samsung). Ovviamente, essendo le applicazioni Android scritte in linguaggio Java, la macchina su cui installeremo l'Android SDK dovrà poter compilare il codice, quindi possedere il kit di sviluppo Java JDK 5 o JDK 6 (non basterebbe disporre esclusivamente dell'ambiente esecutivo JRE, Java Runtime Environment). Le diverse versioni del kit di sviluppo Android sono disponibili all'indirizzo *http://developer.android.com/sdk/index.html* (James & Nelson, 2010). Dopo averne eseguito l'installazione, possiamo aggiungere il direttorio, dove abbiamo estratto gli strumenti di sviluppo, alla variabile d'ambiente *PATH* in modo da poterne usufruire da riga di comando, da qualunque posizione nel filesystem ci trovassimo.

Passiamo ora in esame le principali sotto-cartelle dell'archivio generale che abbiamo scaricato per esaminarne le caratteristiche:

- **Add-on**: contiene i classici componenti aggiuntivi come ad esempio le API aggiuntive di Google.
- **Platforms**: questa cartella, inizialmente vuota, andrà a contenerne altre per tutte le rispettive versioni della piattaforma (ad esempio ci saranno due cartelle separate per la versione 10 delle librerie, Gingerbread, e la più versione un po' più recente 15, Ice-Cream Sandwich e l'ultima release 16, Jelly Bean).
- **Platform-Tools**: cartella che compare con l'installazione di particolari piattaforme e che contiene degli strumenti aggiornabili con ciascuna piattaforma.
- **Tools**: contiene una serie di strumenti di sviluppo e di analisi, indipendenti dalla piattaforma usata per quanto riguardo il loro aggiornamento o utilizzo stesso.
- **SDK Manager e AVD Manager**: sono rispettivamente due strumenti utili a lanciare l'SDK, con la possibilità di decidere successivamente quali pacchetti scaricare e installare; o a gestire gli AVD (Android Virtual Device). Tramite questi agenti è possibile anche configurare alcune impostazioni aggiuntive come il Server Proxy.

Dopo aver installato l'Android SDK con i relativi pacchetti e piattaforme, siamo abilitati a creare applicazioni Android; le quali, tuttavia, non potranno essere eseguite fino a quando non avremo generato anche un **Android Virtual Device** (**AVD**). Questo interessante simulatore riproduce, e mette a disposizione dell'ambiente di sviluppo, tutte le caratteristiche di un dispositivo reale, ma eseguito sulla nostra macchina da lavoro. Durante la creazione di questo strumento, bisogna tuttavia assicurarsi che la piattaforma di destinazione abbia un livello API uguale, o superiore, al livello di API richiesto dalle applicazioni che ci appresteremo a testare, o semplicemente eseguire, tramite questo specifico AVD. Ognuno di questi dispositivi virtuali rappresenta un terminale indipendente con propria connessione, memoria per i dati utente, schede di memoria e quant'altro. Android supporta 16 diversi dispositivi in esecuzione contemporanea (per i quali è fortemente consigliato il supporto della virtualizzazione hardware). Ai vari dispositivi sarà assegnato un numero di porta pari, per permetterne la connessione e l'accesso a Internet, partendo dalla 5554. Il successivo numero di porta verrà invece sempre assegnato al relativo *Android Debug Bridge* (*ADB*).

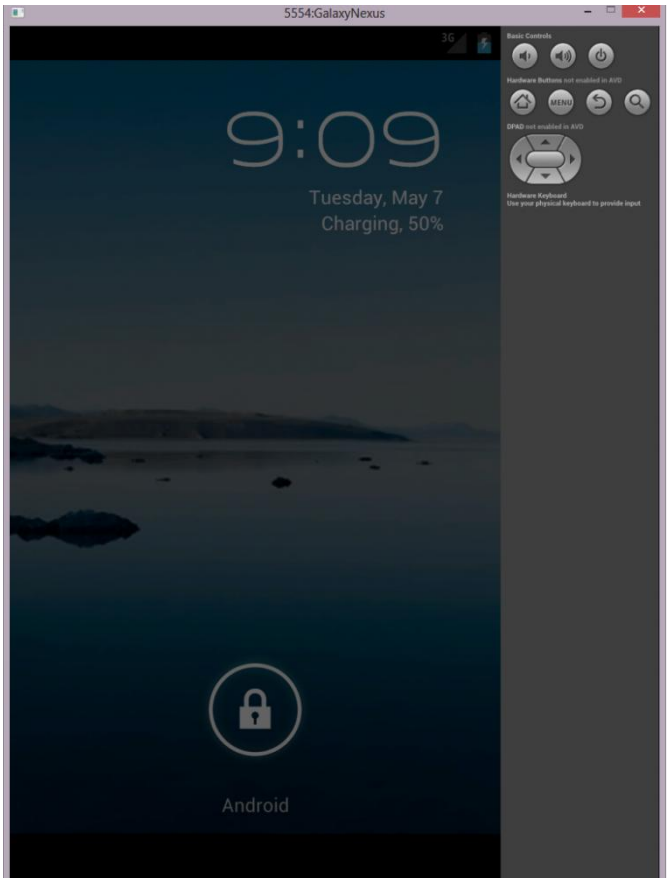

Figura 12 - Il simulatore appena avviato

1.4.2 – Android Debug Bridge

La gestone degli emulatori, o eventualmente di dispositivi reali collegati alla macchina di sviluppo per eseguire test, può essere fata da un tool chiamato **Android Debug Bridge** (**adb**); un versatile strumento a riga di comando (shell, con molti comandi simili a quelli usati in ambienti Unix) che si mette in comunicazione con uno specifico dispositivo. È un programma Client/Server fondamentalmente composto da tre diversi agenti: un Cliente in esecuzione sul computer di sviluppo; un Server, in esecuzione in background sulla stessa macchina, che garantisce la comunicazione tra il Client e il *Demone adb* in esecuzione sull'emulatore in background anch'esso. Lo strumento è reperibile all'interno della cartella *<sdk>/platform-tools/*.

All'avvio di un Client adb, esso controlla in primo luogo se vi è un processo Server adb, in esecuzione, per avviarlo qualora non ci fosse. All'avvio, il Server invece si collega alla porta locale TCP *5037* mettendosi in ascolto dei

comandi inviati dai Clienti adb i quali utilizzeranno tutti la porta di default 5037 per comunicare con il Server. Inoltre esso provvede al settaggio di tutte le opzioni riguardanti le connessioni con ogni emulatore/dispositivo collegato scansionando le porte assegnate dal sistema operativo (5555-5585). Se il Server trova un demone adb, imposta una connessione verso quella porta. Ricordando la regola sull'acquisizione dei due numeri di porta consecutivi da parte degli emulatori facciamo un breve esempio di come operano:

Emulator 1, console: 5554

Emulator 1, adb: 5555

Emulator 2, console: 5556

Emulator 2, adb: 5557

Device 1, console 5558

Device 1, adb 5559 …

Una volta superato il passaggio di connessione, possiamo inviare messaggi di controllo alla shell per controllare le diverse istanze da qualsiasi Client o tramite script. La riga di comando di invocazione è la seguente: **adb [-d|-e|-s <serialNumber>] <command>**; il parametro *–s* serve se siamo in presenza di più istanze di emulatori: *adb -s emulator-5556 install helloWorld.apk*. Possiamo interrogare lo strumento, a proposito del numero di emulatori/dispositivi collegati, in questo modo: *adb devices*, che genererà un output simile:

$ adb devices

List of *devices* attached

emulator-5554 device

emulator-5556 device

emulator-5558 device…

È possibile eseguire tutte le operazioni necessarie da riga di commando, ma vediamo in seguito come operare con un più comodo ambiente integrato.

1.4.3 – Eclipse

Qualora fossimo abituati a lavorare in un ambiente di sviluppo integrato (Integrated Development Environment), possiamo avvalerci del valido plug-in disponibile in Eclipse. Per prima cosa bisogna controllare che la versione di Eclipse che ci apprestiamo a installare sia compatibile tanto con l'Android SDK, quanto con il plug-in che integra l'SDK con il nostro IDE, l'**Android Development Tools** (**ADT**).

Sebbene sia possibile sviluppare applicazioni Android in Eclipse senza l'uso del plug-in ADT, il suo utilizzo agevola il processo di sviluppo, rendendolo più veloce e più facile da testare. Questo particolare componente aggiuntivo ci consente di creare velocemente nuovi progetti Android compilando automaticamente una versione standard dei file di base di cui abbiamo bisogno; inoltre ci collega con utili strumenti di sviluppo, come, ad esempio, il DDMS (*Dalvik Debug Monitor Server*), che consente di gestire il nostro dispositivo di debug direttamente dall'IDE Eclipse, abilitandoci ad esplorarne il filesystem, scattare screenshot (foto della schermata) e trasferire file. Grazie a questa configurazione, disponiamo anche di un editor di file XML, utile nella composizione del manifest file, dei layout e di altre importanti risorse del nostro progetto. Sarà anche possibile, alla fine del processo di sviluppo, esportare automaticamente la nostra nuova applicazione in un file installabile .APK (Application Package) firmato, distribuibile agli utenti.

Il plug-in è reperibile da internet tramite la procedura di installazione di nuovo software all'interno di Eclipse, indicando come repository quella di Google https://dl-ssl.google.com/android/eclipse/. Nelle ultime versioni dell'SDK è inclusa anche una versione di Eclipse Juno preconfigurata con il plugin adt pre-installato.

Capitolo 2 – E-Health

2.1- Introduzione all'e-Health

2.1.1 – Il termine

E-Health è un termine coniato di recente, utilizzato per indicare la gestione della propria salute attraverso strumenti puramente informatizzati. Il termine sta ad indicare tutte le pratiche inerenti la comunicazione medico-paziente, l'utilizzo di strumenti informatici per monitorare la situazione del paziente da personale specializzato; quindi possiamo dire che E-Health sta ad indicare il complesso di risorse, soluzioni e tecnologie informatiche applicate alla salute ed alla sanità. Le possibile sfaccettature di questo termine spaziano dalla medicina/healthcare all'informazione tecnologica:

- **Cartella clinica elettronica:** permette una comunicazione molto più efficace delle condizioni di salute del paziente tra le varie figure professionali (medico di base, medico specialista, farmacista, care-team) permettendo inoltre di visualizzare eventuali patologie pregresse ed esami svolti.

Figura 13 - Un esempio di E-Health

- **Telemedicina:**è un aspetto delle cure mediche che non richiedono una visita del paziente dal proprio medico curante; questo servizio

consente ai medici di allargare il proprio bacino d'utenza ed ai pazienti di evitare visite non necessarie dal proprio medico.

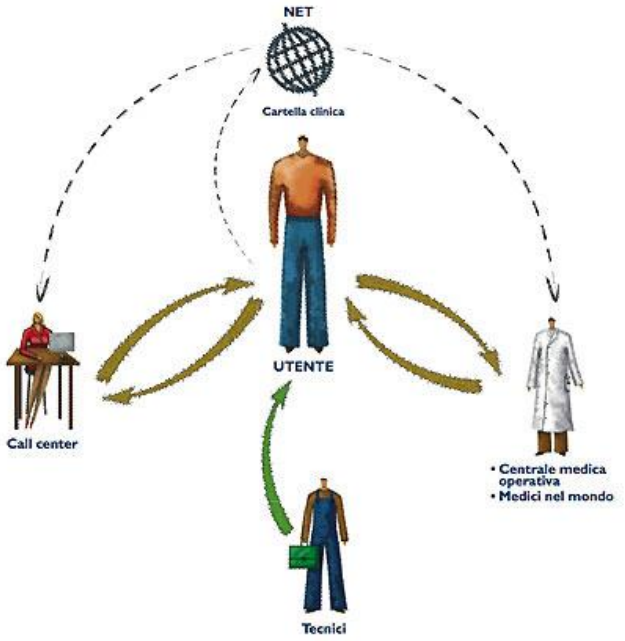

Figura 14 - Il modello di E-Health incentrato sul paziente

- **Medicina basata su prove di efficacia:**il paziente viene collegato a degli strumenti che consentono al medico di verificare, tramite la diagnosi ed i risultati messi in evidenza dalla strumentazione, le proprie ricerche scientifiche e nel contempo poter aggiornare i dati dei pazienti.
- **Consumer HealthInformatics:**è quella branca dell'informazione medica che si occupa di analizzare i bisogni e le abitudini dei consumatori; in base a questi dati indirizza le ricerche e gli studi nel campo medico.
- **Virtual Healthcare Team:** è un team di professionisti che collaborano e condividono informazioni e dati dei pazienti tramite apparecchiature digitali.

Il termine E-Health è stato utilizzato originariamente in ambito industriale prima che in campo accademico, è stato creato in concomitanza con tante altre e-words (ad esempio e-commerce, e-business) per cercare di attirare l'hype dell'e-commerce sul settore sanitario e per espandere l'utilizzo di internet

anche all'healthcare. Intel per esempio ha definito l'E-Health come "uno sforzo comune da parte dei leaders dell'industria sanitaria e tecnologica intrapreso per sfruttare appieno i vantaggi derivati dalla convergenza di internet e dell'healthcare". Le sfide di questo nuovo settore tecnologico sono la possibilità di interagire tra il consumatore e il sistema sanitario online, la capacità di interagire e comunicare tra le varie istituzioni con la definizione di nuovi standard e la possibilità di interazioni tra consumatori in modalità "peer-to-peer".

2.1.2 – Le dieci regole

La E di E-Health non sta solo a indicare "electronics" ma bensì è anche un insieme di 10 regole (le 10 "e" dell'E-Health) che dovrebbero caratterizzare questo nuovo "standard":

1. **Efficienza:**una delle promesse dell'E-Health è aumentare l'efficienza e diminuire i costi dell'healthcare;
2. **Migliorare la qualità (Enancingquality):**migliorare la qualità delle cure, che implica non solo una riduzione dei costi, per esempio indirizzando il paziente verso terapie che hanno avuto risultati migliori per quanto riguarda la sua situazione di salute;
3. **Basati sull'evidenza (Evidencebased):**tramite l'E-Health i vari interventi devono essere eseguiti sulle basi degli esami svolti e su basi efficienti dettate da una rigorosa valutazione scientifica;
4. **Responsabilizzazione (Empowerment):**responsabilizzazione dei consumatori e dei pazienti, facendo si che le basi di conoscenza della medicina e le informazioni personali, siano accessibili ai consumatori tramite Internet; E-Health introduce una medicina incentrata sul paziente e gli consente di effettuare scelte basate sull'evidenza;
5. **Incoraggiamento (Encouragment):**incoraggiamento di una nuova "relazione" tra il paziente e una figura professionale (medico di base, medico specialista, medico curante) dove le decisioni sono

prese in maniera condivisa e basate su una solida documentazione scientifica;

6. **Educazione:**educazione sia dei medici (aggiornamenti per quanto riguarda le nuove scoperte e le nuove terapie) sia dei pazienti (educazione alla salute, basata su informazione preventiva);
7. **Scambio di informazioni (Exchange information):**abilitando lo scambio di informazioni in maniera standardizzata tra i vari enti che si occupano della sanità;
8. **Estensione:**estensione del concetto di health careoltre i confini tradizionali. Questa estensione deve essere intesa sia in senso geografico che in senso concettuale, rendendo disponibile, tramite E-Health, in maniera rapida, una qualsiasi prestazione sanitaria (sia essa estremamente semplice o molto complessa) da qualsiasi ente sanitario nel mondo;
9. **Etica:**l'estensione delle pratiche di health care alla rete porta una nuova via di interazione tra medico e paziente, questa nuova via di interagire deve però tener conto di consensi informati, diritto alla privacy ed alla parità(vedi punto 10).
10. **Parità (Equity):**rendere l'assistenza sanitaria più equa è una delle promesse di E-Health, ma allo stesso tempo vi è una notevole possibilità che la sanità elettronica può accentuare il divario tra i "ricchi" e i "non abbienti". Le persone, che non hanno i mezzi, le competenze e/o l'accesso a computer e Internet, non hanno la possibilità di sfruttare le potenzialità di E-Health. Di conseguenza, queste popolazioni di pazienti (che in realtà trarrebbero i maggiori benefici da informazioni) sono quelle che hanno meno probabilità di trarre vantaggio dai progressi nella tecnologia dell'informazione, a meno che la politica non introduca misure atte a garantire l'accesso alle risorse informatiche a tutti. Il digital divide attualmente le popolazioni rurali rispetto a quelle urbane, ricchi contro poveri, giovani contro vecchi, e tra le malattie quelle trascurate/rare rispetto a quelle più comuni.

In aggiunta a queste 10 regole possiamo aggiungerne alcune che dovrebbero essere comuni ad ogni standard di gestione: la facilità d'uso e l'intrattenimento.

2.1.3 –I modelli principali delle applicazioni nel campo dell'E-Health

Il punto di vista prettamente informatico riguardo E-Health ci fa pensare a un insieme di metodi, modelli e principi di design per l'implementazione e la valutazione di applicazioni prototipali e lo sviluppo di servizi **ICT** (***I**nformation and **C**ommunication**T**echnology*) basati sulla gestione dei dati, informazione e conoscenza dell'healthcare. Questo lavoro di sviluppo, come abbiamo già visto, va espanso a un concetto globale collaborando con gli istituti di ricerca nazionali e internazionali focalizzando lo sviluppo su due temi principali:

- **Patient-centered-eHealth:** lo sviluppo di applicazioni e servizi basati sul modello di healthcare incentrato sui pazienti e cittadini in generale;
- **Clinical-eHealth:** sviluppo di applicazioni e sistemi per gli operatori dell'healthcare (medici, infermieri e tutte le figure che ruotano attorno alla sanità) a supporto della qualità delle cure.

Il PCEH (PatientcenteredeHealth) è una forma di E-Health dove il paziente è considerato il fulcro centrale delle varie applicazioni. Questa branca dell'E-Health inoltre si occupa di analizzare i dati dei pazienti riguardo la necessità e la modalità di reperimento delle informazioni, lo studio di questi dati porta ad una modellazione delle informazioni molto più efficiente ed aiuta a personalizzare con le preferenze dei pazienti le applicazioni. Il PCEH infatti è nato in un contesto sociale incentrato alla personalizzazione delle cure e al potenziamento della capacità di gestione della propria cura da parte del cittadino. Il dominio del PCEH ha infatti portato un nuovo modello di dati: il PHR (***P**ersonal **H**ealth**R**ecord*); questo modello non è ancora definito in una

forma standard ed è oggetto di molte ricerche e conferenze per definirlo in modo da centrare uno dei grandi obiettivi dell'E-Health.

Il CleH (*ClinicaleHealth*) è un modello di comunicazione e assistenza a livello infrastrutturale, infatti consente alle varie entità di collaborare, conoscere e di conseguenza saper orientare le cure di un paziente. Un esempio lampante è dato dalla cura del cancro: il cancro è una patologia complessa che coinvolge più specialisti (spesso provenienti da tutto il mondo) per un periodo di cura molto lungo; la sua cura va decisa e monitorata costantemente e ogni singolo dato va valutato da un pool di esperti in base alle ultime ricerche mediche. Il CleH tramite una pratica denominata CPG (*ClinicalPraticalGuideline*)si occupa proprio di delineare un sistema di *decision-making* che indirizza il paziente verso le cure più adatte ed efficaci in base alla situazione in cui si trova: in particolare cerca di aumentare la qualità delle cure tenendo bassi i costi evitando gran parte delle pratiche mediche superflue.
Il *Computer DecisionSupport System* è una implementazione pratica del CPGs, definito in un linguaggio comprensibile dalla macchina; è in grado di promuovere e portare avanti efficientemente la cura, soprattutto se distribuito attraverso i sistemi informativi e di informazione clinica che risultano automatizzati da questa infrastruttura.
Inoltre c'è un progressivo bisogno di tool computer-based per misurare la qualità delle terapie sostenute da molte organizzazioni di healthcare, che costituirebbe uno strumento fondamentale dello sviluppo per la qualità delle terapie.

I modelli applicativi suddetti richiedono molte competenze sia a livello medico (più teoriche) che di software engineering (aspetto molto più pratico):

- Conoscenza di usi e rappresentazioni di ontologie mediche e semantica di base per sistemi knowledge-based;
- Standard per la rappresentazione e lo scambio di informazioni cliniche;

- Conoscenza delle architetture software web-based, distribuite assicurando lo sviluppo di software affidabile e in linea con i requisiti di privacy e sicurezza;
- Capacità di analizzare le richieste degli utenti e conoscenza dei test di usabilità;
- Disegnare interfacce grafiche incentrate sull'utente, facendo particolare attenzione alle interfacce dirette alla comunità medica ed ai cittadini.

Queste regole sono le basi per la progettazione di un sistema software (distribuito) che permette lo sviluppo di applicazioni compatibili con almeno uno dei due modelli applicativi suddetti.

2.2 – Da E-Health a M-Health

Come abbiamo visto la definizione di E-Health racchiude molti settori della medicina e altrettanti settori dell'ingegneria dell'informazione; lo scopo di questa tesi punta ad analizzare ed ampliare un settore che è a tutt'oggi in fase di sviluppo: le tecnologie sviluppate su dispositivi mobili o anche dette M-Health.

M-Health (*Mobile Health*) vuole ampliare il concetto di E-Health consentendoci di portare con noi i nostri principali dati medici (quali la cartella clinica, la terapia a cui siamo sottoposti, i medici che ci hanno curato…) aumentando così, in maniera del tutto sicura, l'efficacia e la tempestività delle terapie e delle cure nel breve e medio periodo.

2.2.1 – I requisiti

I requisiti fondamentali un'efficace integrazione delle tecnologie in ambito mobile sono estremamente rigidi e precisi, ma soprattutto devono essere considerati come veri e propri standard per garantire l'interoperabilità fra i vari enti:

1. I dati dei pazienti devono essere tutti disponibili in formato digitale;
2. La semantica dei dati deve essere standardizzata per evitare di perdere tempo a dover fare delle "traduzioni" per interpretare i dati;
3. I metodi e gli strumenti devono essere standardizzati in quanto risultano più efficaci per l'autonomia dei medici/personale specializzato utilizzare strumenti quantomeno simili (al limite poche differenze nella UI);
4. Reti estremamente veloci ed affidabili in modo da consentire comunicazioni in "real time" ed avere di conseguenza cure più efficaci;

Questi sono i nodi chiave per favorire l'integrazione delle tecnologie mobili in ambito E-Health; ovviamente dovremmo andare a "sommare" anche i requisiti per quanto riguarda la parte informatica dell'E-Health vero e proprio.

2.2.2 – I problemi

2.2.2.1 – La gestione delle informazioni

Il sovraccarico di informazioni è un problema universale di cui tutti soffriamo. Gli operatori sanitari non sono esclusi da questa difficile situazione. Nell'attuale contesto clinico non sono in grado di possedere tutte le conoscenze necessarie per fornire cure mediche in un' "ambiente" sicuro. Pochi medici possono permettersi il lusso di tenersi aggiornati con la grande quantità di informazioni cliniche che viene pubblicato. La loro dipendenza da "terzi" a vagliare e applicare decisioni cliniche, di informazioni è una tendenza che aumenterà in relazione alla crescita delle informazioni cliniche, sia essa pubblicata e diffuso dalla stampa tradizionale o in formato elettronico. La velocità e la facilità di accesso alle linee guida cliniche basate sull'evidenza è fondamentale per il successo dell'integrazione di conoscenze cliniche nuove ed aggiornate nella pratica clinica quotidiana. L'impossibilità di accedere alla conoscenza clinica può comportare disparità nelle prestazioni sanitarie e inficiare il risultato delle cure.

2.2.2.2 – I soggetti interessati

Prendiamo in esempio un caso estremamente significativo: il Regno Unito. Ivi la proprietà delle cartelle cliniche elettroniche migrerà da un server centrale custodito ed amministrato dalle organizzazioni sanitarie, alla "proprietà" dei singoli pazienti, con livelli ben definiti di accesso. Come conseguenza di questo cambiamento di proprietà, il rapporto tra il medico ed il paziente cambierà inevitabilmente. I pazienti si aspettano di essere parte del processo decisionale per quanto riguarda le cure. La figura del 'paziente esperto', si viene a delineare come persona "istruita" in quanto ha lo stesso accesso, come il medico, alle informazioni mediche sulla rete e può rappresentare una "sfida" per quanto riguarda le capacità decisionali.

2.2.2.3 –Atteggiamento e percezione a livello clinico

Il Regno Unito, come già detto precedentemente, è uno dei pionieri di M-Health; infatti è pratica molto diffusa lo scambio, tramite tecnologie di comunicazione sempre più mobili, di informazioni cliniche. Il successo di queste forme di comunicazioni, oltre che del "note keeping" elettronico, deve essere regolato a livello governativo e supportato da infrastrutture nazionali, quali il NationalSalute Information Service Authority (NHSIA per quanto riguarda il Regno Unito). Sempre tenendo in considerazione il caso di studio del Regno Unito, notiamo che la diffusione di cure primarie e secondarie coordinate a livello elettronico rischia di scemare a causa di un non completo abbandono del cartaceo; alcuni medici infatti continuando ad utilizzare ancora le note cartacee sebbene siano disponibili versioni elettroniche. L'uso dei dispositivi mobili come dispositivi di comunicazione all'interno del contesto clinico può diventare un modo molto efficace per sostenere l'integrazione di note cliniche elettroniche mantenendo un efficiente scambio di informazioni all'interno delle organizzazioni sanitarie e sostenere un interfacciamento molto valido con le organizzazioni correlate al mondo sanitario.

2.2.2.4 – I Dispositivi

Gli smartphone ed i PDA sono i nodi principali per far si che M-Health si diffonda in ambito clinico. Il loro uso come cellulare "clinico", archivio di dati,

è un aggiunta alla diffusa funzione di "diario elettronico". Tuttavia questi presentano anche dei problemi non banali. Un dispositivo portatile quale lo smartphone è più soggetto a furti, danneggiamenti dovuti a cadute o altro, rispetto a un pc. Questa "fragilità" può rappresentare un problema nell'uso quotidiano, in quanto si tradurrebbe in un aumento dei costi per l'organizzazione sanitaria. I professionisti della sanità hanno espresso delle preoccupazioni riguardanti gli schermi di piccole dimensioni che potrebbero limitare la quantità di informazioni visualizzate in una quantità di tempo. L'utilizzo dei dispositivi mobili può essere limitato dalla piccola quantità di dati memorizzabili sul dispositivo, anche se il crescente aumento delle memorie porrà sicuramente fine a questo problema.Un ostacolo più grande, invece, è l'autonomia; infatti con l'aumentare delle prestazioni dei processori in ambito mobile c'è il rischio che l'affidabilità nel tempo del dispositivo stesso venga compromesso. La comunicazione dei dati medici tra diversi dispositivi non deve essere strettamente legato alla piattaforma, ma deve prevedere un modello dei dati standard. L'NHSIA, sempre prendendo in esempio il modello inglese, sta cercando di basarsi su un modello dei dati fortemente legato a un linguaggio di markup estremamente potente quali XML (*Extensible **M**arkup **L**anguage*). Indipendentemente dal dispositivo scelto, l'uso di XML facilita la gestione ed il workflow di un documento elettronico consente la distribuzione dei compiti tra il personale sanitario multi-specializzato che comporta l'ottimizzazione delle cure del paziente.

2.2.2.5 – Gli Standard

Sicurezza e privacy dei dati clinici sono in prima linea in ogni dibattito riguardante l'uso di dispositivi mobili in ambito clinico. È essenziale che si uniformino gli standard per le informazioni sul paziente e lo scambio elettronico di dati che verranno adottate a livello mondiale. Recentemente, il governo degli Stati Uniti ha spinto affinché Health Level Seven (HL7) sia riconosciuto come standard per quanto riguarda il formato di base dei dati.

2.2.3 – Linee guida per una integrazione di successo

2.2.3.1 –Informazioni presso le cliniche

Soluzioni sanitarie mobili consentono un punto di interazione per le cure indipendentemente dal fatto che il paziente è ricoverato in ospedale, a casa o si trovi in qualsiasi altro posto. I dispositivi mobili sono in grado di assistere gli operatori sanitari nella loro quotidianità in ambiente clinico, facilitando lo scambio di informazioni in maniera tempestiva. I dispositivi mobili possono anche essere utili nella ricerca medica, utilizzati sia per la sperimentazione clinica che per registrare e trasmettere i dati che con i metodi tradizionali avrebbero dei costi non accettabili. E' essenziale che le informazioni cliniche vengano consegnate in tempo reale e possano essere facilmente accessibili e comprensibili da tutti i soggetti interessati, compresi i pazienti, che sono coinvolti nella terapia.

2.2.3.2 – Training

L'implementazione efficace delle tecnologie per l'informazione richiede un impegno da parte degli stessi operatori sanitari, nonché delle organizzazioni sanitarie. Un'infrastruttura e dei corsi di formazione sono essenziali, non solo per mostrare ed insegnare l'uso corretto di queste tecnologie, ma anche per dimostrare i benefici che si possono ottenere. L'integrazione di dispositivi mobili nella pratica clinica quotidiana deve essere integrata da programmi di sviluppo professionale continuo e una costante pratica didattica e clinica. Anche se la formazione è costosa e richiede tempo, è un investimento che è fondamentale per lo sviluppo e il mantenimento della sanità elettronica.

2.2.3.3 – Valutazione

L'integrazione di dispositivi mobili per catturare e tenere sotto controllo i dati clinici e di ricerca e sperimentazioni deve essere sottoposto ad una valutazione completa e rigorosa in ogni fase del suo sviluppo, dall'appalto alla sua installazione vera e propria nella clinica. Un elemento chiave della valutazione è la necessità di individuare le "best practice" e garantire che queste vengano implementate in maniera tale da risultare di facile comprensione alla più ampia comunità clinica. A meno che si possa dimostrare che i dispositivi mobili

hanno migliorato l'efficacia organizzativa, e hanno contribuito al miglioramento delle cure cliniche, in generale, il loro contributo alla gestione della conoscenza clinica sarà valutato in termini di costi-benefici anche se realmente non è così.

2.2.3.4 – Gli esiti clinici

La raccolta e la conservazione dei dati clinici per via elettronica presso la clinica offre ai professionisti sanitari l'accesso immediato ai dati clinici, come lo storico dei farmaci somministrati al paziente, i risultati degli esami effettuati in altre strutture o in altri reparti. L'accesso mobile consente inoltre ai professionisti sanitari di accedere e aggiornare i piani di assistenza, le linee guida per il paziente e la terapia da seguire.

2.2.4 – I possibili scenari per un sistema M-Health (un particolare caso di studio riguardante le malattie cardiovascolari)

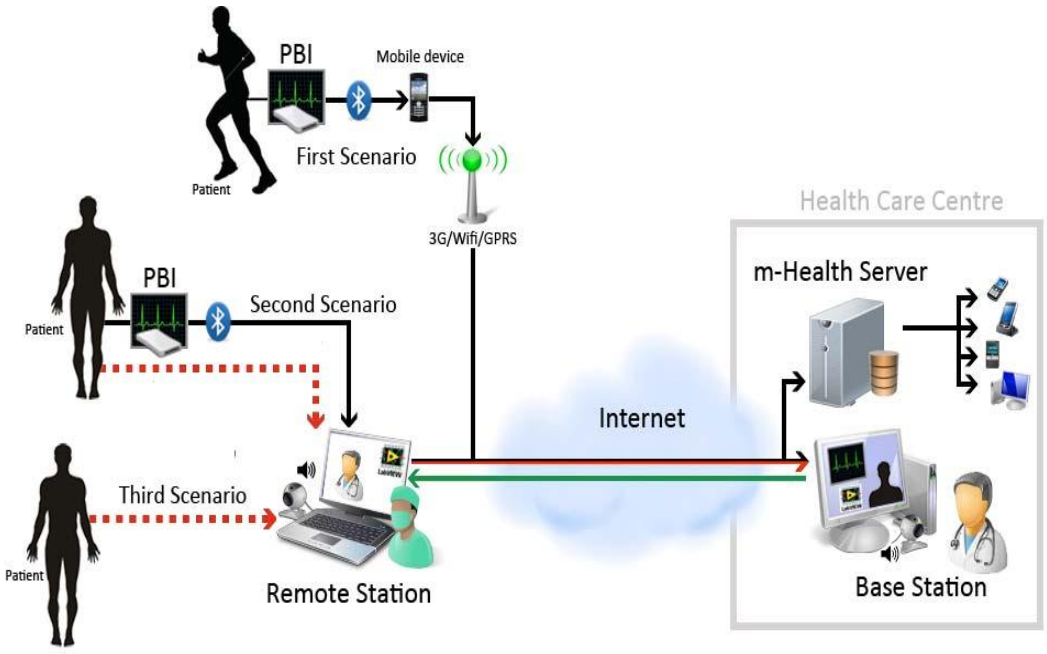

Figura 15 - I tre possibili scenari per un sistema M-Health

La figura 1 mostra lo schema completo della proposta per un sistema di telemedicina che soddisfi le esigenze di e-Health e servizi di M-Health. In

sostanza lo schema è diviso in tre scenari: 1) PBI (*Portable Biomedical Instrument*), 2) Stazione remota e 3) Healthcare Center. Il PBI rileva e trasmette il segnale ECG. La stazione remota permette il teleconsulto (linee tratteggiate in figura 1) comprendente le informazioni del PBI. Healthcare Center include la stazione base per il telerilevamento in tempo reale e un server che mantiene i record del paziente e fornisce servizi web per l'analisi dei dati in maniera asincrona.

I servizi medici sono evoluti e in un prossimo futuro potranno fornire un'ampia gamma, come ad esempio: monitoraggio in tempo reale, monitoraggio dei dati con paziente in movimento e asincrono e così via. Il sistema M-Healthin esempio intende coprire tali esigenze in base agli scenari che vengono mostrati in Figura 1.

Il primo scenario rappresenta un paziente che ha collegato un PBI e potrebbe dover essere continuamente monitorato, mentre sta facendo le sue attività quotidiane. Ad intervalli regolari, precedentemente programmati dal personale medico, il PBI automaticamente invierà i dati registrati al server m-Health utilizzando un dispositivo mobile con funzionalità di accesso alla rete (LTE,3G, 2G), senza alcun intervento umano. Per il secondo scenario si ritiene che il paziente si rechi presso la clinica che ha il PBI e una stazione remota (quali anche un semplice laptop), necessaria per il teleconsulto. Il PBI collegato al paziente verrà impostato come modalità real-time (il PBI non registra, invia solo alla stazione remota). Tutto l'hardware sarà gestito dal personale medico. La GUI del medico indicherà i dati del paziente provenienti dal PBI ed i dati audio-visivi del teleconsulto. Dopo che il teleconsulto è finito e il collegamento con la stazione remota è chiusa i dati del paziente provenienti dal PBI saranno inviati automaticamente al server M-Health. Il terzo scenario è costituito dal sistema di teleconsulto senza utilizzare il PBI. Infine, nel server M-Health, le cartelle cliniche elettroniche (EHR), saranno disponibili per ulteriori analisi tramite il sito web, a cui il medico può accedere attraverso un qualsiasi dispositivo con connessione a Internet (sia esso smartphone, pc o altro). In questa fase del progetto un ECG (*Elettrocardiogramma*) Holter portatile è usato come PBI.

2.2.4.1 – Il PBI

Il sistema M-Health riportato consente l'integrazione di diversi PBI che sono in grado di fornire diverse informazioni per quanto riguarda il battito cardiaco, che ha una grande importanza per i medici; inoltre questi rilevamenti possono aiutare a diagnosticare malattie molto complesse. Per essere comodo, portatile e facile da usare per il paziente, il PBI deve essere a basso consumo energetico e di dimensioni ridotte. Nel nostro caso ricordiamo che il PBI è stato implementato come un ECG Holter dato che il caso di studio che vogliamo prendere in considerazione riguarda un possibile rilevamento di malattie cardiovascolari, che è anche uno scenario piuttosto attuale dato l'elevato tasso di mortalità causato da quest'ultime. In questo caso è molto importante mantenere i pazienti sotto stretto controllo, in alcuni casi continuo, e anche di avere un programma di prevenzione che permette al settore sanitario di fornire un servizio migliore, con un uso efficiente delle risorse e delle comunicazioni. Pertanto, questo caso di studio considera come prioritaria l'attuazione di un sistema di monitoraggio cardiovascolare. Il sistema ECG si compone di tre derivazioni sulla base di condizionamento del segnale, da analogico a digitale (A/D), la conversione e l'elaborazione del segnale. In scenario un dispositivo BlackBerryviene utilizzato come interfaccia per inviare dati dal dell'ECG Holter al data server facendo uso della rete EDGE/3G, come una soluzione per il primo scenario. Per il secondo scenario, l'ECG Holter è impostato sulla modalità in tempo reale e invia i dati al computer portatile che si occupa di inviare i dati al server M-Health e al computer del medico presso L'Healthcare Center.

Questo PBI include anche gli algoritmi che permettono la diagnosi automatica e in tempo reale, al momento comprende la misurazione del QT e la rilevazione di eventuali aritmie che aiutano a prevenire ulteriori sintomi di insufficienza cardiaca tra cui la morte improvvisa. I dati ECG e i risultati dell'analisi sono mostrati nella Figura 2. Dove il grafico inferiore mostra il segnale in tempo reale, la tavola superiore destra visualizza le misurazioni medi calcolati sotto il segnale ECG per 3 secondi e la tabella in alto a sinistra mostra la

frequenzacardiaca e la lunghezza del valore QT come fattore per la diagnosi precoce di aritmie.

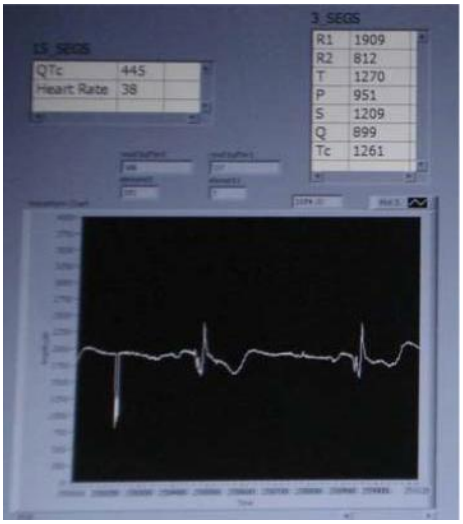

Figura 16 - L'ECG e le sue analisi generate nel PBI

2.2.4.2 – Il sistema di tele-consultazione

Il sistema di tele-consultazione ha lo scopo di fornire dati per i servizi medici in tempo reale, come mostrato nel secondo e terzo scenario in Figura 1. Questo sistema è composto di due moduli principali, uno che risiede nel Central Healthcare e l'altro nella posizione remota. Entrambi i siti, il Central Healthcare e la stazione remota, si basano sul collocamento di un computer portatile per integrare l'applicazione teleconsulto. Il computer ha integrato tutti i dispositivi edi driver necessari per gestire il teleconsulto come: webcam, altoparlanti, microfono, e porte USB 2.0, Bluetooth, WiFi, Ethernet e EDGE/3G attraverso un BlackBerry. La figura 3 mostra uno schema a blocchi per il trasferimento dei dati in maniera sicura dal PBI utilizzando la rete 3G.

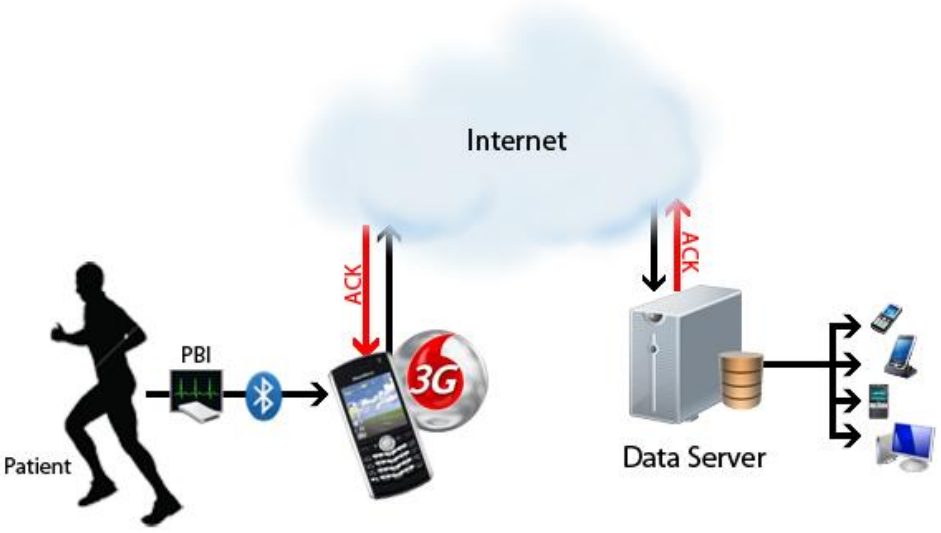

Figura 17 - Il trasferimento di dati biomedici ad un server remoto

Come accennato in precedenza presso la clinica a distanza uno staff medico è necessario per gestire tutto l'hardware, sarà anche incaricato di stabilire la comunicazione con la stazione remota, al fine di effettuare il teleconsulto. Il personale medico assiste il medico che è responsabile del collegamento del PBI per il paziente nel caso in cui siano richiesti quel tipo di dati. Il Central Healthcare dall'altro lato è controllato da un medico specializzato.

Su entrambi i lati del sistema di teleconsulto, l'applicazione si basa su un linguaggio di programmazione visiva, come mostrato nella Figura 4. L'applicazione comprende i campi necessari per i parametri di connessione e login (per far si che avvenga una connessione sicura da un'identità quantomeno verificabile), uno schermo che contiene i dati. La richiesta del medico comprende anche gli indicatori e controlli per la manipolazione dei dati PBI. La figura 4 mostra quattro aree per teleconsulto, i primi grafici a sinistra i segnali audio in alto a destra del paziente, in basso a destra l'analisi del segnale e in basso a sinistra il segnale ECG in tempo reale.

Un teleconsulto richiede un appuntamento al fine di impostare tutte le configurazioni di connessione necessarie. Entrambe le parti devono passare un filtro di sicurezza basato su un form in modo da garantire che un determinato medico corrisponde a un determinato paziente e anche per evitare che tale persona non autorizzata possa gestire il sistema edi dati disponibili nel server.

Figura 18 - Un esempio dell'interfaccia grafica

2.2.4.3 – Il server M-Healthed i Web Services

Il progetto preso in considerazione prevede che il server che contiene l'applicazione Web è lo stesso utilizzato come server dati. Il server presenta i servizi necessari per la partecipazione a visite e l'amministrazione remota dei clienti. Le caratteristiche sono: httpd server Apache 2.0 per HTTP e requisiti del protocollo HTTPS, SSH per l'amministrazione remota, server Tomcat è il contenitore di applicazioni Java, MySQL come DBMS e NTP per garantire la sincronizzazione dell'ora del server con il Central Healthcare.L'obiettivo del server è non solo memorizzare i record del sito Web (informazioni di login, news, ecc), ma anche di fornire il database EHR per la memorizzazione dei dati generatidalla PBI nella posizione remota per un'ulteriore analisi.

Lo sviluppo di un sito web per future analisi dei dati del paziente, come detto all'inizio, è importante perché permetterà al medico di accedere sempre e ovunque tramite qualsiasi dispositivo (cellulare, smartphone, computer portatile, ecc) con accesso ad Internet a questi dati. Il sito può essere separato in due parti, l'infrastruttura di comunicazione del server e l'interfaccia utente. Il Model View Controller (MVC) come pattern architetturale viene utilizzato

perché isola la logica di business (accesso ai dati e la logica di business) dall'interfaccia utente (codice di presentazione).
Uno degli aspetti fondamentali di utilizzo di linguaggi lato server è quello di consentire la creazione di siti dinamici sostenuti principalmente dalle banche dati. Per assicurare una integrazione ottimale del sito in qualsiasi server indipendentemente dal DBMS che viene eseguito, viene utilizzato Hibernate, che è incaricato di stabilire la comunicazione e attualizzazione della sintassi SQL tra i drivers.
Per il lato client è previsto lo sviluppo di una Rich Internet Application (RIA), rendendo facile la navigazione del cliente e con la possibilità di presentare i grafici relativi ai dati del PBI, con la possibilità di manipolarli per un'analisi più approfondita del medico.In termini generali l'applicazione sarà in grado di adattarsi a qualsiasi schermo del dispositivo.

2.2.5 – Il Fascicolo Sanitario Elettronico

Il Fascicolo Sanitario Elettronico (FSE) è un servizio sviluppato dalla regione Emilia Romagna, che mira a salvare tutti i dati sanitari e la storia clinica di una persona; l'accesso al servizio è regolamentato e subordinato da un login con credenziali personali.
Le funzionalità del FSE per il momento sono limitate,dato che attualmente è ancora fase di sviluppo; per il momento la storia clinica di un utente è visualizzabile solo dall'utente stesso, in un futuro prossimo questi dati saranno consultabili anche dal personale medico specializzato in maniera sicura e rispettosa della privacy.
Il progetto FSE è stato possibile grazie al supporto della rete SOLE (*Sanita On Line*), che è una rete che collega i medici di base, i pediatri ed i medici di famiglia alle strutture sanitarie ed ospedaliere.

2.2.5.1 – I documenti contenuti

Il FSE segue l'evolversi della rete SOLE e recupera da quest'ultima tutti i documenti presenti per dal 1 gennaio 2008, il sistema inoltre permette agli utenti di inserire anche altri documenti sanitari in loro possesso recanti data precedente.

Il sistema rende disponibili una varietà di dati relativi a prestazioni mediche e tutte le anagrafiche utili:

- Referti di visite ed esami erogati dalle strutture pubbliche del Servizio sanitario regionale dell'Emilia-Romagna (Aziende Usl, Aziende Ospedaliere, Aziende Ospedaliero-Universitarie, Istituti di ricovero e cura a carattere scientifico);
- Referti di pronto soccorso;
- Lettere di dimissioni da ricovero in ospedali pubblici;
- Prescrizioni Specialistiche;
- Prescrizioni Farmaceutiche;
- Documenti amministrativi;

Una caratteristica anche molto interessante è la possibilità di caricare la propria cartella clinica in formato PDF (la cartella è possibile ottenerla da un servizio chiamato *Prontocartella*). Essendo un servizio ancora in fase sperimentale, è previsto che il paziente debba comunque avere una copia cartacea dei documenti presenti nel sistema.

2.2.5.2 – La tutela della privacy

Il progetto FSE ha ricevuto l'approvazione del Garante per la protezione dei dati personali, in merito alla protezione e alla tutela dei dati degli utenti; ricordiamo inoltre che è alimentato dai file del programma SOLE che è disciplinata sotto il profilo degli adempimenti descritti dalla normativa sulla privacy e dal documento intitolato: *Linee Guida in materia di comunicazione di dati personali e sanitari nell'ambito del Progetto Sole Rete integrata Ospedale – Territorio nelle Aziende sanitarie della Regione Emilia-Romagna. Modalità di prestazione ed acquisizione del relativo consenso*.

2.2.5.3 –I servizi disponibili Online

Il sistema mette a disposizione vari servizi aggiuntivi, oltre il caricamento e l'accesso alla propria cartella clinica; questi servizi sono disponibili anche (come vedremo in seguito) nell'applicazione per smartphone, i servizi a disposizione sono:

- Pagamenti online di prestazioni sanitarie;

- Prenotazioni SSN: aiuta a gestire le proprie prenotazioni per quanto riguarda il Servizio Sanitario Nazionale;
- Prenotazioni Libera Professione: aiuta a gestire le prenotazioni in regime di libera professione;
- Cambio o revoca del medico di famiglia;
- Gestione del consenso SOLE: permette di gestire l'invio dei dati al sistema SOLE;

2.2.5.3.1 – I Pagamenti

Il sistema permette di poter visualizzare ed inviare pagamenti per ticket di prestazioni sanitarie, rende inoltre possibile scaricare in PDF le ricevute dei pagamenti già effettuati;

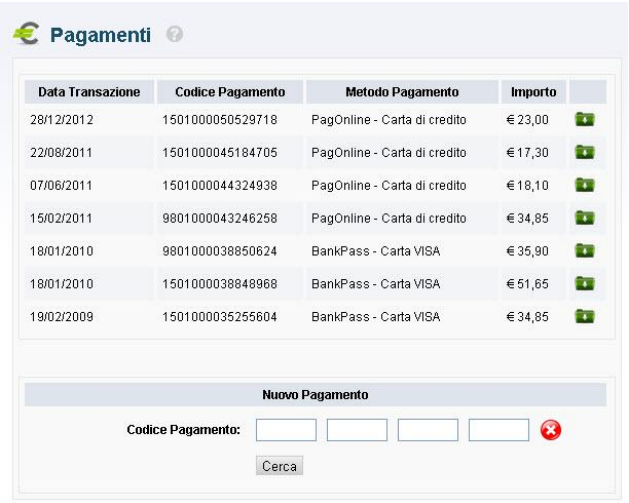

Figura 19 - La schermata dei pagamenti

2.2.5.3.2 – La Prenotazione SSN

FSE oltre a immagazzinare i dati degli esami, permette anche di prenotarne. La prenotazione può essere effettuata per visite ed esami specialistici per le aziende sanitarie aderenti al progetto, inoltre rende possibile anche modificare (ove possibile) gli appuntamenti presi con SSN, CUP Web, gli sportelli del CUP (*Centro Unico Prenotazioni*) o in farmacia; consente inoltre di stampare tutto il materiale cartaceo degli appuntamenti attivi prenotati su tutte le Aziende della Regione Emilia Romagna.

Le prestazioni prenotabili online sono le più standard, le visite specialistiche più complesse non sono prenotabili in quanto necessitano dell'intervento di un operatore. Una prenotazione deve rispettare due caratteristiche:

- La data non deve essere antecedente a sei mesi;
- Tutte le prestazioni in essa prescritte devono essere prenotabili online (altrimenti la procedura onn andrà a buon fine);

Le prenotazioni verranno visualizzate in base alla specialità medica e sono visibili anche le prenotazioni con operatore. Il sistema rende anche possibile (ai fini della prenotazione) la ricerca delle strutture che erogano una particolare prestazione sanitaria, limitatamente al territorio regionale.

Figura 20 - il riepilogo delle prenotazioni online

2.2.5.3.3 – Prenotazione prestazione da Libero Professionista

La prenotazione per i servizi erogati da Libero Professionista presentano le medesime caratteristiche delle prenotazioni effettuate in SSN più altre caratteristiche utili, in quanto non è possibile pagare online una prestazione ad un libero professionista:

- Stampare il prospetto del modulo da presentare per il pagamento della prestazione al libero professionista;
- Stampa del promemoria per il riepilogo dell'appuntamento preso;
- Modifica dell'appuntamento;

- Possibilità di disdire l'appuntamento;
- Dettagli dell'appuntamento (ora e luogo, indirizzo completo del luogo, nome del medico);

Figura 21 - Il riepilogo di un appuntamento

2.2.5.3.4 – La gestione del consenso SOLE

Il consenso SOLE (*Sanità On Line*), consente di regolare l'invio e la trattazione dei propri dati al sistema SOLE; questo consenso è necessario per regolare l'accesso ai propri dati da parte del personale medico specializzato, i livelli di consenso a disposizione sono due:

- **Livello 1:** consenso all'invio dei dati al sistema SOLE e alla loro consultazione da parte del medico di medicina generale o del pediatra di libera scelta, oltre che dei professionisti sanitari che interverranno nel suo percorso assistenziale, in tutta la Regione Emilia-Romagna;
- **Livello 2:** consenso all'invio dei dati al sistema SOLE e alla loro consultazione solo da parte del medico di medicina generale o del pediatra di libera scelta o dell'eventuale specialista che ha effettuato la prescrizione, ma solo per l'esito dell'esame o della visita specialistica richiesta.

Indipendentemente dal livello di consenso scelto rimane comunque possibile non far conoscere i dati relativi a singole prestazioni o episodi di diagnosi e cura; questa possibilità è stata inserite per garantire comunque la riservatezza

dell'utente ed è definita *oscuramento*. Questa funzione di manifestazione e/o modifica è disponibile nelle sole aziende sanitarie di: Parma, Reggio Emilia, Modena, Bologna, Imola, Ferrara, Ravenna, Cesena, Rimini.

2.2.5.4 – Accenni sul progetto SOLE

Il progetto sole fornisce servizi e scambio di informazioni tra i medici e pediatri di famiglia, altri specialisti e operatori sanitari ospedalieri e ambulatoriali, strutture amministrative delle Aziende sanitarie, operatori regionali autorizzati, operatori del progetto Sole.

2.2.5.4.1 – I servizi

La rete sole offre molti servizi a livello di backofficeed a livello burocratico:

- Allineamento anagrafico automatico degli assistiti tra l'Azienda Usl e i medici e pediatri di famiglia con le relative scelte o revoche effettuate.
- Utilizzo di un unico catalogo regionale delle prestazioni (catalogo Sole) per la prescrizione di prestazioni specialistiche ambulatoriali da parte di medici e pediatri di famiglia e medici specialisti.
- Gestione della prescrizione elettronica per visite ed esami specialistici: le prescrizioni effettuate dai medici e pediatri di famiglia e dagli specialisti sono messe a disposizione dei sistemi Cup aziendali i quali, attraverso un codice, "recuperano" in via informatica tutte le informazioni contenute nella ricetta cartacea. Ciò permette di velocizzare i tempi allo sportello e agevolare leprenotazioni telefoniche.
- Ritorno dalle Aziende sanitarie ai medici e pediatri di famiglia dei referti dei propri assistiti per accertamenti di laboratorio, di radiologia e specialistica, oltre che delle notifiche di avvenuto ricovero/dimissione in ospedale, delle relative lettere di dimissione, dei referti sintetici di prestazioni di pronto soccorso , nel rispetto della normativa vigente sul trattamento dei dati personali.
- Gestione integrata delle esenzioni ticket nella cartella clinica dei medici e pediatri di famiglia.

- Realizzazione, per alcune Aziende sanitarie, della cartella di continuità assistenziale per la gestione dei contatti e delle attività del Servizio di continuità assistenziale.
- Gestione del processo amministrativo dell'Assistenza domiciliare integrata (ADI), grazie alla condivisione tra medici, pediatri di famiglia e Azienda Usl del piano di avvio della presa in carico dell'assistito.
- Sperimentazione dello scambio, in tempo reale, tra medici e pediatri di famiglia e specialisti delle informazioni relative alla presa in carico dei pazienti diabetici (assistenza integrata diabete tra il medico di famiglia e il centro diabetologico).
- Gestione delle vaccinazioni effettuate dai servizi vaccinali, dalle pediatrie di comunità, dal medico di famiglia.
- Notifica da parte dei medici e dei pediatri di famiglia all'Azienda Usl delle prestazioni aggiuntive erogate agli assistiti (fleboclisi, medicazioni, ecc.)
- Invio delle prescrizioni farmaceutiche effettuate dai medici e pediatri di famiglia.
- Invio all'Azienda Usl dei bilanci di salute generati dai pediatri di famiglia.
- Gestione del Patientsummary (profilo sanitario sintetico dell'assistito) generato dal medico di famiglia.
- Servizio di help desk specializzato per medici e operatori sanitari.

Tutto questo va ad integrarsi nella parte orientata alla burocrazia con il *Fascicolo Sanitario Elettronico* (di cui abbiamo parlato sopra).

2.2.5.4.2 – Le Tecnologie

La realizzazione dei servizi previsti nell'ambito della rete Sole è stata utilizzata una architettura di cooperazione applicativa coerente con le linee guida CNIPA (*Centro Nazionale per l'Informatica nella Pubblica Amministrazione*), implementando un modello di collaborazione paritetica tra i molteplici attori coinvolti. L'architettura, a livello di contenuti e applicazioni, richiede lo

scambio di messaggi XML il cui formato e la cui grammatica è stato condivisa e standardizzata.

L'architettura della rete Sole è distribuita su tre diversi livelli:

1. Livello Aziendale:Ogni Azienda sanitaria è dotata di una porta applicativa e del relativo hardware, composta dai seguenti moduli:
    a. strato di cooperazione, composto da sottostrati di software, tra loro indipendenti (web server pubblico, sistema di autenticazione, web service con interfaccia pubblica, adattatore multicanale di input/output, sistema di notifica eventi)
    b. strato di integrazione, sul quale avviene l'implementazione dei web service accedendo alle basi dati aziendali o alle funzioni degli applicativi gestionali in uso e sul quale si produce l'output richiesto.
2. Livello Medico e Pediatra di famiglia:
   Medici e pediatri di famiglia sono dotati di un certificato digitale a doppia chiave nonché di una carta nazionale dei servizi. Gli applicativi software da loro utilizzati sono stati integrati per consentire lo scambio dei dati con il livello aziendale.
3. Livello Regionale:
   La Regione dispone di un accesso applicativo del tutto analogo agli accessi aziendali. L'architettura, a livello di contenuti e applicazioni, richiede lo scambio di messaggi strutturati XML il cui formato e la cui grammatica è condivisa e standardizzata. Questo meccanismo è conforme alle indicazioni dei principali organismi di normazione a livello italiano (UNI/U72), europeo (CEN/TC251) e internazionale (ISO/TC215, HL7) per lo scambio di informazioni sanitarie.

2.3 – Android,M-Health e E-Health

Android non può vantare ancora applicazioni strettamente correlate al concetto di M-Health sopra esposto, ma altresì ha una moltitudine di applicazioni sempre correlate ai concetti chiave dell'M-Health solo in settori più di nicchia quali: il fitness e i monitoraggi fatti artigianalmente. Invece troviamo più

applicazioni se “allarghiamo” la ricerca al concetto più esteso quale l’E-Health. La categoria (sempre riferita al Google Play Store) che racchiude questo genere di applicazioni è: Salute e Fitness, il che sta ad indicare come l’E-Health in campo mobile non si sia ancora affermato bene come standard.

2.3.1 – Le applicazioni Android per E-Health.

Una ricerca molto accurata sul Play Store (lo store di applicazioni ufficiale di Google) ha evidenziato una forte presenza di applicazioni improntate sull’E-Health.

La prima su tutte che possiamo prendere in considerazione è: eHealth Mobile, ma anche e-Health Sensor Plattform e molte altre di cui parleremo successivamente.

2.3.1.1 – eHealth Mobile

eHealth Mobile è una applicazione di monitoraggio più a livello amministrativo che a livello terapeutico. Questa infatti consente di trovare i dottori più vicini e tenere traccia dei dottori precedentemente “frequentati” (di cui tiene anche una breve anagrafica, come in figura 5);

Figura 22 - Un esempio di breve anagrafica

Permette inoltre di tenere conto dello stato della propria assicurazione medica e delle scadenze.

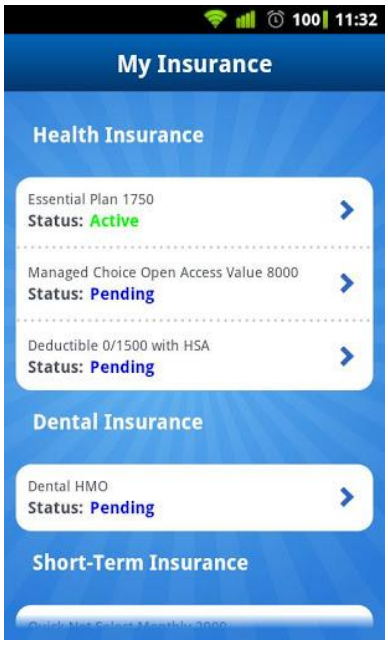

Figura 23 - Le scadenze delle polizze assicurative

Infine è implementato anche un servizio di acquisto delle proprie polizze assicurative in base alla locazione.

Figura 24 - L'acquisto di eventuali polizze

Come possiamo vedere questa è un’applicazione che è pensata prettamente per il mercato americano il cui sistema sanitario prevede piani assicurativi e la

possibilità di frequentare più dottori (in Italia vi è principalmente il medico di famiglia, si ha la possibilità di potersi far visitare da più specialistial limite).

2.3.1.2 – TreC Mobile Apps

Questo insieme di applicazioni provenienti tutte dallo stesso autore (e-Health Unit - FBK), consente al paziente di tenere nota dei propri stati fisici e dei valori rilevati da esami e rilevazioni quotidiane e di sincronizzarle con un middleware proprietario della TreC; le applicazioni spaziano dal monitoraggio del Diabete all'Ipertensione comprendendo anche Asma e Scompenso Cardiaco. Queste applicazioni sono finanziate e fanno parte dell'ecosistema TreC che è un progetto su ampia scala voluto dalla Fondazione Bruno Kessler.
Andando ad esaminare il contenuto delle applicazioni troviamo: il diario e il monitoraggio per l'asma, il diario ed il monitoraggio per l'ipertensione, il monitoraggio per lo scompenso cardiaco, il monitoraggio per il diabete ed il modulo di base per l'interfacciamento con il middleware.

2.3.1.2.1 – Monitoraggio per l'asma

Il monitoraggio per l'asma presenta varie opzioni, tra cui anche la possibilità di inserire eventuali allergie che potrebbero andare ad influire sull'andamento della terapia e sulla salute del paziente.

Figura 25 - La scheda allergologica per l'asma

L’applicazione inoltre permette di tenere traccia delle terapie in corso, dei farmaci in uso e della frequenza di assunzione.

Figura 26 - Le terapie in corso

Oltre che con le varie terapie è anche possibile impostare l’assunzione di farmaci tramite trigger (eventi) ottenuti incrociando anche i dati allergologici.

2.3.1.2.2 – Monitoraggio Scompenso

Lo scompenso rispecchia sempre il modello dell’asma solo prevede una parte più concentrata sulla terapia.

Figura 27 - Il Piano terapeutico

Mentre per quanto riguarda l'assunzione dei farmaci troviamo un'attività molto più banale ma estremamente efficace a livello semantico.

Figura 28 - Assunzione dei farmaci

2.3.1.2.3 – Monitoraggio Ipertensione

Il monitoraggio dell'Ipertensione a livello strutturale è molto simile alle precedenti applicazioni esaminate, ma permette un monitoraggio più capillare delle misurazioni della pressione arteriosa del sangue. Questo monitoraggio però viene effettuato "manualmente" senza interfacciarsi a nessun dispositivo o rilevatore esterno.

Figura 29 - Le misurazioni dell'ipertensione

2.3.1.2.4 – Il modulo di base

Il modulo di base va ad integrare la capacità di rilevazioni e monitoraggi delle applicazioni descritte in precedenza con un middleware proprietario della TreC, il software consente, previa autenticazione con nome utente e password, permette di caricare e sincronizzare i propri dati sulla memoria di massa dello smartphone.

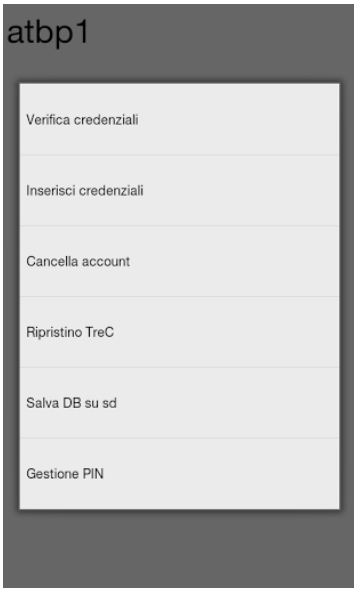

Figura 30 - La sincronizzazione

La Figura 13 evidenzia come sia possibile ripristinare e salvare il proprio DB dal e verso lo smartphone, basandosi principalmente sui dati raccolti con le precedenti applicazioni.

2.3.1.3 – Health Data Managment

Uno dei punti chiave dell'E-Health sta nella diffusione in tempi molto brevi di risultati e scoperte da parte di studi e ricerche effettuate in tutto il mondo, questo avviene tramite pubblicazioni su riviste cartacee e digitali e paper online. L'applicazione in questione si occupa di mettere a disposizioni di un'utenza specializzata tutte le ultime pubblicazioni, tramite riviste in formato digitale e video in streaming, sul mondo dell'E-Health.

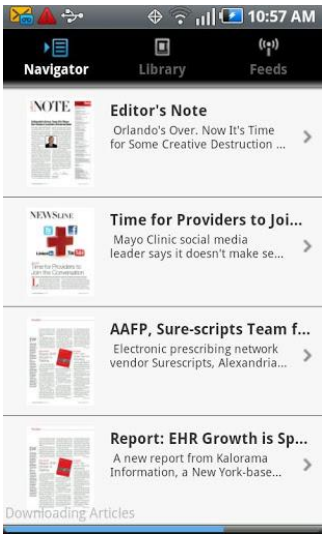

Figura 31 - Le riviste disponibili

Questo software è considerato inoltre uno dei più influenti e completi per quanto riguarda la capillarità delle pubblicazioni, avendo una rete di informazione molto vasta e ben organizzata.

2.3.1.4 – FSE Emilia Romagna

Il progetto FSE (di cui abbiamo precedentemente parlato) ha anche rilasciato un'applicazione per accedere al portale dal proprio smartphone, quest'applicazione consente di accedere al proprio fascicolo elettronico quindi a tutti i propri esami, referti e ricette mediche; questi possono essere consultati direttamente online o scaricati sul proprio dispositivo per una consultazione offline successiva.

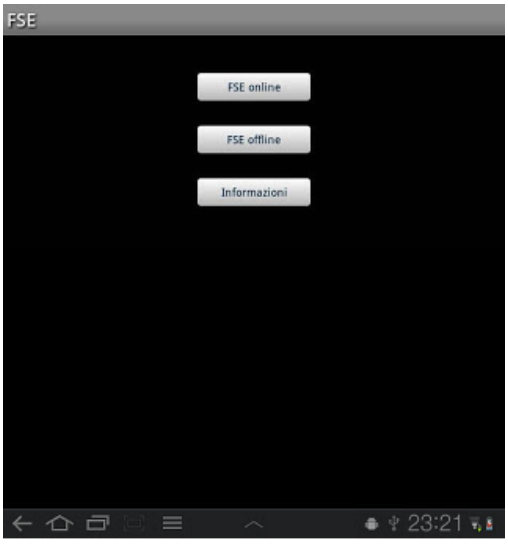

Figura 32 - Scelta della consultazione dei documenti

Figura 33 - La visualizzazione online dei documenti sanitari

Un'altra caratteristica di quest'applicazione, oltre al classico accesso al portale, è la possibilità di tenere un taccuino con le cure e le diete da seguire, inoltre ha anche un'agenda con gli appuntamenti presi per visite e per prenotarne se necessario; il software permette inoltre di pagare anche il ticket per degli esami già svolti.

2.3.2 – Android e M-Health

I software presenti nel Play Store per quanto riguarda invece i concetti di M-Health, sono più "amatoriali" che professionali con alcune eccezioni. Il punto più seguito della filosofia dell'M-Health, per quanto riguarda i software presenti, è il monitoraggio in tempo reale, sia esso a livello di fitness che a livello di ECG (*Elettrocardiogramma*).

2.3.2.1 – E-Health Sensor Platform

Una delle applicazioni per il monitoraggio puro che punta ad utilizzare dispositivi esterni open source quali Arduino e RaspberryPi, quest'applicazione consente l'interfacciamento con il sistema e ne renderizza i risultati in tempo reale.

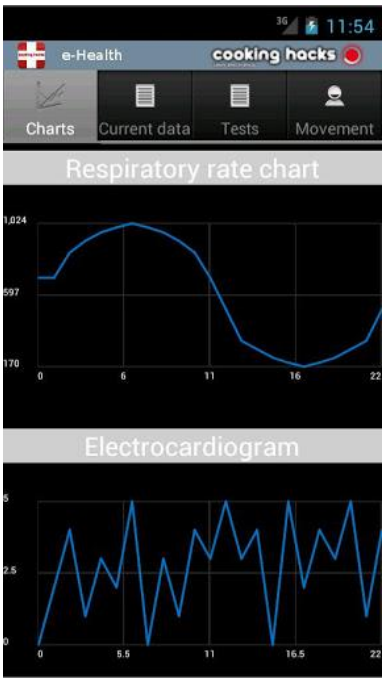

Figura 34 - I risultati della BAN basata su Raspberry e Arduino

Quest'applicazione però è basata molto su dispositivi realizzati su hardware open source e non ha il supporto di alcuna casa farmaceutica o ente.

2.3.2.2 – EPI mHealth Lite

Il software che rispecchia in pieno il caso di studio preso in considerazione in questo capitolo, pone il dispositivo mobile come accesspoint tra l'ECG e il server remoto e inoltre lo stesso smartphone fa anche da remote station per controllare in tempo reale i dati inviati dall'ECG.

Figura 35 - Il dispositivo fa da ponte tra il server e l'ECG

La connessione tra l'ECG e lo smartphone è affidata al Bluetooth, utilizzando un metodo di comunicazione proprietario solo con un ECG proprietario (come in Figura 20). L'applicazione ci consente, oltre a un diretto monitoraggio del nostro ECG (come evidenziato in figura 21), anche di controllare la pressione del sangue il glucosio ed il collesterolo.

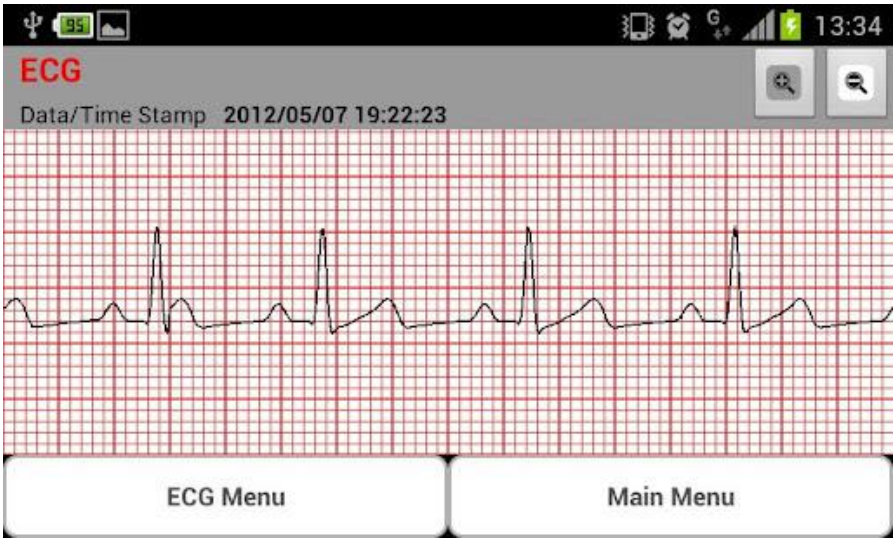

Figura 36 - Il nostro ECG in tempo reale

2.3.2.3 – Cardiografo

Un'altra delle applicazioni di monitoraggio, che al contrario della precedente sfrutta l'hardware del dispositivo e non si appoggia a dispositivi esterni; in particolare richiede l'uso della fotocamera.

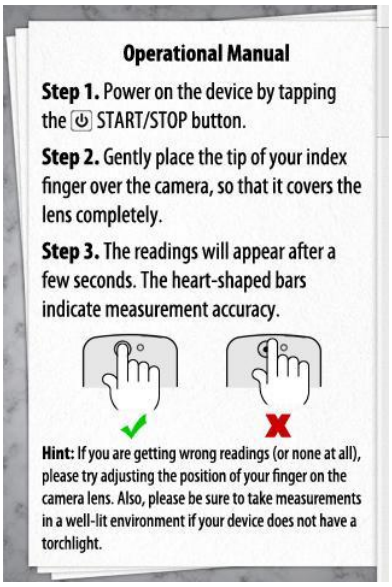

Figura 37 - Le istruzioni per il rilevamento del battito cardiaco

Mentre effettua i rilevamenti l'applicazione renderizza in tempo reale l'andamento dei battiti (come evidenziato in figura 23).

Figura 38 - I grafi in tempo reale

Capitolo 3 – Lo stato dell'arte della crittografia su Android

3.1 – Introduzione alla Crittografia

La crittografia è la branca della crittologia che tratta delle "scritture nascoste", in altre parole dei metodi per rendere un messaggio "offuscato" in modo da non essere comprensibile/intellegibile a persone non autorizzate a leggerlo. Un tale messaggio si chiama comunemente crittogramma e le tecniche usate tecniche di cifratura.

In tal modo si garantisce la confidenzialità dei dati impedendo così la realizzazione di diversi tipi di attacchi e proteggendo di fatto i dati sensibili. L'approccio inverso di studio volto a rompere un meccanismo crittografico è detto invece crittoanalisi che rappresenta l'altra branca della crittologia.

La necessità della crittografia è, al giorno d'oggi, evidente: una parte della comunicazione che avviene tra enti e individui deve essere riservata; si pensi per esempio a comunicazioni bancarie o militari. Inoltre la crittografia serve a proteggere il copyright di testi, software, o immagini per i quali si desidera mantenere il diritto d'autore.

3.1.1 – Tipi di Crittografia

La Crittografia si divide in due grandi tipi: Crittografia Simmetrica e Crittografia Asimmetrica. Queste due tipologie trovano usi in diversi scenari, non sono intercambiabili e hanno richieste computazionali, meccanismi e algoritmi di cifratura completamente diversi.

3.1.1.1 – La Crittografia Asimmetrica

La crittografia asimmetrica, conosciuta anche come crittografia a coppia di chiavi, crittografia a chiave pubblica/privata o anche solo crittografia a chiave pubblica è un tipo di crittografia dove, come si evince dal nome, ad ogni attore coinvolto nella comunicazione è associata una coppia di chiavi:

- La chiave pubblica, che deve essere distribuita, serve a cifrare un documento destinato alla persona che possiede la relativa chiave privata.

- La chiave privata, personale e segreta, utilizzata per decifrare un documento cifrato con la chiave pubblica;

evitando così qualunque problema connesso alla necessità di scambio in maniera sicura dell'unica chiave utile alla cifratura/decifratura presente invece nella crittografia simmetrica.

3.1.1.1.1 – Funzionamento

L'idea base della crittografia con coppia di chiavi diviene più chiara se si usa un'analogia di carattere postale, in cui il mittente è Giovanna ed il destinatario Marco, i lucchetti fanno le veci delle chiavi pubbliche e le chiavi recitano la parte delle chiavi private:

1. Giovanna chiede a Marco di spedirle il suo lucchetto, già aperto. La chiave dello stesso verrà però gelosamente conservata da Marco.
2. Giovanna riceve il lucchetto e, con esso, chiude il pacco e lo spedisce a Marco;
3. Marco riceve il pacco e può aprirlo solo con la chiave di cui è l'unico proprietario;

Se adesso Bob volesse mandare un altro pacco ad Alice, dovrebbe farlo chiudendolo con il lucchetto di Alice (che lei dovrebbe aver preventivamente dato a Bob) che solo lei potrebbe aprire.

Per utilizzare questo tipo di crittografia è necessario creare una coppia di chiavi, una chiave pubblica (da diffondere) ed una chiave privata (da tenere segreta). La proprietà fondamentale della coppia di chiavi pubblica/privata è che un messaggio cifrato usando una delle due chiavi può essere decifrato soltanto usando l'altra chiave. In pratica, la pubblicazione di una delle due chiavi (chiave pubblica) consente a chiunque di:

1- Codificare un messaggio in modo che sia leggibile solo al possessore dell'altra chiave (chiave privata),

2- Verificare l'autenticità di un messaggio proveniente dal possessore della chiave privata, con essa codificato e pertanto decodificabile solo tramite la corrispondente chiave pubblica.

La coppia di chiavi pubblica/privata viene generata attraverso un algoritmo (ad esempio RSA o DSA) a partire da dei numeri casuali. Gli algoritmi asimmetrici sono studiati in modo tale che la conoscenza della chiave pubblica e dell'algoritmo stesso non siano sufficienti per risalire alla chiave privata e tale meccanismo è reso possibile grazie all'uso di funzioni univoche. In realtà, in molti casi, l'impossibilità di risalire alla chiave privata non è dimostrata matematicamente, ma risulta essere un procedimento estremamente lungo data la potenza di calcolo disponibile attualmente. Oltre alla cifratura dei dati di una comunicazione la crittografia asimmetrica presenta altri possibili impieghi: firma digitale per verificare l'autenticazione del mittente e l'integrità informativa del messaggio, fornire una condizione di ending e per i programmi che tentano la forzatura delle chiavi; supporto alla fase di handshake ovvero di avvio di una sessione con crittografia simmetrica per negoziare la chiave di sessione, il protocollo e gli altri aspetti della connessione cifrata.

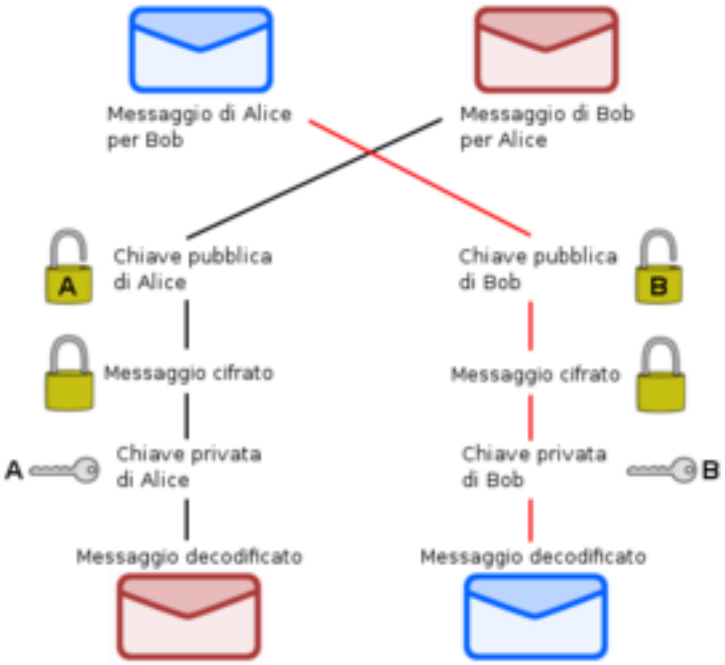

Figura 39 - Lo scambio di chiavi (nell'esempio Alice e Bob)

3.1.1.2 – La Crittografia Simmetrica

Con crittografia simmetrica, o crittografia a chiave privata, si intende una tecnica di cifratura più leggera rispetto a una cifratura asimmetrica. La cifratura simmetrica utilizza una sola chiave condivisa sia per cifrare che decifrare i messaggi.

Generalmente gli algoritmi di crittografia simmetrica sono computazionalmente molto più veloci di quelli a chiave pubblica (o crittografia asimmetrica), per questo vengono usati in tutte le operazioni di cifratura che richiedono determinate performance. Oltretutto i cifrari a crittografia simmetrica permettono l'uso di chiavi lunghe N bit quanto il messaggio, ottenendo uno spazio delle chiavi di dimensioni 2N: ad esempio nella codifica XOR (algoritmicamente semplicissima) è possibile scegliere una chiave lunga quanto il messaggio da cifrare, rendendo il messaggio cifrato assolutamente sicuro. In pratica dunque la crittografia simmetrica è più semplice, veloce di quella asimmetrica, ma necessita che prima venga condivisa una chiave in maniera sicura. Quindi normalmente si usa la crittografia asimmetrica per scambiarsi una chiave in maniera sicura, con cui procedere poi ad una comunicazione a crittografia simmetrica, preservando così i vantaggi di questa. È quanto accade ad esempio nei protocolli di rete crittografici quali SSL/TLS e IPsec.

Figura 40 - Un esempio di funzionamento di crittograifa simmetrica (Chiave 1 = Chiave 2)

3.2 – La Crittografia nei linguaggi di programmazione

Gli algoritmi di crittografia sono implementati ed ottimizzati in quasi tutti i linguaggi di programmazione, le API (*Application Programming Interface*) di sicurezza e crittografia sono raccolte in apposite librerie (o package, in base al linguaggio di programmazione preso in considerazione). I meccanismi invece sono da implementare in base alle esigenze del programmatore, sia per la crittografia simmetrica che per l'asimmetrica.

3.2.1 – Crypto++

Crypto++ è una libreria open source per il linguaggio di programmazione C++, supporta una grande varietà di algoritmi di crittografia sia simmetrica che asimmetrica. Gli algoritmi supportati sono:

Tipo di algoritmo	**Nome**
Sistemi di crittografia autenticati	GCM, CCM, EAX
Cifrari ad alta velocità	Panama, Sosemanuk, Salsa20, XSalsa20
AES e candidati AES	AES (Rijndael), RC6, MARS, Twofish, Serpent, CAST-256
Altri cifrari a blocchi	IDEA, Triple-DES (DES-EDE2 and DES-EDE3), Camellia, SEED, RC5, Blowfish, TEA, XTEA, Skipjack, SHACAL-2
Funzionamento a cifratura a blocchi	ECB, CBC, CBC ciphertext stealing (CTS), CFB, OFB, counter mode (CTR)
Codici di autenticazioni per i messaggi	VMAC, HMAC, GMAC (GCM), CMAC, CBC-MAC, DMAC, Two-Track-MAC
Funzioni di hash	SHA-1, SHA-2 (SHA-224, SHA-256,

	SHA-384, and SHA-512), SHA-3
Crittografia a chiave pubblica (Asimmetrica)	RSA, DSA, ElGamal, Nyberg-Rueppel (NR), Rabin-Williams (RW), LUC, LUCELG, DLIES (variants of DHAES), ESIGN
Schemi di padding per crittografia a chiave pubblica	PKCS#1 v2.0, OAEP, PSS, PSSR, IEEE P1363 EMSA2 and EMSA5
Schemi a key agreement	Diffie-Hellman (DH), Unified Diffie-Hellman (DH2), Menezes-Qu-Vanstone (MQV), LUCDIF, XTR-DH
algoritmi insicuri o obsoleti mantenuti per retrocompatibilità e/o valore storico	MD2, MD4, MD5, Panama Hash, DES, ARC4, SEAL 3.0, WAKE-OFB, DESX (DES-XEX3), RC2, SAFER, 3-WAY, GOST, SHARK, CAST-128, Square

La libreria inoltre presenta anche altre caratteristiche utili per controllare l'integrità dei file (*checksum*) e per la generazione e la verifica dei numeri primi. Questa libreria implementa dei wrapper per le socket di tipo Berkley e Windows.

3.2.2 – Stanford Javascript Crypto Library

La Stanford Javascript Crypto Library è un progetto della Stanford Computer Security Lab con l'obiettivo di costruire una libreria sicura, potente, veloce, piccola, facile da usare e cross-browser per la crittografia in Javascript .
SJCL è facile da usare: è sufficiente eseguire

```
sjcl.encrypt("password", "data")
```

Per criptare i dati, oppure

```
sjcl.decrypt("password", "encrypted-data")
```

Per decriptarli. SJCL è sicuro, utilizza l'algoritmo AES standard industriale a 128, 192 o 256 bit, la funzione hash SHA-256, il codice di autenticazione HMAC; il seed PBKDF2, e le modalità di autenticazione CCM e OCB. Altrettanto importante, i parametri di default sono sensibili: SJCL rafforza le password di un fattore pari a 1000 e applica il salt per proteggerle da attacchi di tipo rainbow table, ed autentica (firma) ogni messaggio che invia a impedire che venga modificato. Possiamo ritenere che SJCL fornisce la migliore sicurezza che è disponibile in Javascript.

SJCL è cross-browser. E' stata testata su tutti i browser tra cui diverse versioni di Internet Explorer, Chrome, Firefox, Safari e Opera su Mac, Linux e Windows e sul motore Rhino, ma ha ancora bisogno di più test.

3.2.3 – Il Package javax.crypto

Il package javax.crypto è un package per il linguaggio Java che fornisce le interfacce e le implementazioni di tutte le classi e gli algoritmi di crittografia. Gli algoritmi di crittografia definiti in questo pacchetto includono la criptazione, la generazione di chiavi e gli algoritmi basati su accordi di chiavi, e la generazione di Message Authentication Code (MAC). Il supporto alla cifratura comprende algoritmi simmetrici, asimmetrici, a blocchi e cifrari a stream. Questo pacchetto supporta anche i stream cifrati e sealed objects. Molte delle classi previste nel pacakge sono basate su dei *Service Provider*, che "configurano" l'oggetto per lo scambio di accordi, generazione di chiavi e quant'altro possa essere creato in modi diversi con parametri personalizzati in base al caso di studio.

Il package presenta anche due tipi di stream per operazioni di input ed output cifrate (rispettivamente *CipherInputStream* e *CipherOutputStream*), che sono estremamente utili per scrivere su file contenuti già criptati secondo il cifrario scelto; questi stream possono essere utilizzati anche per comunicare contenuti già cifrati su canali in chiaro.

Gli algoritmi implementati in questo package sfruttano l'ottimizzazione hardware per avere un impatto computazionale, per quanto riguarda le

prestazioni del software, estremamente ridotto; questa caratteristica, insieme all'intrinseca portabilità del codice Java, ha fatto di questo package un riferimento estremamente valido per la comunità di sviluppatori; proprio per questo Google ha deciso di importare questo package nel suo SDK per quanto riguarda lo sviluppo di applicazioni su Android.

3.2.4 – Bouncy Castle

Il pacchetto Bouncy Castle Crypto è una implementazione in Java di algoritmi crittografici, è stato sviluppato dalla Bouncy Castle Legion, il package è organizzato in modo che contenga API leggere adatte all'uso in qualsiasi ambiente (compreso J2ME) con un'infrastruttura supplementare per conformarsi agli algoritmi del framework JCE.

La Bouncy Castle Crypto API sono così composte:

- Le API di crittografia "leggere", per tiny client;
- Un Provider per JCE (*Java Cryptography Extension*)e uno per JCA(*Java Cryptography Architecture*);
- Una implementazione pulita delle API JCE 1.2.1;
- Una libreria per leggere e scrivere le gli oggetti codificati ASN.1;
- API leggere per l'implementazione di TLS e DTLS;
- Generatori per la Versione 1 e la Versione 3 di X.509, Versione 2 di CRLs e PKCS12;
- Generatore per gli attributi ed i certificati della Versione 2 di X.509;
- Generatori/Processori per S/MIME e CMS;
- Generatori/Processori per OCSP;
- Generatori/Processori per TSP;
- Generatori/Processori per CMP e CRMF;
- Generatori/Processori per Open PGP;
- Generatori/Processori per EAC (*Extended Access Control*);
- Generatori/Processori per DVCS (*Data Validation and Certification Server*);

Figura 41 - La Bouncy Castle Legion

3.3 – La Crittografia nei sistemi mobili

L'evoluzione dei sistemi mobili sta portando ad una maggior complessità strutturale, questo aumento della complessità porta ad un utilizzo dei sistemi mobili quasi al livello dei computer con tutte le conseguenze del caso. Il mobile computing (definizione di elaborazione in mobilità, di cui gli odierni smartphone e tablet ne sono i maggiori interpreti) si sta facendo strada nell'uso quotidiano sia aziendale che personale, il che porta ad immagazzinare grandi quantità di dati su questi dispositivi mobili oltre che a comunicazioni di vario tipo; questi dati e queste comunicazioni però hanno bisogno di "protezione" e riservatezza, in quanto un dispositivo mobile ha molte più probabilità di essere smarrito oppure, nel caso peggiore, rubato. L'unione della maggiore complessità strutturale, vista come evoluzione per quanto riguarda i servizi che vi è possibile implementare, e l'esigenza di protezione dei dati e delle comunicazione ha portato a un vero e proprio porting delle maggiori librerie di crittografia e dei meccanismi a tutt'ora creati. I sistemi mobili a tutt'oggi supportano la gran parte degli algoritmi di crittografia simmetrica ed asimmetrica, vi è anche il supporto per tutti i meccanismi di autenticazione basata su scambio di chiavi e generazione di chiavi; l'unica mancanza è il supporto ad autenticazioni basate su smart card che però potrà, in breve tempo,

essere sostituito dall'autenticazione tramite chip NFC oppure tramite salvataggio dei dati su SIM.

3.3.1 – Un caso di studio: LPKI in ambiente mobile.

Gli algoritmi a chiave pubblica convenzionali sono in genere lenti e comportano ingenti costi computazionali quindi non sono così adatti per gli ambienti mobili e dispositivi con risorse limitate. Con l'evoluzione di nuove applicazioni come il Mobile Commerce, la diffusione di PKC (*Public Key Cryptography*) per quanto riguarda i dispositivi mobili è inevitabile. Anche se le capacità di elaborazione dei dispositivi mobili è in continua crescita, la progettazione di efficienti tecniche di crittografia è ancora una questione importante dal momento che una buona progettazione può ridurre notevolmente il carico computazionale e quindi non andare ad inficiare troppo sull'autonomia. La *Elliptical Curve Cryptography* (ECC) è considerata come una soluzione adatta per i dispositivi mobili. Tuttavia, le tradizionali infrastrutture a chiave pubblica (PKI) come PKIX e *Wireless Public Key Infrastructure* (WPKI) solitamente sono basati su algoritmi che sfruttano l'elevamento a potenza modulare (RSA) che non è adatto per le dispositivi a risorse limitate. Questo ha fatto si che gli sforzi si concentrassero al fine di utilizzare ECC in PKI.

3.3.1.2 – Elliptic Curve Criptography.

Le soluzioni EC-based si basano generalmente sulla difficoltà di risolvere l'*Elliptic Curve Discrete Logarithm Problem* (ECDLP) e fattorizzazione in curve ellittiche. I Sistemi EC-based possono raggiungere un livello di sicurezza desiderato con chiavi significativamente inferiori a quella di richieste dai loro omologhi basati su esponenziali. Come esempio, è dimostrato che una chiave a 160 bit in un sistema basato su curve ellittiche fornisce lo stesso livello di sicurezza di quello di una chiave a 1024 bit in un sistema RSA-based. Questo è molto efficiente poiché apporta notevoli vantaggi nel salvataggio della chiave in termini di dimensione del certificato, utilizzo della memoria, richiesta di capacità di elaborazione e porta ad un uso efficiente delle potenza, quindi non

va ad alterare di molto l'autonomia, e dello spazio di archiviazione che sono i limiti fondamentali dei dispositivi mobili.

3.3.1.3 – Signcryption.

Il Signcryption è una tecnica crittografica relativamente nuova che dovrebbe soddisfare le funzionalità di *digital signature* e cifratura in un unico passo logico e può efficacemente diminuire i costi computazionali e spese generali di comunicazione in confronto con il tradizionale schemi di *signature-then-encryption*. Nei tradizionali sistemi di crittografia che rispecchiano il pattern *signature-then*, un messaggio è firmato digitalmente quindi seguita da una crittografia che ha due problemi: bassa efficienza ed alto costo.

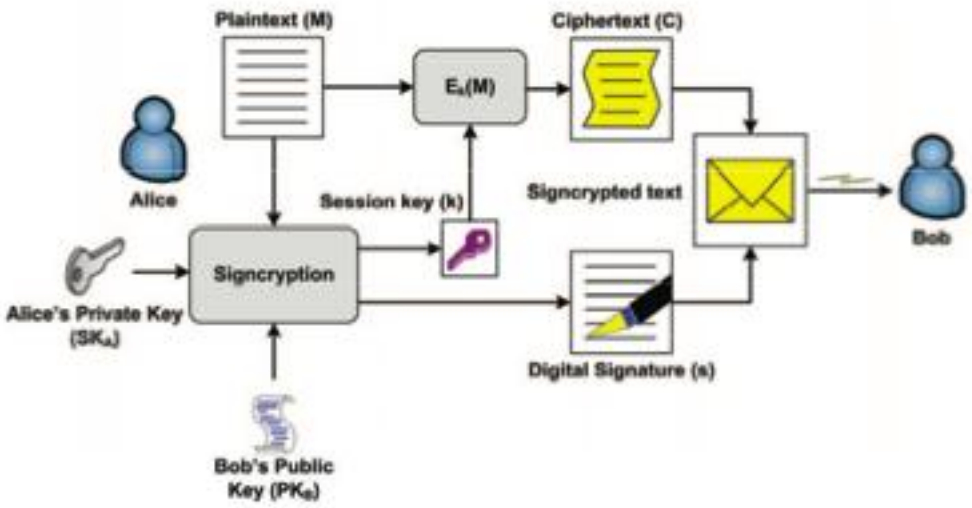

Figura 42 - Lo schema si Signcryption

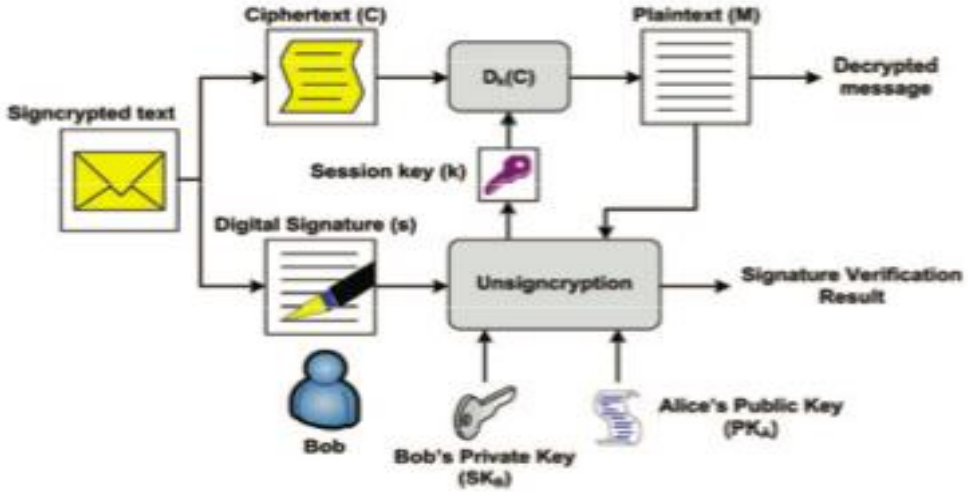

Figura 43 - Lo schema di unsigncryption

3.3.1.4 – L'infrastruttura proposta

PKI è un insieme di hardware, software, persone, politiche, oltre che delle procedure necessarie per creare, gestire, archiviare, distribuire, e revocare i certificati digitali. Le principali differenze tra la soluzione che andremo a vedere e le infrastrutture più popolari come PKIX e WPKI sono le seguenti:

1. L'utilizzo di ECC. Ciò si contrappone alle infrastrutture attualmente disponibili che sono basate sull'elevamento a potenza modulare (RSA).
2. La crittografia simmetrica in cui la PKC è utilizzata per l'impostazione della chiave di sessione. Ciò è opposto a molti dei sistemi attualmente disponibili che utilizzano algoritmi asimmetrici per la crittografia.
3. L'utilizzo di Signcryption ogniqualvolta siano necessarie contemporaneamente sia la crittografia che la firma digitale. Questo si oppone ai tradizionali schemi *signature-then-encryption*.
4. Assegnazione di una sola coppia di chiavi private-pubbliche per ciascun sottoscrittore. Questo si oppone ad alcune infrastrutture come WPKI che dedica due coppie di chiavi private-pubbliche per ciascun sottoscrittore, una coppia per la crittografia e l'altra coppia per la firma digitale.
5. L'introduzione di un nuovo componente chiamato *Validation Autority*(VA) per eseguire le convalide delle richieste di identità.
6. Tenendo conto di diverse considerazioni come la convalida della chiave pubblica che sono essenziali nei sistemi EC-based e non sono considerate nelle PKI tradizionali.

Tali miglioramenti possono effettivamente ridurre i costi computazionali e le spese generali di comunicazione. Dato che LPKI si basa su ECC, dobbiamo prendere in considerazione vari fattori che non rientrano nelle infrastrutture tradizionali basati su algoritmi derivati dall'elevamento a potenza.

3.3.1.4.1 – I componenti di LKPI

I componenti software di LKPI sono i seguenti:

- *Registration Authority* (RA): si occupa di registrare i dati del sottoscrittore e l'emissione di un identificatore univoco per ogni abbonato.

- *Certification Authority* (CA): si prende cura dell'emissione e gestione dei certificati.
- I certificati digitali: stringa d'informazione che lega l'identificatore univoco di ogni abbonato al suo/la sua corrispondente chiave pubblica.
- *Certificate Repository* (CR): il repository di tutti i certificati.
- *OCSP server*: si occupa di rispondere alle richieste riguardo delle informazioni sullo stato della revoca o meno di un certificato. Esso può essere considerato come una parte della CA.
- *Validation Authority* (VA): un *Trusted Third Party* (TTP) che compie tutte le convalide necessarie per le entità finali che richiedono la validazione del certificato.
- *Key Generating Server* (KGS): un componente opzionale che si occupa di generare le chiavi pubbliche-private e può essere considerato come una parte del RA. Non è richiesto se le politiche di sicurezza permettono alle end-entities di generare le proprie chiavi pubbliche-private e se hanno adeguate capacità per farlo.
- *End-Entities*: Mobile Equipments (ME), come telefoni cellulari e PDA con i seguenti compiti:
    - Generazione di una coppia di chiavi pubblica-privata del sottoscrittore, aaplicando per la prima volta il rilascio della certificazione, possono richiedere l'aggiornamento ed il rinnovo della coppia di chiavi e certificati;
    - Memorizzazione e possibilità di accesso alla coppia di chiavi pubblica-privata del sottoscrittore;
    - Recupero di un certificato e del suo stato di revoca, facendo le validazioni necessarie tra cui la convalida dei certificati e convalida delle chiavi;
    - Prendersi cura dei meccanismi di sicurezza.
- *Server Timestamp*: Produce informazioni di temporizzazione e validazione per tutti i componenti del LPKI.

3.3.1.4.2 – Il generatore di chiavi pubblica/privata

Per ogni utente U, la chiave privata è un intero selezionato casualmente

$SK_{U \in R}$ [1,n-1], e la chiave pubblica è un punto della curva ellittica che si genera come $PK_U = SK_U G$. Ogni sottoscrittore è identificato con un identificativo univoco (IDU). per il caso delle reti mobili, l'identificatore univoco può essere il numero di telefono internazionale del sottoscrittore. LPKI supporta due modi di generare la chiave pubblica:

Modo 1: Generare le chiavi pubbliche-private nel KGS;

Modo 2: Far generare le chiavi pubbliche-private dall'End-Entities;

Nella prima modalità, le chiavi pubbliche-private vengono generati nei KGS attendibili. Le chiavi pubbliche generate vengono poi certificati dalla CA e le informazioni appropriate vengono memorizzati direttamente e in modo sicuro sulla smart card dell'abbonato, ad esempio, la SIM. Secondo le politiche di sicurezza, la CA può memorizzare la chiave privata e di altre informazioni pertinenti in un modulo sicuro, come il *Key Backup Center* (anche il CR), al fine di utilizzarlo per eventuali recuperi delle chiavi.

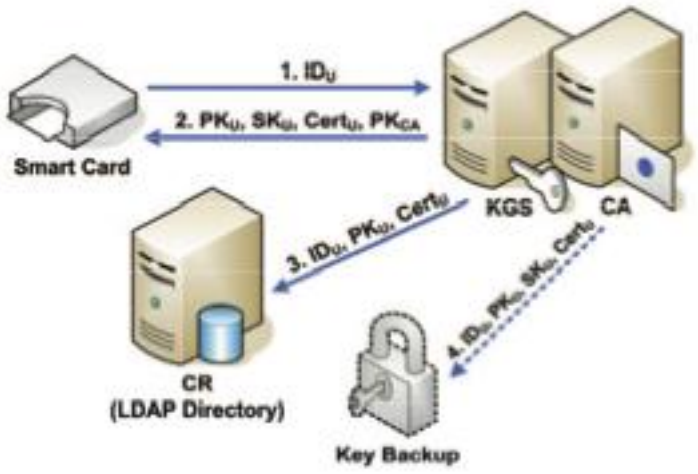

Figura 44 - Il modo 1

Nella seconda modalità, le chiavi vengono generate dalle End-Entities, e vengono memorizzati nelle loro smart card o altri moduli di sicurezza. La CA

quindi certificare il pubblico generato chiavi dopo prove adeguate. Il CA deve verificare che gli end-entities possiedono realmente le corrispondenti chiavi private del loro chiavi pubbliche. Ciò può essere realizzato mediante una tecnica *zero-knowledge*.

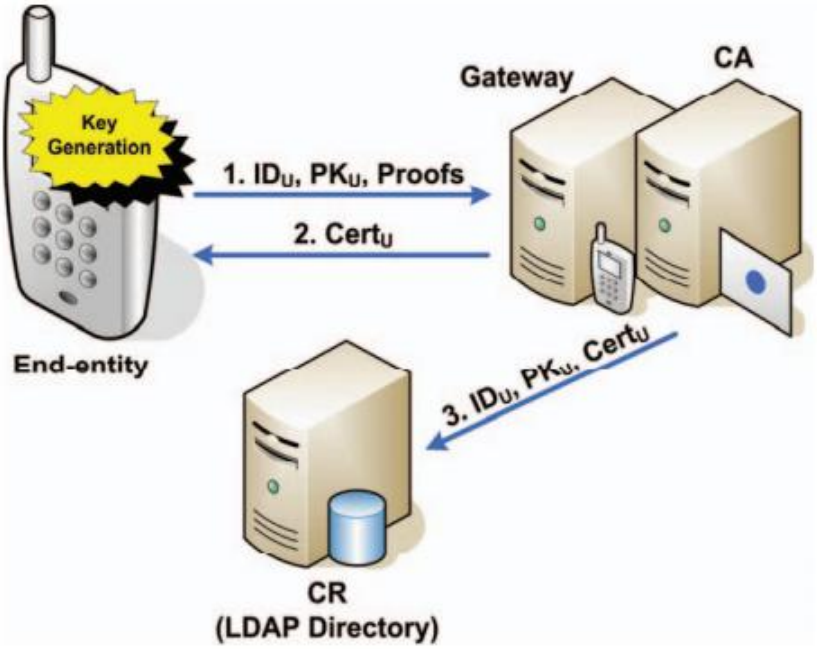

Figura 45 - Il modo 2

Poiché i dispositivi con risorse limitate sono non in grado di generare numeri casuali molto "forti" per essere utilizzati come chiavi private, ed a causa delle vulnerabilità intrinseche dell'interfaccia wireless, il primo approccio ha un evidente vantaggio per quanto riguarda la sicurezza sulla seconda. Tuttavia, la seconda modalità prevede maggiore flessibilità in quanto le procedure di aggiornamento automatico di chiavi, e rinnovo del certificato possono essere facilmente realizzate.

3.3.1.4.3 – I certificati digitali

Ogni abbonato dovrebbe ottenere un certificato digitale dal CA $Cert_U$ per il suo/la sua chiave pubblica. Nel LPKI, ogni subscriber è associato ad una sola coppia di chiavi pubblica-privata così il CA rilascia un solo certificato per ogni sottoscrittore. LPKI utilizza lo stesso formato dei certificati X.509v3.

3.3.1.4.4 – Il Repository di certificati

Per ciascun abbonato, sono memorizzate in RC una serie di informazioni tra cui l' identificativo univoco, la chiave pubblica, e il certificato. Il CR nel LPKI memorizza i certificati con il protocollo LDAP. Questo era inizialmente sviluppato per l'accesso, l'esecuzione di query e la manipolazione i servizi di directory X.500, supporta TCP/IP, utilizza una struttura gerarchica ad albero, e rappresenta un servizio leggero, di directory veloce e scalabile. Nel protocollo LDAP le registrazioni devono essere aggiornate al momento opportuno, quando vi è il rilascio o la revoca di un certificato.

3.3.1.4.5 – Il meccanismo di sicurezza

La riservatezza, l'autenticazione, integrità sono i servizi di sicurezza più ambiti per molti sistemi che sono forniti con i meccanismi di sicurezza, come crittografia e firma digitale. Nel LPKI, tutti i messaggi vengono criptate tramite un algoritmo di crittografia simmetrica sicuro come ad esempio AES in cui è stabilito la sua chiave segreta utilizzando la chiavi pubbliche-private delle entità coinvolte. Questo può essere realizzato utilizzando gli schemi Signcryption che devono avere solo una coppia di chiavi per eseguire simultaneamente la criptazione e la firma digitale.

3.3.1.4.6 – La validazione dei certificati

Qualsiasi chiave pubblica non è affidabile se non viene verificata utilizzando il corrispondente certificato di convalida. La validazione comprende principalmente:

1. Verificare l'integrità e l'autenticità del certificato verificando la firma del CA su di esso.
2. Verifica che il certificato non è scaduto.
3. Verifica che il certificato non è stato revocato.

Per poter eseguire la convalida del certificato, è necessaria per ottenere lo stato di validità del certificato. Generalmente, la stato di validità può essere ottenuto usando uno dei Certificate Revocation List (CRL), Delta CRL, o Status Certificate Protocol (OCSP). L'OCSP è la soluzione più redditizia per i dispositivi con risorse limitate che non sono in grado di salvare una troppo

grande CRL, un secondo vantaggio lo troviamo nei tempi di risposta molto bassi, LPKI ha introdotto l'OCSP server come uno dei suoi componenti principali. Le risposte OCSP sono firmate digitalmente con una chiave privata, la cui chiave pubblica corrispondente è ben nota alle End-Entities.

3.4 – Lo stato dell'arte della Crittografa sul sistema operativo mobile Android

Android è uno dei sistemi operativi mobile più diffusi, la cui architettura si accosta maggiormente ad un sistema desktop; come abbiamo già detto il sistema si basa su un kernel linux e una jvm rimodellata (la Dalvik Virtual Machine), questo ha facilitato di molto il porting e l'implementazione delle varie librerie di crittografia. Le librerie di crittografia che sono state implementate sia utilizzando le API Java messe a disposizione dal SDK nativo, sia le API native rese disponibili dal NDK.

3.4.1 – Il porting di javax.crypto

Il package *javax.crypto* è un package molto apprezzato dalla comunità di sviluppatori in quanto è un porting diretto della versione J2SE (*Java 2 Standard Edition*); questo rende molto più facile la traduzione di interi moduli software orientati alla sicurezza e all'utilizzo degli algoritmi di crittografia. Il package è composto da varie interfacce e sottopackage che implementano i vari meccanismi ed algoritmi di crittografia:

- *Javax.crypto*: Questo pacchetto fornisce le classi e le interfacce per applicazioni crittografiche impementando algoritmi per la crittografia, decrittografia o meccanismi basati su accordo di chiavi.
- *Javax.crypto.interfaces*: Questo pacchetto fornisce le interfacce necessarie per implementare gli algoritmi basati su accordo di chiavi.
- *Javax.crypto.spec*: Questo pacchetto fornisce le classi e le interfacce necessarie per definire le chiavi ed i parametri per gli algoritmi di crittografia.

Il package è presente nel SDK sin dalla prima versione, non ha subito mai alcuna revisione; così facendo non vi è stata alcuna difficoltà a sviluppare applicazione che si interfacciavano con sistemi a chiave condivisa oppure a

scambio di chiavi fin dalle prime versioni di Android. Il package, come il suo omonimo J2SE, supporta algoritmi di crittografia simmetrica (nell'esempio seguente AES):

```
public class SymmetricAlgorithmAES extends Activity {

    static final String TAG = "SymmetricAlgorithmAES";

    @Override

    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.symm);

        // Original text

        String theTestText = "This is just a simple test";

        TextView tvorig = (TextView)findViewById(R.id.tvorig);

        tvorig.setText("\n[ORIGINAL]:\n" + theTestText + "\n");

        // Set up secret key spec for 128-bit AES encryption and decryption

        SecretKeySpec sks = null;

        try {

            SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");

            sr.setSeed("any data used as random seed".getBytes());

            KeyGenerator kg = KeyGenerator.getInstance("AES");

            kg.init(128, sr);

            sks = new SecretKeySpec((kg.generateKey()).getEncoded(), "AES");


        } catch (Exception e) {

            Log.e(TAG, "AES secret key spec error");

        }

        // Encode the original data with AES

        byte[] encodedBytes = null;

        try {

            Cipher c = Cipher.getInstance("AES");

            c.init(Cipher.ENCRYPT_MODE, sks);

            encodedBytes = c.doFinal(theTestText.getBytes());

        } catch (Exception e) {

            Log.e(TAG, "AES encryption error");

        }

        TextView tvencoded = (TextView)findViewById(R.id.tvencoded);

        tvencoded.setText("[ENCODED]:\n" +
```

```
            Base64.encodeToString(encodedBytes, Base64.DEFAULT) + "\n");


        // Decode the encoded data with AES
        byte[] decodedBytes = null;
        try {
            Cipher c = Cipher.getInstance("AES");
            c.init(Cipher.DECRYPT_MODE, sks);
            decodedBytes = c.doFinal(encodedBytes);
        } catch (Exception e) {
            Log.e(TAG, "AES decryption error");
        }
        TextView tvdecoded = (TextView)findViewById(R.id.tvdecoded);
        tvdecoded.setText("[DECODED]:\n" + new String(decodedBytes) + "\n");
    }
}
```

E supporta anche algoritmi di crittografia asimmetrica (nell'esempio RSA):

```
public class AsymmetricAlgorithmRSA extends Activity {
    static final String TAG = "AsymmetricAlgorithmRSA";
    @Override
    public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.asym);
    // Original text
    String theTestText = "This is just a simple test!";
    TextView tvorig = (TextView)findViewById(R.id.tvorig);
    tvorig.setText("\n[ORIGINAL]:\n" + theTestText + "\n");
    // Generate key pair for 1024-bit RSA encryption and decryption
    Key publicKey = null;
    Key privateKey = null;
    try {
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(1024);
        KeyPair kp = kpg.genKeyPair();
        publicKey = kp.getPublic();
        privateKey = kp.getPrivate();
```

```
    } catch (Exception e) {
        Log.e(TAG, "RSA key pair error");
    }
    // Encode the original data with RSA private key
    byte[] encodedBytes = null;
    try {
        Cipher c = Cipher.getInstance("RSA");
        c.init(Cipher.ENCRYPT_MODE, privateKey);
        encodedBytes = c.doFinal(theTestText.getBytes());
    } catch (Exception e) {
        Log.e(TAG, "RSA encryption error");
    }
    TextView tvencoded = (TextView)findViewById(R.id.tvencoded);
    tvencoded.setText("[ENCODED]:\n" +
        Base64.encodeToString(encodedBytes, Base64.DEFAULT) + "\n");
    // Decode the encoded data with RSA public key
    byte[] decodedBytes = null;
    try {
        Cipher c = Cipher.getInstance("RSA");
        c.init(Cipher.DECRYPT_MODE, publicKey);
        decodedBytes = c.doFinal(encodedBytes);
    } catch (Exception e) {
        Log.e(TAG, "RSA decryption error");
}
```

Come possiamo vedere dagli esempi oltre al logger ed i metodi che vanno implementati per definire una nuova attività, tutto il codice di sicurezza rispecchia il paradigma POJO (*Plain Old Java Object*).

3.4.2 – Spongy Castle

Spongy Castle è il porting su piattaforma Android di Bouncy Castle, di cui abbiamo già parlato precedentemente; questo porting è stato necessario in quanto il classloader di Android andava in conflitto con le librerie originali di Bouncy Castle e ne rendeva difficile l'installazione di una versione aggiornata. Questo package supporta comunque a pieno tutti gli algoritmi precedentemente supportati da Bouncy Castle, il package si comporta solo da sistema di nomi alternativo:

- tutti i nomi dei pacchetti sono stati spostati da org.bouncycastle.* a org.spongycastle.* al fine di evitare conflitti classloader;
- Il nome del *Java Security API Provider* è SC invece che BC;
- nessun cambiamento dei nomi di classe, in modo che la classe BouncyCastleProvider rimane Bouncy, non Spongy, ma è stata sposta sul pacchetto org.spongycastle.jce.provider.

Lo sviluppo di questa libreria va di pari passo con lo sviluppo di Bouncy Castle, con un ovvio delay di tempo per l'adattamento dei nomi e di alcune funzionalità su Android.

3.4.3 – ULTRA Android Library

ULTRA Android Library è una libreria crittografica, basta su Elliptic Curve Cryptography, ottimizzata per piattaforma Android. La libreria è ottimizzata per ARMv7, MIPS. ULTRA Android Library serve allo sviluppo di applicazioni mobili con requisiti di sicurezza elevati nella trasmissione ed storage di informazioni sensibili/critiche.

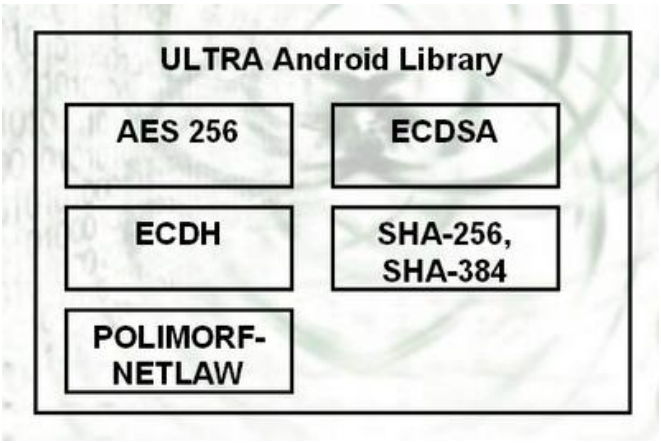

Figura 46 - Gli algoritmi supportati da ULTRA Android Library

ULTRA Android Library implementa le specifiche crittografiche dettate dalla NSA Suite B:

- Advanced Encryption Standard (AES) con chiavi a 256 bit, algoritmo di crittazione simmetrica;
- Elliptic Curve Digital Signature Algorithm (ECDSA) , algoritmo per la firma digitale;

- Elliptic Curve Diffie-Hellman (ECDH), algoritmo per lo scambio di chiavi;
- Secure Hash Algorithm 2 (SHA-256, SHA-384), algoritmo per il calcolo di hash

ULTRA Android Library include:

- Engine crittografico nativo ad elevate prestazioni Elliptic Curves Cryptography, ottimizzato per le seguenti architetture native: ARMv7, MIPS, x86
- Software nativo VPN;
- Serializzazione ed interscambio sicuro di oggetti con le altre librerie ULTRA

ULTRA Android Library , attraverso gli algoritmi crittografici implementati, permette di comunicare in modo sicuro attraverso i seguenti canali di comunicazione:

- WiFi;
- Reti private 3G/4G LTE
- Reti pubbliche 3G/4G LTE
- SMS (testo e/o binari)
- NFC
- Bluetooth

Questa libreria si propone di inserirsi come layer tra le varie interfacce di comunicazione criptando a priori i messaggi e successivamente sfruttando le interfacce con una comunicazione standard.

3.4.4 – La crittografia a blocchi implementata con NDK, un caso di studio

L'utilizzo del NDK permette di scrivere codice più efficiente, questo vale anche per gli algoritmi di crittografia; per quest'ultima però l'efficienza aumenta all'aumentare della complessità e del numero di bit coinvolti nell'algoritmo. Esamineremo qui di seguito un esempio di cifrario a blocchi realizzato con il codice nativo.

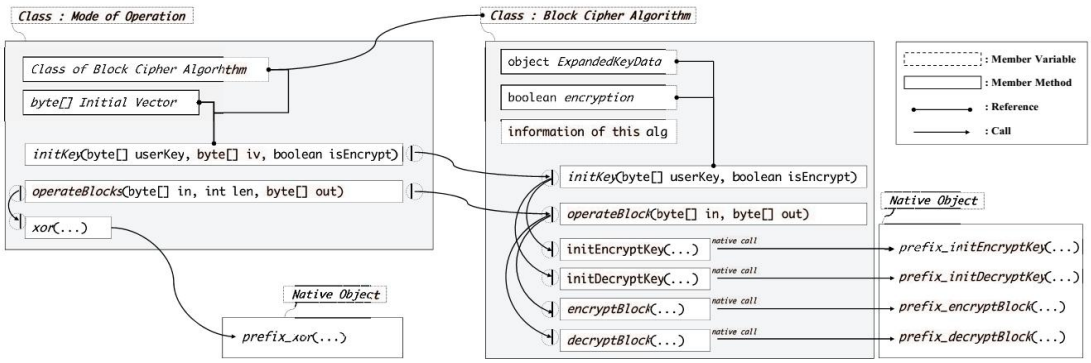

Figura 47 - Il diagramma delle classi dell'implementazione del cifrario

La figura 9 ci mostra il diagramma delle classi per quanto riguarda l'implementazione del cifrario:

- *Class - Block Cipher Algorithm*: Questa classe specifica l'algoritmo di cifratura a blocchi. Vi sono tre variabili di sistema che sono *ExpandedKeyData*, *encryption* e le informazioni di questo algoritmo. *ExpandedKeyData* contiene le chiavi che vengono utilizzate da *initKey ()*. *initKey()* chiama internamente l'interfaccia nativa per lo scheduling della chiave. La variabile di sistema *encryption* definisce se l'istanza è per la crittografia o no. Le informazioni di questo algoritmo sono variabili e il blocco. Il metodo *operateBlock()* è la cifratura o decifratura del blocco singolo basata su funzioni native.
- *Class - Mode of Operation:* Questa classe implementa la modalità di funzionamento, come CBC (*Cipher Block* Chaining) e CTR (*Counter*). Questo classe ha un'unica istanza di *Class - Block Cipher Algorithm* ed il suo vettore iniziale. *initKey()* inizializza questa classe e chiama *initKey()* dell'istanza di *Block Cipher Algorithm*. *operateBlocks()* implementa modalità di funzionamento usando *operateBlock ()* e la lo XOR nativo.

Questa architettura migliora le prestazioni e la flessibilità di Java. L'idea principale di migliorare l'efficienza dell'attuazione del metodo per trasformare i dati primitivi del linguaggio Java. È perché il dato primitivo array (ad esempio *byte []*) in java deve essere trasformato nel tipo di dato array

dell'ambiente nativo quando vengono tradotti per l'ambiente su cui sta girando la JVM. La scelta è ricaduta su *GetPrimitiveArrayCritical()* funzione che può ottenere direttamente un riferimenti ad i dati primitivi. Perché le altre funzioni per l'elaborazione di dati primitivi, come *Get<Type>ArrayRegion()* e *Get<Type>ArrayElements()*, duplicano il payload dell'array primitivo, quindi ritardano l'intera performance del cifrario.

3.5 – Le applicazioni che fanno uso della crittografia sul Play Store

Le applicazioni presenti sul Google Play Store che fanno uso di crittografia sono molte e con funzionalità simili, possono essere suddivise in tre categorie:

1. Applicazioni che fanno uso della crittografia per criptare file locali (ed eventualmente caricarli su un server remoto);
2. Software che tramite gli algoritmi di crittografia simmetrica permettono di criptare del testo ed eventualmente inviarlo tramite i canali classici di comunicazione (SMS, Mail…);
3. Password Manager che salvano le credenziali di accesso ad applicazioni web ed altri dati sensibili dell'utente sulla memoria fisica del dispositivo, tramite gli algoritmi di crittografia fanno in modo da non renderli leggibili.

3.5.1 – Crypto

Crypto è un applicazione che provvede a criptare i tuoi file e nasconderli al sistema.

Utilizza una serie di algoritmi ed è stata sviluppata per proteggere al meglio i tuoi dati, permette di scegliere tra la cancellazione dell'estensione nascondendolo al sistema operativo fino ad una crittografia DES completa.

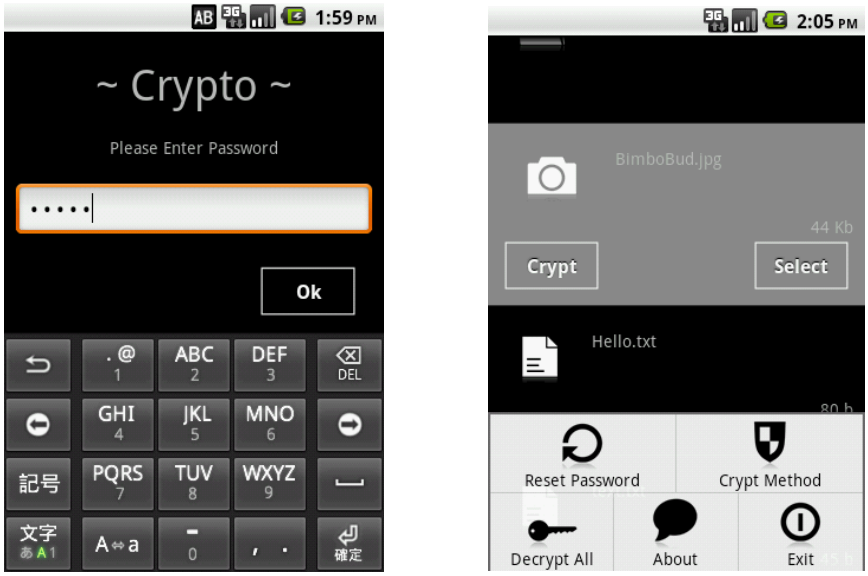

Figura 48 - La password (per criptare) e l'inserimento del file da criptare

3.5.2 – Crypto SMS

Crypto SMS è un'applicazione semplice per inviare messaggi di testo (SMS cifrati) tra gli utenti Android. Crypto SMS utilizza AES a 128-bit Encryption Algorithm che è un algoritmo estremamente efficace per la protezione dei dati sensibili in una comunicazione che fa uso di algoritmi di crittografia simmetrici. I destinatari devono avere Crypto SMS installato sul loro dispositivo mobile, inoltre è necessario precondividere la chiave con il destinatario degli sms in modo dato che si fa uso di un algoritmo a chiave pubblica condivisa (PSK). Con questa applicazione è possibile inviare le informazioni di conti bancari,, credenziali di accesso e dati personali sensibili in modo sicuro.

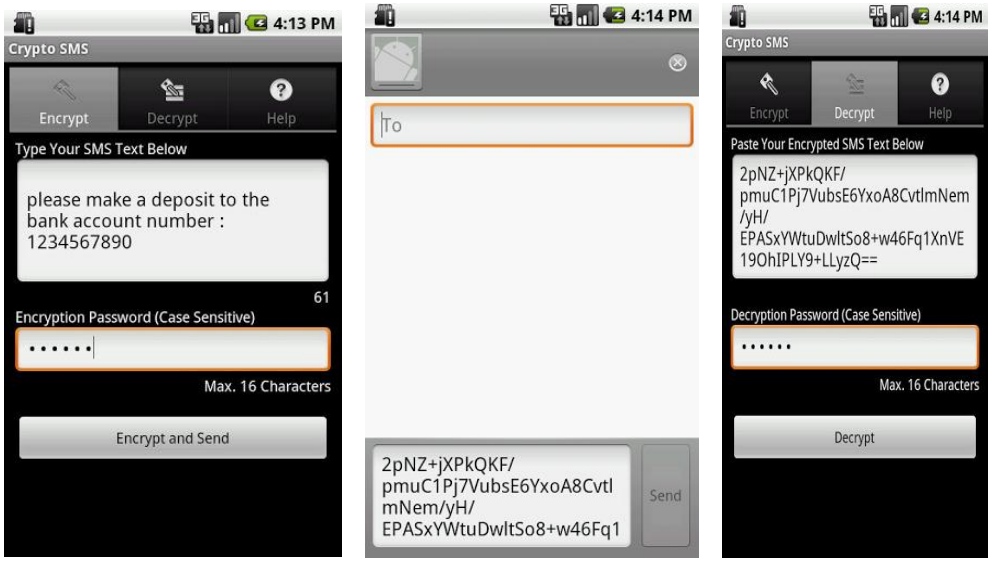

Figura 49 - Le tre fasi della comunicazione criptata

L'applicazione però non rende sicuro anche il canale di comunicazione, bensì trasmette in chiaro dati cifrati (in quanto il protocollo SMS non consente comunicazioni cifrate); quindi un eventuale attaccante può leggere quello che viene inviato ma impiegherebbe troppo tempo a decifrarlo (circa $1{,}02 \times 10^{18}$ anni).

3.5.3 – mSecure Password Manager

mSecure Password Manager appartiene alla categoria di applicazioni che fa uso della crittografia per salvaguardare dati sensibili quali le password dei vari account (mail, instant messaging, ftp, banca…) di sua proprietà ed fa uso anche di un generatore di numeri casuali per generare password sicure (che verranno a loro volta criptate).

Il software utilizza una codifica Blowfish a 256 bit (un algoritmo crittografico simmetrico a blocchi) , l'applicazione inoltre prevede dei meccanismi di protezione ulteriori in caso di smarrimento del dispositivo:

- Dati sincronizzati nel Cloud (*Dropbox*) per una disponibilità anche su altre piattaforme (desktop, notebook) ;
- Distruzione del file di dati dopo un numero di tentativi di inserimento della password falliti;
- Blocco automatico dell'applicazione quando si va in multitasking.

Il file di dati viene salvato in locale e sincronizzato, o manualmente o in push, con la piattaforma cloud passando da un canale sicuro.

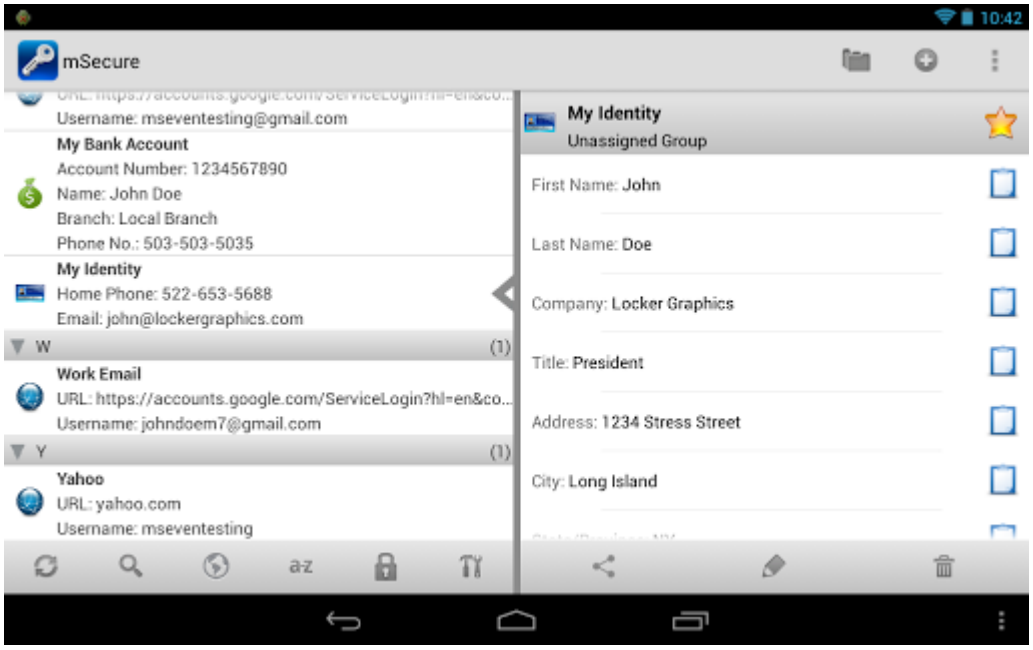

Figura 50 - La visualizzazione di dati in mSecure

Il software inoltre dispone di un browser interno dove visualizzare (tramite canale sicuro) l'interfaccia web del sistema a cui si vuole accedere tramite i dati salvati nell'applicazione.

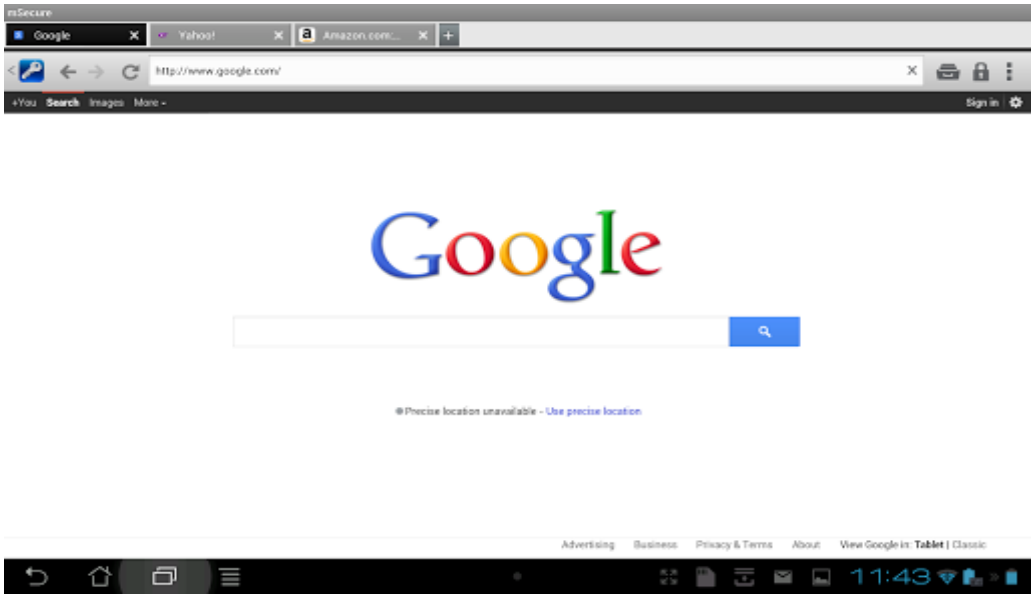

Figura 51 - Il browser

Capitolo 4 – Droidcare, un esempio di accesso a Personal Health Record

4.1– Introduzione al progetto

4.1.1 – I requisiti

I requisiti del progetto di questa applicazione sono molteplici, sono orientati alla sicurezza dei dati, alla riservatezza di quest'ultimi e alla possibilità di tenerli aggiornati tramite un server. La lista dei requisiti è la seguente:

- L'applicazione deve criptare qualsiasi file presente sulla memoria fisica riguardante la cartella clinica dell'utente;
- La criptazione deve essere effettuata tramite algoritmo di crittografia robusto, ma non ad alto costo computazionale in modo da non inficiare drasticamente sull'autonomia del dispositivo;
- La visualizzazione dei dati clinici deve essere subordinata ad un "login" effettuato tramite protocollo NFC;
- Il login deve convalidare la decriptazione dei file presenti sulla memoria fisica del dispositivo;
- Il login deve essere effettuato tramite la lettura di un tag NFC con sopra salvata la password sotto forma di hash;
- Il software al primo avvio deve scrivere il tag NFC, con specifiche decise arbitrariamente, in modo da salvarci l'hash della password che sarà immessa dall'utente (in un futuro aggiornamento è possibile far si che in caso di smarrimento del tag l'utente possa scriverne un altro digitando la password, sfruttando la proprietà degli hash);
- L'applicativo deve connettersi ad un server che supporti lo standard HL7v2 per la sincronizzazione delle informazioni riguardante la cartella clinica dell'utente, dopo l'update queste informazioni devono venire criptate ed il file criptato deve venir salvato su memoria fisica;
- Il software deve mostrare a video il contenuto dei file della cartella clinica in apposite view che rendano comprensibile il contenuto del file, solo e unicamente dopo le fasi dette precedentemente;

Questi sono i requisiti su cui mi sono basato per lo sviluppo dell'applicazione, ovviamente sono state fatte delle scelte implementative che lasciano aperte più strade per un futuro aggiornamento dell'applicazione.

4.1.2 - Descrizione

Il progetto che ho modellato consta nello sviluppare un'applicazione per il sistema operativo mobile Android, che vada ad interagire ed operare con lo standard HL7v2 (*Health Level 7*). Le funzionalità principali dell'applicazione prevedono la visualizzazione della propria cartella clinica tramite dei file che risiedono in memoria fisica, sincronizzazione di questi con un server che comunica tramite il protocollo HL7v2; i file che vengono visualizzati rappresentano dei dati molto sensibili, pertanto l'applicazione prevede di salvarli criptati sul dispositivo e decriptarli solo all'occorrenza.

L'applicativo prevede inoltre una procedura di login, al fine di evitare che chiunque entri in possesso del dispositivo mobile possa visualizzare liberamente le informazioni salvate. La scelta per la procedura di login è ricaduta sul protocollo NFC e l'autenticazione tramite lettura di un tag formattato secondo specifiche da noi imposte.

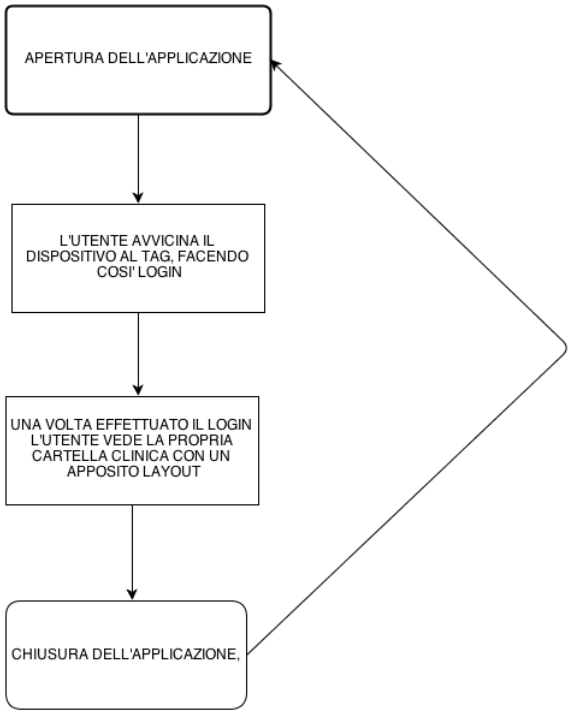

Figura 52 - Il grafico di interazione con l'utente

4.1.2– La struttura del progetto

L'applicazione è suddivisa in quattro diversi package ognuno con compiti diversi:

- *com.tesi.crypto.engine*: è il package dedicato al vero e proprio motore che si occupa di criptare i file e generare il file di hash utile per fare il confronto con il tag di login;
- *com.tesi.crypto.controller*: questo package ha la funzione di controller, simile al concetto di controller del pattern MVC (*Model-View-Controller*), per quanto riguarda il confronto tra l'hash salvato e quello che viene letto dal tag NFC;
- *com.tesi.hl7*: qui sono wrappate tutte le funzioni di interazione e comunicazione con il protocollo HL7, oltre che il parsing e la preparazione dei risultati in modo da poterli incapsulare in una view appropriata;
- *com.tesi.droidcare*: il cuore dell'applicazione, qui vi sono implementate tutte le activity e tutta la logica applicativa del software ed il suo workflow;

Il package *com.tesi.droidcare* sfrutta gli altri moduli per la logica applicativa, delegando a sua volta ad ogni activity implementata un compito preciso e distinto dalle altre. Gli altri moduli sono indipendenti fra loro e possono essere riutilizzati in altri software che prevedono funzionalità simili, grazie anche all'estrema portabilità del codice Java e al SDK di Android che prevede l'uso di questo linguaggio di programmazione; i package *com,tesi.crypto.** sono stati scritti per un'applicazione Desktop orientata alla semplice criptazione dei file e successivamente ne è stato fatto il porting per il progetto.

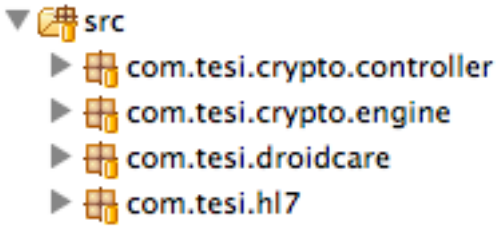

Figura 53 - La struttura dei package

4.1.3– Gli strumenti utilizzati

Lo sviluppo di questo progetto ha impiegato diversi software, la tipologia di questi software spazia tra l'ambiente di sviluppo, integrato al SDK Android, Eclipse al simulatore OpenNFC per mimare il comportamento dell'avvicinamento ed allontanamento di un tag NFC (sia in scrittura che in lettura) fino alla creazione e configurazione di un applicativo che svolgesse le mansioni di server.

4.1.3.1 – Eclipse e SDK Android

L'ambiente di sviluppo Eclipse ADT (*Android Developer Tools*)incluso nel SDK Android si è rivelato essere uno strumento molto efficace nella gestione e nella creazione di Activity, Intent, file di configurazione e molto altro per lo sviluppo di una Applicazione Android; questo pacchetto, come già detto precedentemente, è reso disponibile da Google tramite il sito dedicato agli sviluppatori Android. L'ambiente di programmazione è una versione di Eclipse con preinstallato il plugin ADT, quindi espandibile ed aggiornabile con tutti i plugin disponibili per facilitare lo sviluppo con altri tipi di risorse (es. Enterprise Server).

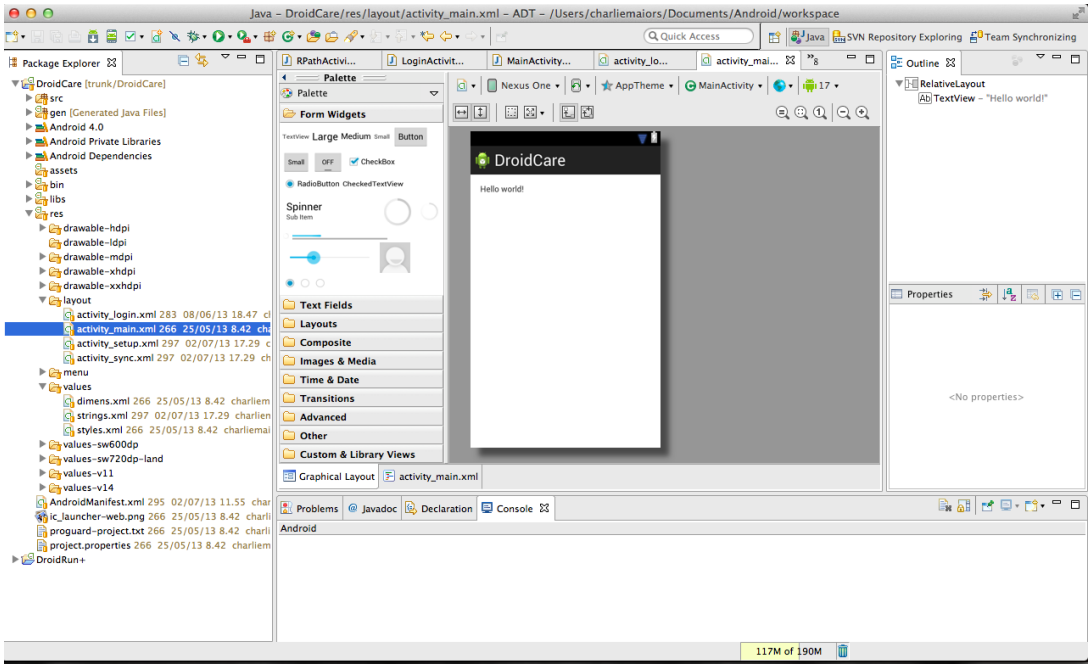

Figura 54 - L'interfaccia di Eclipse ADT nella progettazione di una Activity

Come possiamo vedere dalla figura 3, il plugin ADT implementa anche un designer per disegnare le interfacce grafiche delle Activity e generare il codice XML con tutte le proprietà che andiamo a definire.

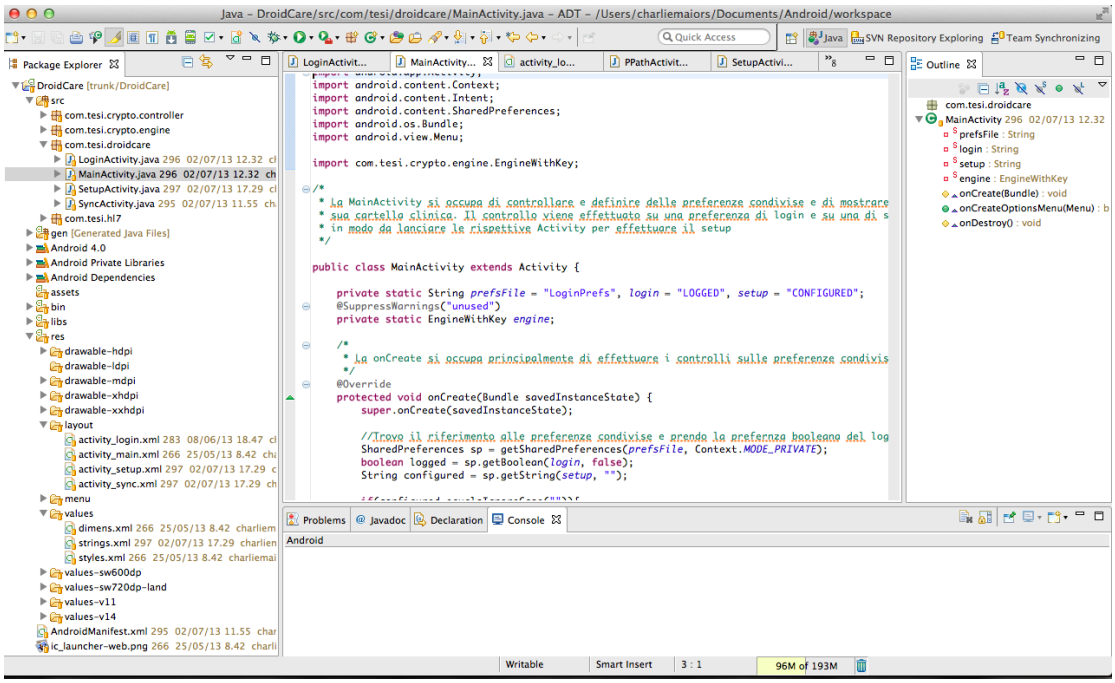

Figura 55 - La scrittura del codice Java dell'applicazione

L’ambiente alla creazione di una nuova attività crea oltre al file xml per la gestione della GUI, anche il file Java corrispondente e ne realizza il mappaggio nei file di configurazione dell’applicazione; nel file Java noi andremo ad implementare tutta la logica di gestione dell’applicazione, la gestione degli eventi e degli intenti (esterni).

Il plugin ADT inoltre integra i comandi per la gestione del SDK (*SDK Manager*) e degli AVD (*Android Virtual Device*)

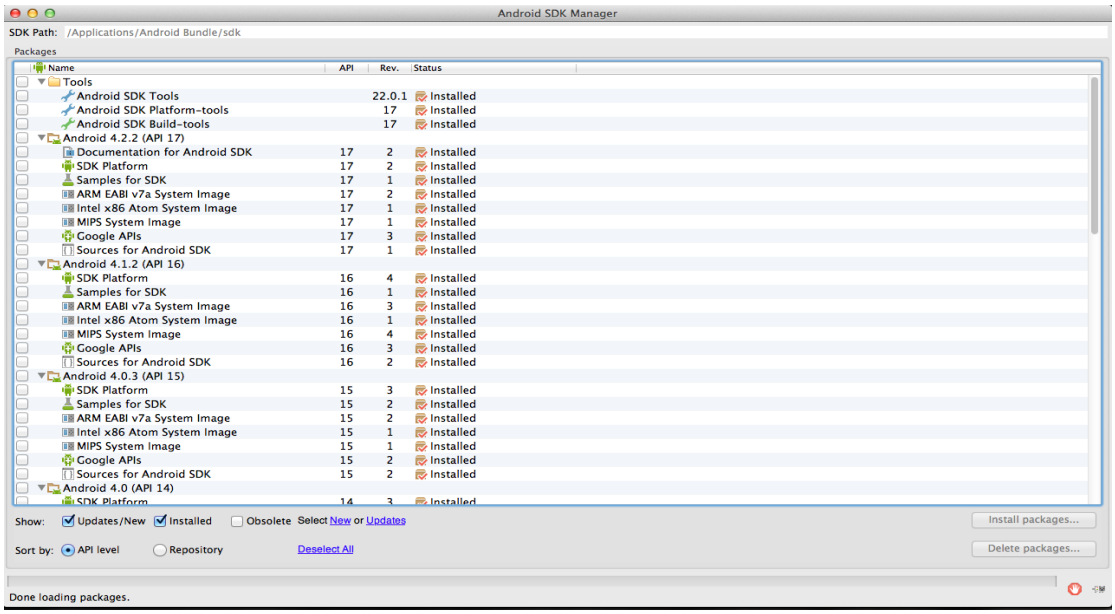

Figura 56 - SDK Manager

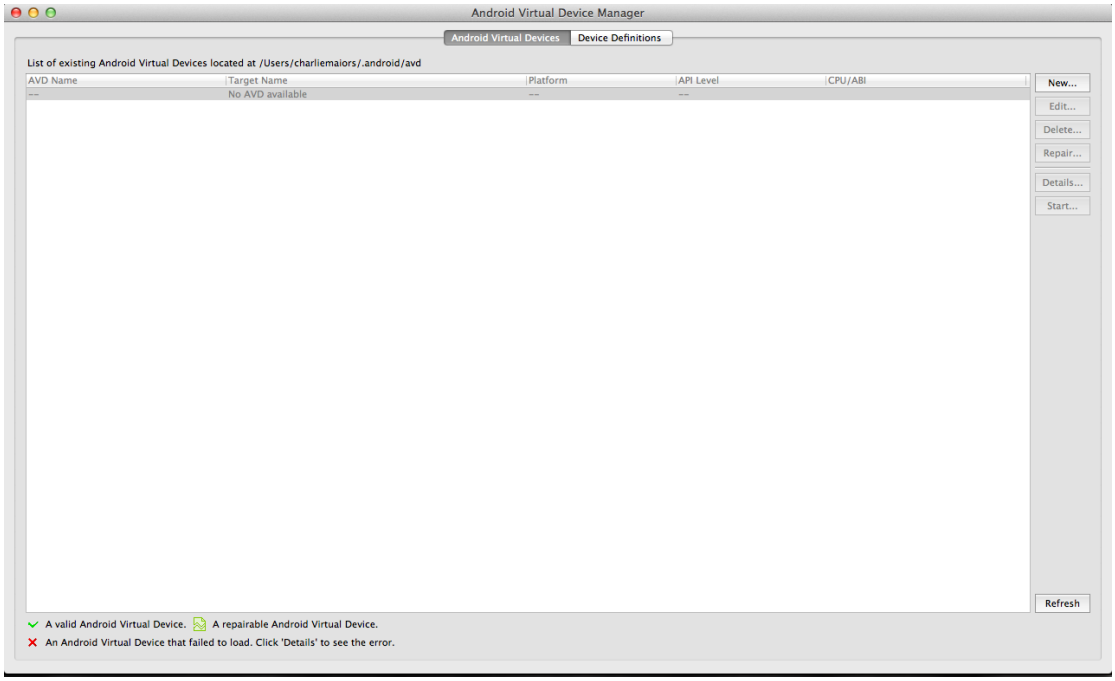

Figura 57 - AVD Manager

4.1.3.2 – Open NFC

Open NFC è uno stack software gratuito che implementa le funzionalità di NFC su di un chipset del controller NFC. Le sue caratteristiche principali sono:

- Supporta una vasta gamma di tag e protocolli NFC attraverso interfacce di alto e di basso livello;

- Supporta diverse modalità di funzionamento di NFC: la modalità Reader, modalità di emulazione della scheda, e la modalità peer-to-peer;
- Supporta il connection handover, tra pairing Bluetooth e accoppiamento Wifi;
- Supporta l'emulazione della scheda su elementi sicuri e integra una pila di sicurezza per filtrare l'accesso a questi elementi;
- Stack NFC indipendente dall'hardware, basato su un NFC Hardware Abstraction Layer. Sono forniti moduli NFC HAL per alcuni chipset, tra cui un simulatore software NFC;
- Codice portabile, con diversi porting di riferimento in base alle varie edizioni del software. Lo stack è disponibile in tre diverse architetture per adattarsi meglio ai diversi ambienti: monolitico, user-driver o client-server;

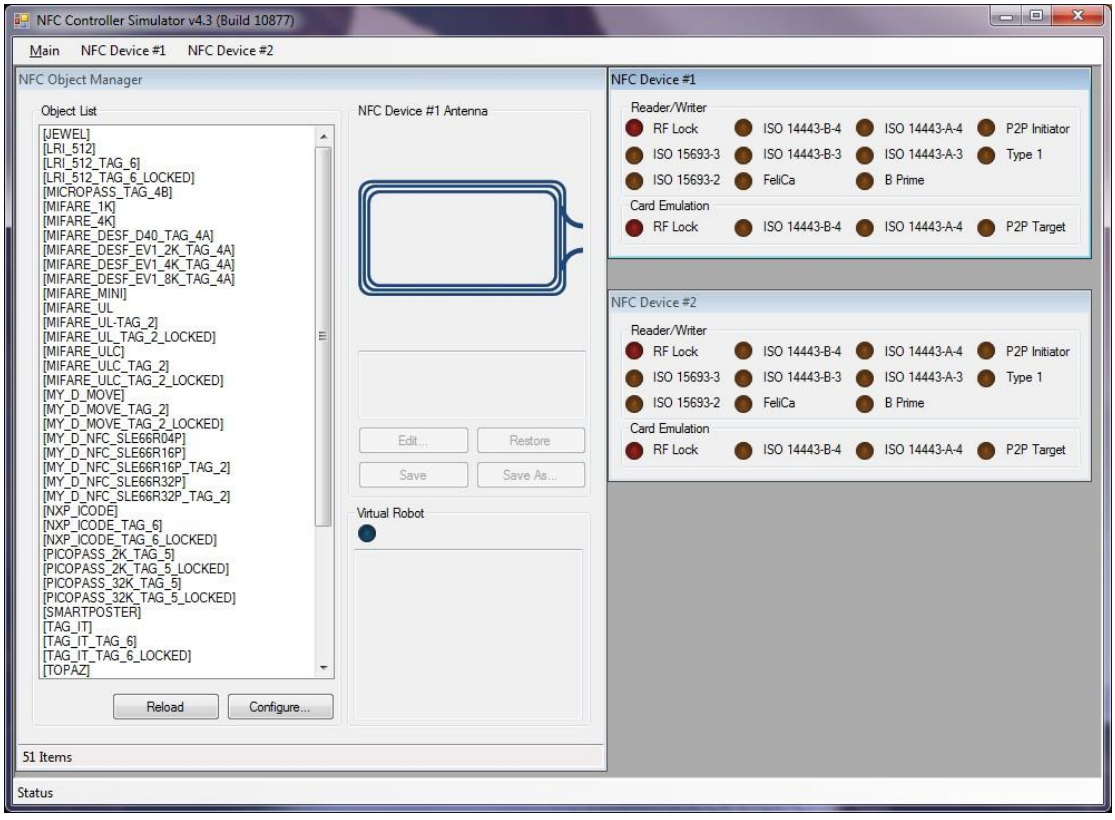

Figura 58 - I vari tipi di tag disponibili su OpenNFC

4.1.3.3 – Il Server

Il server è una implementazione piuttosto semplificata di un server HL7, vengono sfruttate tutte le API di comunicazione del protocollo ed anche strumenti (*HL7Server*) messi a disposizione dallo standard. Il protocollo di basso livello utilizzato è TCP/IP su cui vengono poi ridefiniti tutti i meccanismi di comunicazione proprietari dello standard (tra cui il *TransportLayer*).

4.2 – I package com.tesi.crypto.*

Questi package contengono tutta la logica applicativa per quanto riguarda l'implementazione dei meccanismi di crittografia dell'applicazione. Le API di sicurezza Android permettono di criptare i file con i vari algoritmi di crittografia simmetrica, ma nel momento in cui si va a decifrare questi file se la chiave è sbagliata non viene lanciata alcuna eccezione e il file viene decodificato in una maniera che lo rende illeggibile (salvando però la copia criptata); per questo abbiamo scelto di salvare l'hash della password su memoria di massa mediante un algoritmo estremamente robusto (SHA-256) per poi andarlo a confrontare con l'hash (che è una sequenza di byte) letta dal tag.

4.2.1 – Il motore

Il package *com.tesi.crypto.engine* contiene due classi:

- HashUtility;
- EngineWithKey;

La classe Hash utility espone tre metodi pubblici statici che implementano la logica di creazione degli hash. I primi due metodi riguardano la creazione di hash tramite l'algoritmo di hashing SHA-256, differenziati l'uno dall'altro dal parametro di input. Il metodo *public static byte[] getSHA(String input)* prende in ingresso una stringa e ci restituisce l'array di byte che corrisponde all'hash effettivo della stringa passata in input, questo metodo viene utilizzato dall'Activity di Setup per creare l'hash che andrò a scrivere nel tag per un successivo confronto con quello salvato su memoria di massa. L'altro metodo

public static byte[] getSHA(byte[] input) viene utilizzato dal motore di criptazione per rendere sicuro il file che è stato creato attraverso la SyncActivity.

La classe EngineWithKey è il vero e proprio motore crittografico, si occupa creare ed istanziare tutti gli oggetti utili alla realizzazione delle chiavi il meccanismo di crittografia, inizializza i cifrari e espone i metodi per criptare e decriptare i file.

I metodi esposti sono:

- *Public EngineWithKey(byte[] sks)*: è costruttore pubblico che a sua volta sfrutta un metodo privato per effettuare il setup vero e proprio;
- *Private void setupEngine()*: è il metodo di setup del motore, si occupa di creare le specifiche per la chiave (*SecretKeySpec*), creare l'hash da salvare su memoria fisica (tramite la classe *HashUtility*) ed inizializzare (a meno di eccezioni) i cifrari di criptazione e decriptazione, inoltre il metodo si occupa di creare un file dove viene scritto (a byte) l'hash che verrà usato per fare i controlli al momento;

```
private void setUpEngine() {
    SecretKeySpec keyspec = new SecretKeySpec(key, "AES");
    byte[] iv = new byte[]{ 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
    IvParameterSpec spec = new IvParameterSpec(iv);
    byte[] hash = HashUtility.getSHA(key);

    try {
        encCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        encCipher.init(Cipher.ENCRYPT_MODE, keyspec,spec);

        decCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        decCipher.init(Cipher.DECRYPT_MODE, keyspec,spec);

        BufferedOutputStream bf = new BufferedOutputStream(new FileOutputStream("HashSec"));
        bf.write(hash);
        bf.close();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Figura 59 - Il codice della setUpEngine

- *Public void encrypt(FileInputStream fileIn)*: Il metodo, come si può anche capire dal nome, si occupa di criptare un file prendendo in ingresso il *FileInputStream* ad esso connesso, per effettuare la crittografia byte per byte (a blocchi), sfruttando gli stream cifrati del package *javax.crypto*;
- *Public File decrypt()*: questo è il metodo che, dualmente al metodo *encrypt*, si occupa di decriptare il file, restituendo a chi lo ha invocato

un riferimento al file decriptato, in maniera tale da poter essere utilizzato e/o connesso ad appropriati stream; nell'applicazione verrà utilizzato per decriptare il file che è stato scritto su memoria fisica dall'Activity di sincronizzazione;

Questa classe è stata progettata come serializzabile, in quanto volevamo che fosse "passabile" tra le attività previste nell'applicazione.

4.2.2 – Il Controller

Il package *com.tesi.crypto.controller* si occupa di effettuare i controlli degli hash per effettuare la validazione del login all'apertura delle applicazioni. Il package consta di una classe:

- DecriptController;

La classe *DecriptController* espone due metodi statici, realizzando così un servizio di delega dei compiti senza dover necessariamente istanziarla. I metodi che espone sono:

- *Public static byte[] readHashFromFile(File file, int l)*: questo metodo è stato scritto per leggere un array di byte dal file che viene creato all'atto dell'istanziazione del motore di crittografia, viene utilizzato principalmente dal metodo di comparazione per ricavarsi l'hash da comparare;
- *Public static boolean compareHashWithKey(File hashFile, byte[] hash)*: il metodo che effettua il controllo tra l'hash preso dal file indicato dal riferimento che viene passato come parametro e l'array di byte, che rappresenta l'altro hash, sempre incluso tra i parametri; questo metodo sfrutta il metodo *readHashFromFile* sopradescritto;

4.2.3 – Gli algoritmi ed i meccanismi utilizzati

Gli algoritmi utilizzati per implementare la crittografia sono tutti implementati nel package *javax.crypto*, è stata operata una scelta su quale utilizzare basata su tre fattori:

1. La robustezza e la resistenza alle collisioni;
2. L'impatto sulle risorse computazionali, di conseguenza quindi anche sull'autonomia;

3. La “semplicità” di utilizzo;

La scelta più lampante che soddisfacesse tutti questi requisiti è ricaduta su un algoritmo a chiave simmetrica; quest’ultimo infatti ha un impatto computazionale non eccessivo e inoltre richiede la generazione di una chiave sola, per quanto riguarda la semplicità di utilizzo è un meccanismo che si presta alla perfezione in quanto la criptazione e la decriptazione vengono fatte utilizzando la medesima chiave. Un discorso leggermente diverso va fatto per la robustezza, il quale ci ha portato ad utilizzare un algoritmo leggermente più pesante a livello computazionale, ma che ci ha permesso di utilizzare una funzione di hash sensibilmente più leggera: AES (*Advanced Encryption Standard*). AES è un algoritmo di cifratura a blocchi che implementa la rete di sostituzione e permutazione, offre ottime prestazioni sia per l’implementazione via hardware che per quella via software e ha un utilizzo piuttosto esiguo di memoria rispetto ai suoi vari concorrenti; questo standard di crittografia supporta versioni a 128, 192 e 256 bit la cui complessità e robustezza, ma anche ingombro di memoria e aumento della richiesta delle risorse computazionali, aumentano con il crescere del numero di bit.

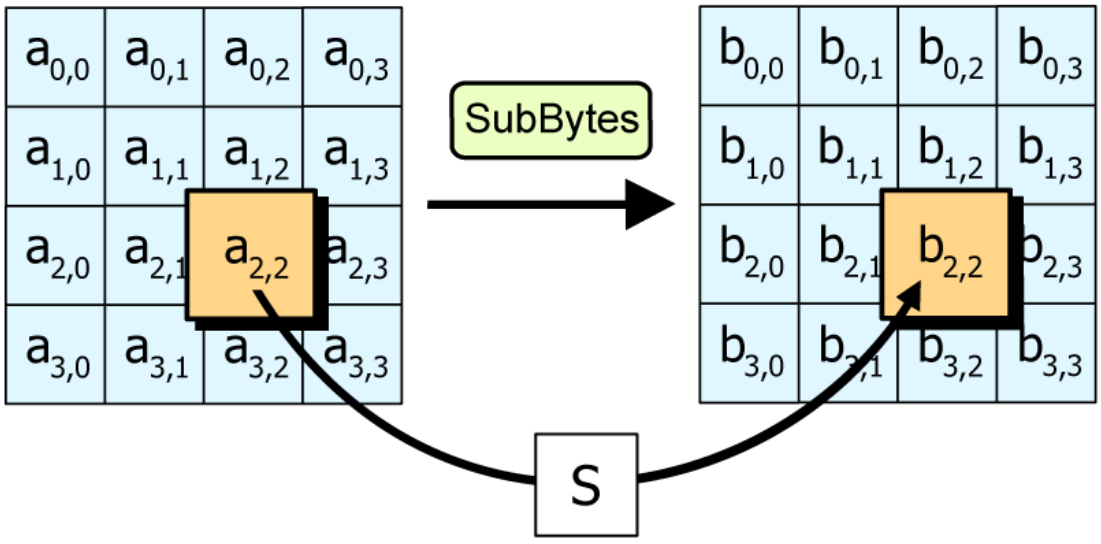

Figura 60 - La rete di sostituzione e permutazione di AES

La versione di AES da noi scelta è a 128 bit, con un algoritmo di hashing molto leggero computazionalmente: MD5 (*Message Digest 5*); quest’ultimo prevede

in input una stringa di lunghezza arbitraria e ne produce, in output, una di lunghezza di 128 bit.

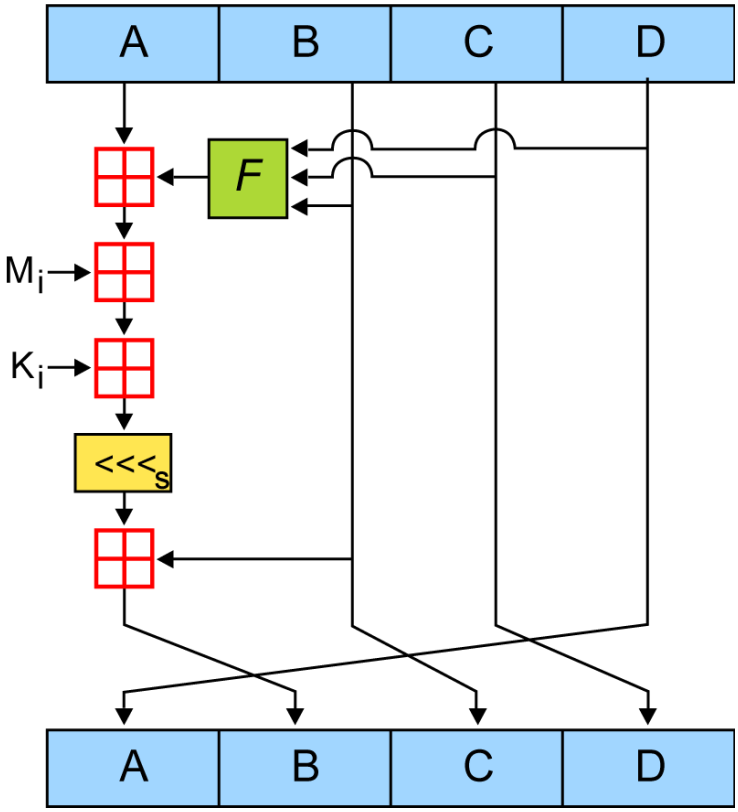

Figura 61 - Lo schema dell'algoritmo MD5

La combinazione dei due algoritmi non è la più sicura a livello di collisioni e robustezza, ma è un ottimo compromesso in quanto garantisce una discreta protezione e un utilizzo piuttosto esiguo di risorse all'atto della creazione delle chiavi.

```
SecretKeySpec keyspec = new SecretKeySpec(key, "AES");
byte[] iv = new byte[]{ 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
IvParameterSpec spec = new IvParameterSpec(iv);
byte[] hash = HashUtility.getSHA(key);

try {
    encCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    encCipher.init(Cipher.ENCRYPT_MODE, keyspec,spec);

    decCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    decCipher.init(Cipher.DECRYPT_MODE, keyspec,spec);
```

Figura 62 - L'istanziazione dei cifrari

La libreria che abbiamo utilizzato non prevede la gestione di eccezioni nel caso in cui la chiave con cui proviamo a decriptare il file non sia esatta, quindi abbiamo ideato un meccanismo di confronto tramite salvataggio di un secondo hash su filesystem e confronto con la chiave che ci verrà passata dall'esterno. Il salvataggio avviene in un file senza estensione, utilizzando un secondo algoritmo di hash (applicato al risultato del primo): SHA-256 (*Secure Hash Algoritm – 256 bit*), più importante a livello di utilizzo di risorse computazionali ma molto più sicuro di MD5. SHA è un algoritmo di hash che produce un *message digest*, come MD5, ma più sicuro e quasi irreversibile; questo algoritmo ci garantisce una robustezza ad attacchi di forza bruta estremamente elevata e di conseguenza ci rende sicuri per un eventuale "furto" del file dal dispositivo. Il meccanismo di utilizzo degli algoritmi di hashing più l'algoritmo a chiave simmetrica AES è riassunto dalla figura 12 qui di seguito.

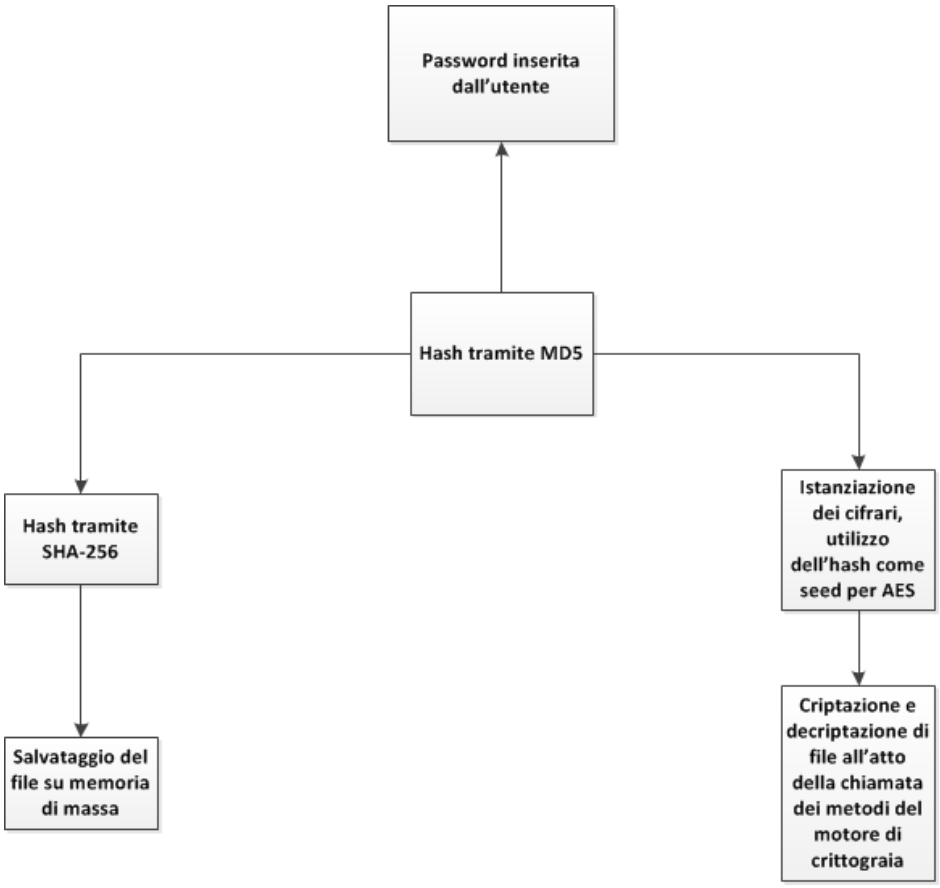

Figura 63 - Lo schema di interazione dei vari algoritmi utilizzati

4.3 – HL7 e il relativo package

Health Level 7 (HL7) è un'associazione non profit internazionale che si occupa di gestire standard per la sanità. HL7 è riferito anche ad alcuni degli specifici standard creati da questa associazione (es. HL7 v2.x). Fondata nel 1987 e riconosciuta dall'American National Standards Institute nel 1994, riunisce organizzazioni affiliate da oltre 40 paesi. Lo standard che noi andiamo ad utilizzare in questo progetto è appunto HL7v2, uno standard rappresentato da un progetto chiamato HAPI (*HL7 Advanced Programming Interface*) che è completamente open source.

4.3.1 – Il modello di funzionamento del protocollo

Il protocollo ha un modello di funzionamento estremamente integrato con le applicazioni mediche, le quali si scambiano messaggi tramite un linguaggio di markup dalla grande capacità espressiva quali XML. Il supporto allo scambio di dati con questo meccanismo può essere molto costoso e complesso, in quanto ci sono delle grammatiche da rispettare (il che implica impiego di risorse computazionali per la validazione dei file, da entrambi i lati) e dei file (a volte anche di grandi dimensioni) da inviare per poi utilizzarne una minima parte; il costo dell'invio dei file diventa abbastanza trascurabile quando si è in reti LAN e WAN in quanto si hanno velocità di trasmissione che vanno dai 54Mb/s a salire, ma quando c'è il bisogno di scambiare questi dati con l'esterno che il fattore velocità (e disponibilità di banda) diventa critico. HL7 ha minimizzato questo utilizzo, creando una sintassi personalizzata per lo scambio dei messaggi e un supporto alla trasmissione ed al parsing di questi ultimi estremamente efficiente.

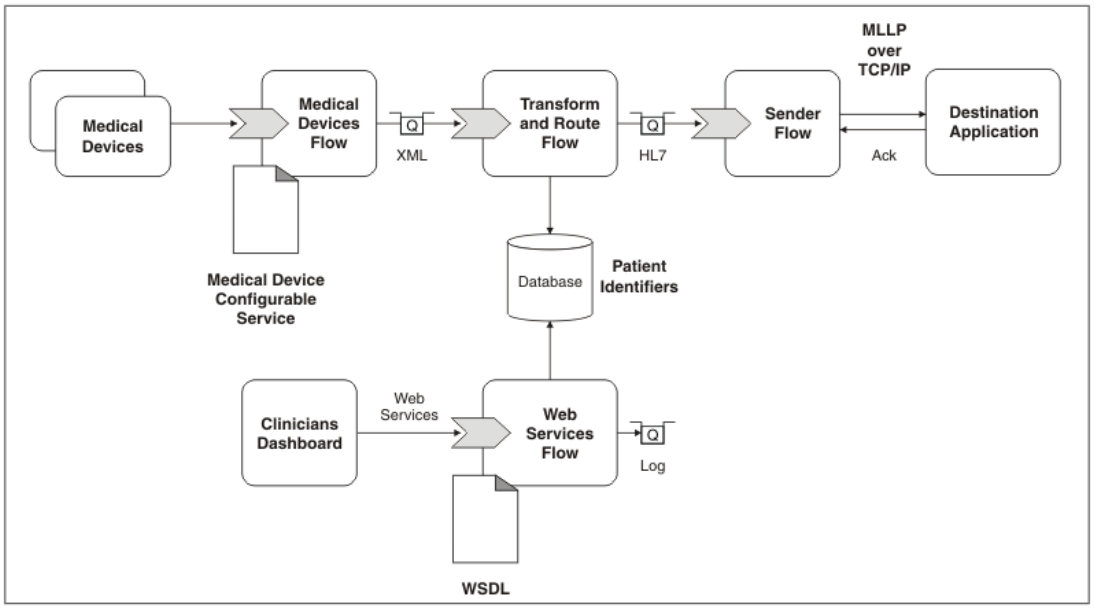

Figura 64 - Il workflow di una normale applicazione medica con l'utilizzo di HL7

La figura 13 ci mostra chiaramente come HL7 intervenga solo nella parte finale per l'interazione con l'applicazione remota, ma prima tutto il messaggio da comunicare deve passare attraverso il *Transform and Route Flow* che al suo interno avrà tutte le funzionalità di parsing e conversione necessarie; possiamo vedere che oltre a comportarsi da connettore (che potrà essere implementato con il pattern DAO, oppure avere un comportamento full-ORM) con il database, dovrà prevedere di utilizzare le API di parsing ed encoding previste dal protocollo. La versione 2 di HL7 prevede delle API estremamente accurate per incapsulare tutta la logica di comunicazione tra il server e l'applicazione remota, funzionalità di parsing e un modello di threading per la parte server molto simile al modello standard dei server Java TCP/IP. Questa versione definisce anche un protocollo proprietario chiamato MLLP (*Minimal Lower Layer Protocol*). MLLP è un protocollo di trasporto dei messaggi molto affidabile. Garantisce la consegna ordinata e con sematica *at-least-once* di contenuti che rispettano lo standard HL7. Il contenuto è incapsulato in un Blocco e inviato alla destinazione. Il sistema di destinazione conferma la ricezione del messaggio, restituendo un messaggio di conferma Ack. Il protocollo MLLP è sincrono: il sistema di origine non trasmette nuovi contenuti finché non viene ricevuto un ack per il precedente invio.

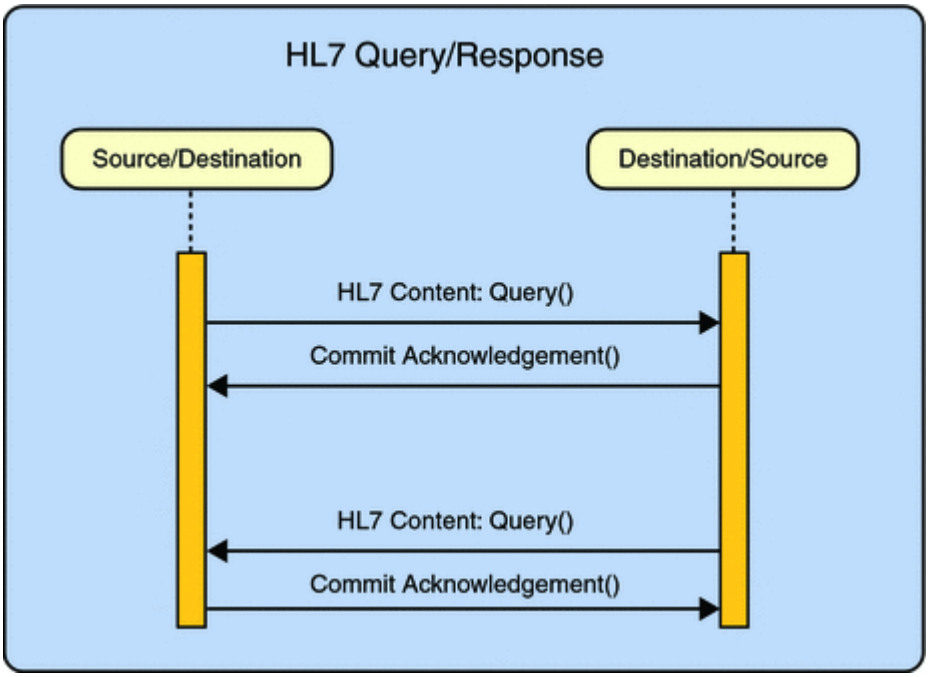

Figura 65 - Il modello di comunicazione di MLLP

4.3.2 – Le API di comunicazione

HL7v2 è caratterizzato da una serie di interfacce estremamente vasta per incapsulare i vari tipi di comunicazione con i server, inoltre alcune di queste interfacce presentano delle implementazioni per dei modelli di comunicazione semplice (server HL7 che utilizza MLLP su TCP/IP tramite funzioni standard), oppure comunicazione complessa (API per l'accordamento di messaggi in topic e code di messaggi gestite da JMS). Il package che contiene tutte le implementazioni sopra elencato è: *ca.uhn.hl7v2.protocol.impl*, come possiamo capire dal nome è il package che contiene tutte le implementazioni delle interfacce esposte nel package "di livello superiore".

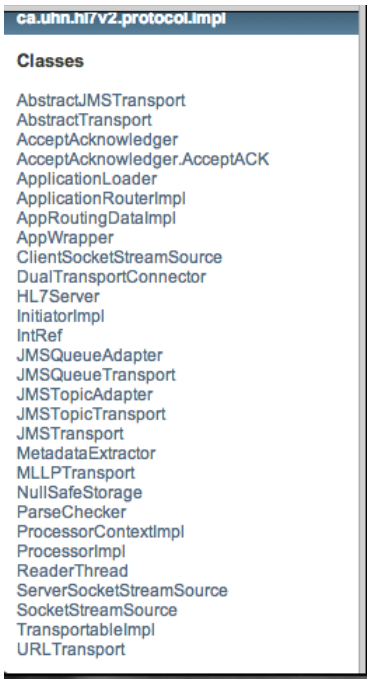

Figura 66 - Le classi presenti nel package

Le funzionalità da noi utilizzate riguardano MLLP su TCP/IP, queste API sono estremamente funzionali ed il loro modello di utilizzo prevede l'incapsulamento e l'automazione dei meccanismi per la comunicazione TCP/IP. Il modello di esecuzione prevede inoltre il passaggio dei parametri tra i vari sottolivelli del modello in maniera del tutto automatizzata.

4.3.2.1 – Il Server

L'implementazione del server è stata molto semplice, proprio grazie ad una implementazione ed un incapsulamento dei meccanismi di comunicazione di basso livello molto ben progettati. Le API ci hanno fornito la Classe *HL7Server* che una volta istanziata rappresenta un vero e proprio server HL7 con politiche di threading, gestione delle richieste e interfacciamenti con i sottolivelli di trasporto già implementati; i parametri per istanziare il server sono:

- *ServerSocket:* una ServerSocket standard dal package *java.net.** utilizzata per comunicazioni TCP/IP e UDP/IP;

- *ApplicationRouter*: è una classe che si occupa di fare il routing dei *Message* ricevuti all'applicazione a cui viene collegato (*bind*), il costruttore si occupa autonomamente di effettuare il bind a se stesso;
- *SafeStorage*: il SafeStorage è un'astrazione che rappresenta uno storage sicuro dei *Message* che vengono ricevuti;

Il server in se ha una politica di threading per quanto riguarda la gestione delle richieste in arrivo che può variare tra due modalità:

- Un thread separato per ogni richiesta;
- Gestione di ogni richiesta nello stesso thread in cui viene accettata;

Questo è possibile tramite l'utilizzo di due metodi differenti: il metodo *accept()* ed il metodo *start()*. Nel primo caso abbiamo un thread unico che si occupa di accettare e rispondere alle richieste, nel secondo il server si mette in un ciclo di attesa attiva e istanzia un nuovo thread ad ogni richiesta che arriva.

```
public class SimpleHL7Server {

    public static void main(String[] args) throws Exception {

        ServerSocket servsock = new ServerSocket();
        servsock.bind(new InetSocketAddress("localhost", 8080));
        HL7Server theServer = new HL7Server(servsock, new ApplicationRouterImpl(), new NullSafeStorage());

        BufferedReader bf = new BufferedReader(new FileReader("resources/ADT_A01.txt"));
        String message = "", line = "";

        while ((line = bf.readLine()) != null)
            message += line;

        bf.close();

        while(true){

            Processor proc = theServer.accept(null);
            Map<String, Object> metadata = new HashMap<String, Object>();
            metadata.put("version", "2.4");
            proc.send(new TransportableImpl(message,metadata), 1, 100);

        }

    }

}
```

Figura 67 - Il codice sorgente del nostro Server

La politica scelta per il nostro server corrisponde al modello a singolo thread per tutte le richieste, quindi viene ad essere un server sequenziale creato per rispondere ad un'unica richiesta per volta; abbiamo effettuato una scelta di questo genere, poiché era la modalità di implementazione più semplice. Inoltre

abbiamo scelto di leggere il messaggio da un file di testo per semplificare al massimo le operazioni che vengono effettuate dal server, questo messaggio preso come stringa viene incapsulato in un oggetto di tipo *Transportable* ed inviato dal server all'atto della connessione. Il costruttore di *Transportable* prevede anche un parametro di tipo *Map<String,Objetc>* che sta ad indicare eventuali metadati da inserire che una volta deserializzati (automaticamente dal *TransportLayer*, oggetto a sua volta presente nelle API ma istanziato automaticamente da qualsiasi oggetto di più alto livello nella pila) sono disponibili al programmatore identificandoli con una stringa come chiave. L'invio viene gestito da un oggetto di tipo *Processor* che si occupa di ricevere i dati, oppure inviare risposte; quest'oggetto in realtà è un interfaccia che viene poi implementata dalla classe *ProcessorImpl*.

4.3.2.2 – Il Client

Il client da noi implementato è stato incapsulato nell'applicazione che abbiamo progettato. Il compito del client è di ricevere un oggetto di tipo *Message* dal server, codificarlo come file XML e salvarlo (criptato) su memoria di massa; successivamente, una volta effettuato il login, questo file deve venir decriptato e il contenuto renderizzato in maniera che sia leggibile dall'utente. Il meccanismo suddetto ci ha portato ad implementare due diverse Activity con due compiti distinti:

- *SyncActivity*: si occupa di gestire la comunicazione con il Server HL7, lo scambio di *Message* e di salvare (criptandolo) il file su memoria di massa;
- *DTRenderActivity*: si occupa di leggere il file, istanziare il tipo di dato ad essa associato e generare dinamicamente delle TextView che mostrano il contenuto dell'oggetto che è appena stato istanziato;

La *SyncActivity* utilizza tutte le API di comunicazione messe a disposizione dal package *com.uhn.hl7v2.protocol.impl* per incapsulare una comunicazione TCP/IP. In particolare si è scelto di definire una classe chiamata *SimpleHL7Receiver* come una implementazione dell'interfaccia *ReceivingApplication*; questa interfaccia mappa il comportamento di

un'applicazione orientata a ricevere messaggi, pertanto richiede l'implementazione di due metodi:

- *Public boolean canProcess(Message message)*: che restituisce un valore booleano in base a delle caratteristiche del messaggio definite da noi in fase di progettazione del *ReceivingApplication*;
- *Public Message processMessage(Message theMessage)*: questo metodo definire il processing vero e proprio del messaggio ricevuto, sia esso un ACK oppure un messaggio vero e proprio codificato secondo lo standard; dualmente il parametro di ritorno di questo metodo sarà un ACK nel caso in cui abbiamo ricevuto un messaggio codificato, oppure un Messaggio codificato nel caso in cui abbiamo ricevuto un ACK e prevediamo di voler inviare una risposta.

La *ReceivingApplication* nel modello di esecuzione complessivo utilizza, similarmente alla parte Server, un *ApplicationRouter* sul quale dovrà registrarsi (*bind*) e che provvederà al dispatching dei messaggi ricevuti. In questo caso però non avremmo l'appoggio di un oggetto che mappi il comportamento di un client, quindi abbiamo scelto di ricorrere alla classe *Connection* presente nel medesimo package riguardante l'implementazione del protocollo; *Connection* mappa una connessione TCP/IP ad un server definito come istanza della classe *HL7Server*, per fare questo richiede il riferimento ad un *LowerLayerProtocol* (noi abbiamo scelto di utilizzare eMLLP, *ExtendedMinLowerLayerProtocol*), una *Socket* (Socket standard del package *java.net.**) e l'istanza di un *Parser*. Risalendo la pila verso un livello di astrazione più alto abbiamo poi deciso di affidarci ad un *Receiver* per l'effettiva ricezione dei messaggi, il quale utilizza la *Connection* precedentemente creata ed avvia (tramite il metodo *start()*) il Thread di ricezione delle risposte.

La *DTRenderActivity* si occupa semplicemente di leggere il file XML decriptato utilizzando la classe *DefaultXMLParser* nel package *com.uhn.hl7v2.parserImpl* istanziando così un *Message* e successivamente utilizza questo per istanziare un oggetto di tipo *DT* di versione 2.4.

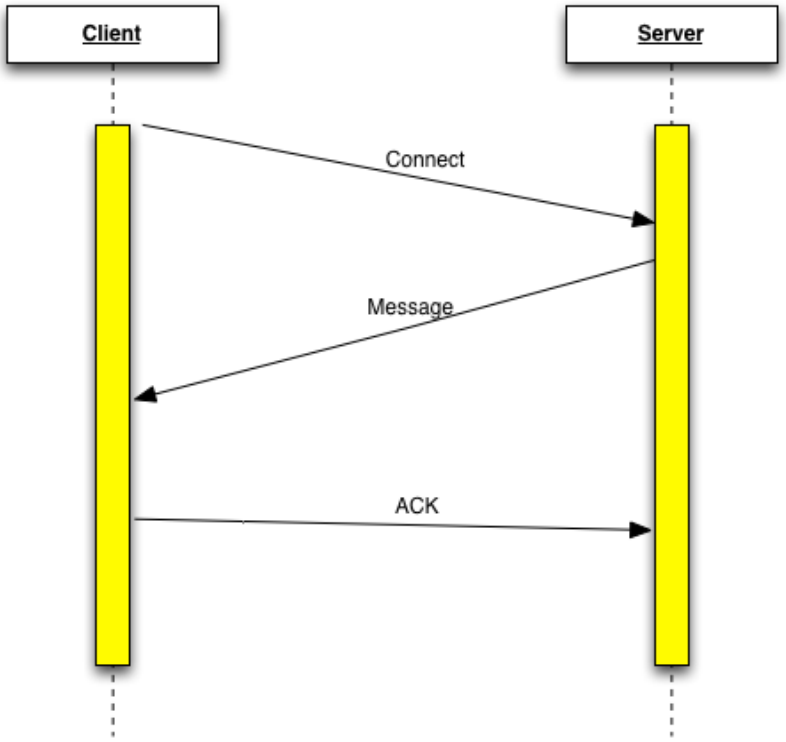

Figura 68 - Il modello di interazione tra Client e Server

4.4 – L'organizzazione dell'applicazione, il package *com.tesi.droidcare*

Android, come abbiamo già detto, è un sistema con un livello di complessità elevato; questa peculiarità permette agli sviluppatori di avere ampia libertà di manovra a livello progettuale. La complessità è dovuta ad un modello di threading e gestione delle notifiche estremamente concorrenziale, il quale consente una gestione molto efficiente delle risorse computazionali.

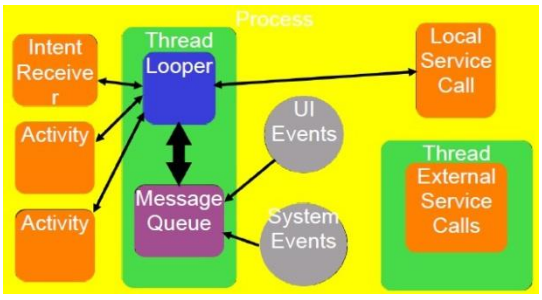

Figura 69 - Il modello di threading

Android ha una gestione dei thread diversa rispetto alla normale gestione a cui si è abituati in ambito desktop, l'ambiente nativo (basato su kernel linux) prevede una gestione della JVM diversa dallo standard. La gestione delle risorse avviene in questo modo:

1. Quando l'utente avvia un'applicazione, viene creata un'istanza della JVM che viene associata solo ed unicamente a quell'applicazione;
2. Ogni istanza della JVM solo un thread all'avvio, che corrisponderà all'Activity che dovrà venir eseguita al lancio dell'applicazione (specificata nel manifest dell'applicazione);
3. Ogni Activity dell'applicazione corrisponde ad un thread nella JVM, non vengono messi in esecuzione più Activity contemporaneamente, ma possono essere eseguiti più thread se vengono definiti dei task;
4. Le Activity vengono messe in pausa e riattivate in base alla posizione nello stack, che viene gestito con la politica FIFO (*First In – First Out*).

Basandoci su questo modello è possibile progettare l'applicazione con un modello di esecuzione molto efficiente, è possibile, infatti, creare un modello di esecuzione in base alle Activity che sono create e trasferire il controllo come meglio crede. La gestione dello stack FIFO ci garantisce che, anche in caso di altre Activity lanciate dall'esterno, il controllo torna alla nostra applicazione una volta terminate.

Il modello di esecuzione della nostra applicazione fa uso di cinque *Activity*, ognuna delle quali viene lanciata dall'Activity principale in base a dei valori letti dalle *SharedPreferences*; ognuna di queste *Activity* ha un compito diverso e specifico in base alla preferenza condivisa a cui si fa riferimento. Le Activity che abbiamo definito sono:

- *MainActivity*: l'attività principale, si occupa di istanziare il motore di crittografia e in base ai valori delle *SharedPreferences* si occupa di fare dispatching del controllo alle altre attività;
- *SetupActivity*: gestisce tutta la parte di setup del tag NFC in base alla password inserita dall'utente nella TextView definita nell'interfaccia grafica;

- *LoginActivity*: gestisce l'interazione tra lo smartphone ed il tag NFC letto, si occupa del confronto tra il valore letto e quello memorizzato su memoria di massa ed in caso sia andato tutto bene, restituisce il controllo alla *MainActivity*;
- *SyncActivity*: come già detto precedentemente è l'attività che incapsula tutta la gestione della comunicazione con tramite il protocollo HL7 per il download di informazioni cliniche più aggiornate;
- *DTRenderActivity*: attività il cui compito è mostrare a video e presentare i dati scaricati dal server in una forma leggibile e comprensibile all'utente finale.

Queste ci ha portato ad elaborare il modello di esecuzione rappresentato dallo schema seguente.

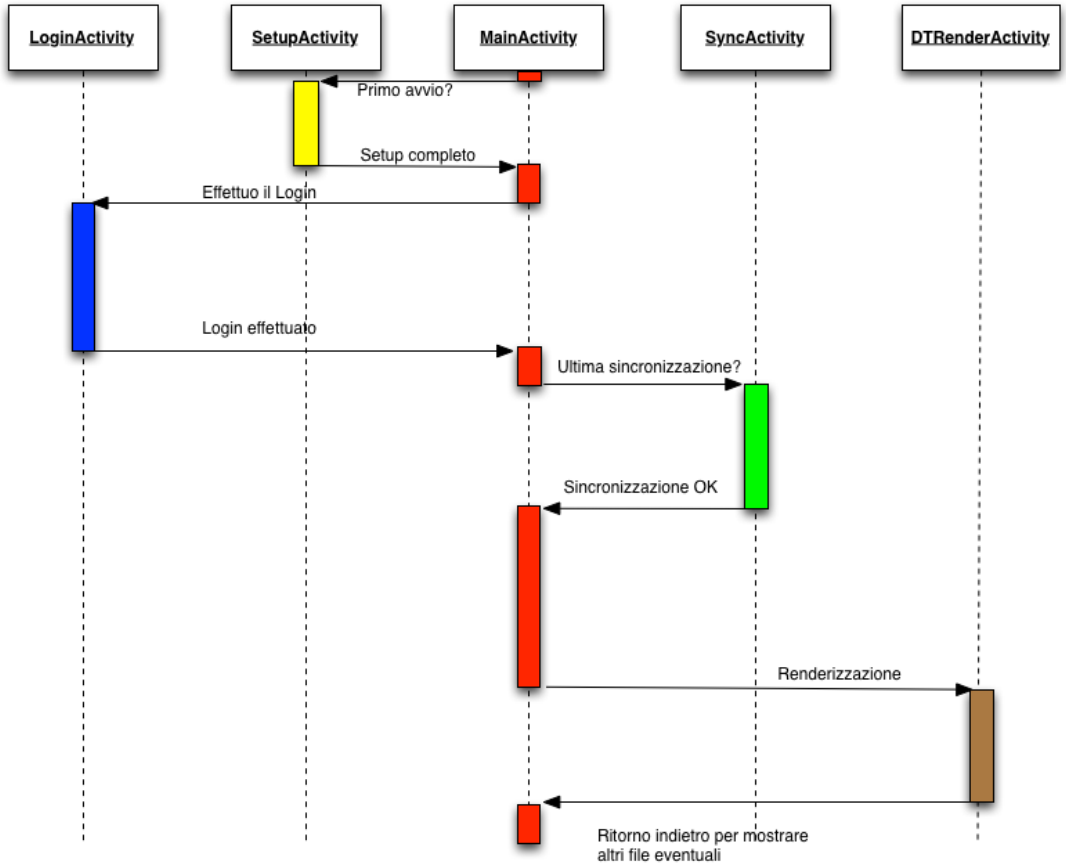

Figura 70 - Il Workflow della nostra applicazione

4.4.1 – MainActivity

La *MainActivity* ha il ruolo di controller e dispatcher. Controller perché in base ad un modello rappresentato dalle *Shared Preferences* seleziona quale view mostrare all'utente, dichiarando *Intent* espliciti per lanciare le Activity. Il ruolo di dispatcher è a livello di interazione con l'utente, in quanto definisce in base al modello quale dovrà essere l'intervento specifico dell'utente per proseguire con il workflow.

La *MainActivity* utilizza, come detto in precedenza, le *SharedPreferences* per determinare a quale Activity lanciare e trasferirne il controllo.

```
//Trovo il riferimento alle preferenze condivise e prendo la prefernza booleana del login
SharedPreferences sp = getSharedPreferences(prefsFile, Context.MODE_PRIVATE);
boolean logged = sp.getBoolean(login, false);
String configured = sp.getString(setup, "");
String dateLastUpdate = sp.getString(date, new Date().toLocaleString());
boolean firstDownloaded = sp.getBoolean(firstDownload, false);
```

Figura 71 - La dichiarazione ed il recupero delle preferenze condivise

La figura 20 ci mostra il meccanismo che abbiamo utilizzato per la dichiarazione delle preferenze condivise, i valori che vogliamo recuperare in base all'Activity da lanciare; successivamente verrà fatto un controllo sul valore di ognuna per determinare se l'Activity mappata da questa deve essere o meno lanciata.

Le preferenze che abbiamo dichiarato corrispondono ad ogni Activity che lanceremo (tranne l'Activity di sincronizzazione che ne utilizza due), fatta eccezione per le Activity di rendering dei contenuti; abbiamo quindi quattro preferenze:

- *Boolean logged*: preferenza booleana che indica se l'utente ha già effettuato il login, la chiamata del metodo *onDestroy()* sull'Activity setta a *false* il valore della preferenza;
- *String configured*: questa preferenza viene inizializzata vuota, successivamente vi verrà scritto il valore dell'hash della password trascodificato in stringa e poi la preferenza verrà posta a null all'atto di chiusura dell'applicazione (il controllo viene effettuato sul valore della stringa vuota, che è diverso dal valore *null*);

- *String dateLastUpdate*: rappresenta la data codificata in stringa, per farne un controllo ne viene effettuato il parsing a *Date* e viene fatta la differenza con quella attuale, se la differenza è maggiore di un intero arbitrario (noi abbiamo scelto il valore 5) si effettua la sincronizzazione con il server;
- *Boolean firstDownloaded*: questa preferenza indica se è stato eseguito la prima sincronizzazione, insieme (tecnicamente sono messe in OR logico) alla preferenza *dateLastUpdate* determina se effettuare o meno la sincronizzazione;

```
@Override
protected void onDestroy(){
    super.onDestroy();
    SharedPreferences sp = getSharedPreferences(prefsFile, Context.MODE_PRIVATE);
    SharedPreferences.Editor spe = sp.edit();
    spe.putBoolean(login, false);
    spe.putString(setup, null);
    spe.commit();
    toBePassed.delete();
}
```

Figura 72 – Il metodo onDestroy() che va a settare alcune preferenze condivise

La scelta di utilizzare il meccanismo delle preferenze condivise è stata dettata dall'esigenza di voler utilizzare una risorsa con una struttura *<chiave>:<valore>* con una tipatura forte e persistente su memoria di fisica senza dover andare a ridefinire regole di parsing, sistemi di nomi e persistenza.

4.4.2 - SetupActivity

L'Activity di setup ha il compito, al primo avvio dell'applicazione, di inizializzare un tag con una password inserita dall'utente; questa password sarà prima codificata come un array di *byte* tramite le funzioni di codifica degli hash presenti nel package di crittografia, dopodiché verrà salvata su un tag NFC di tipo Ndef.

Figura 73 - Il WorkFlow dell'Activity di Setup

Le API NFC che utilizza questa Activity sono le funzionalità standard offerte dal package *android.nfc.**, che permette di interagire con tutti i tag Ndef e non standard; offre anche la possibilità, a meno di funzionalità piuttosto complesse, di wrappare il funzionamento con tag che richiedono driver o interfacciamenti proprietari. Tramite queste funzioni è possibile formattare un tag in maniera personalizzata, inserendo campi (*Ndef Record*) di vario tipo; noi abbiamo scelto di utilizzare un record di tipo URI per identificare il tag come leggibile unicamente dalla nostra applicazione, ed un record scritto puramente a byte per identificare l'hash della password.

Figura 74 - La struttura del nostro tag

Una volta terminata la scrittura del tag l'Activity viene finalizzata (*finish()*) e restituirà, prevedibilmente, il controllo all'Activity principale.

4.4.3 - LoginActivity

La prima Activity che è lanciata dall'Activity principale(fatta eccezione per il primo avvio), tutta l'interazione dell'utente con i dati contenuti nell'applicazione è subordinata al successo dell'interazione con questa Activity. Le API utilizzate da quest'Activity sono anch'esse prese dal package *android.nfc.**, più precisamente tutte le funzioni ed i meccanismi orientati alla lettura dei tag (compresa gestione dei nuovi Intent derivati dalla rilevazione di tag NFC). L'Activity controlla in prima istanza se il tag presenta come primo

record l'URI da noi creato per la validazione, successivamente va a leggere la password in byte scritta sul tag e la confronta con quella scritta sulla memoria di massa. La gestione di questo meccanismo è affidato al metodo *onNewIntent()*, il quale viene invocato nel momento in cui viene rilevato un nuovo tag NFC; la logica implementata prevede anche un controllo sul tipo di intento che ne ha scatenato l'invocazione del metodo.

```
@Override
public void onNewIntent(Intent intent){
    if(NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction())){
        Parcelable[] rawMessagge = intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
        if(rawMessagge != null){
            NdefMessage message = (NdefMessage) rawMessagge[0];
            NdefRecord[] extra = message.getRecords();
            passwdHasHash= extra[0].getPayload();

            if(DecriptController.compareHashWithKey(hashFile, passwdHasHash)){
                myTxtView.setText(new String("Login effettuato con successo"));
                SharedPreferences sp = getSharedPreferences(prefsFile, Context.MODE_PRIVATE);
                SharedPreferences.Editor spe = sp.edit();
                spe.putBoolean(login, true);
                spe.commit();
                finish();
            }
            else{
                myTxtView.setText(new String("Gli hash non corrispondono, non e' il tag giusto"));
            }

        }


    }
}
```

Figura 75 - il metodo onNewIntent() che incapsula tutta la business logic di questa Activity

Il metodo fa uso anche del controller per quanto riguarda il package crittografico, in quanto la validazione del login (oltre che quello del tag) va fatto all'interno dell'attività; una volta effettuata la validazione viene settata la preferenza condivisa e restituito il controllo all'Activity principale.

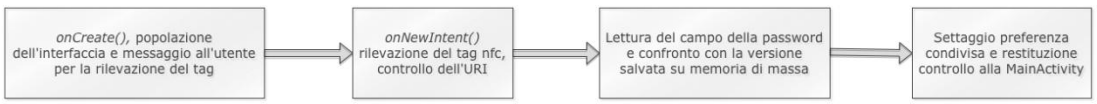

Figura 76 - Il workflow della LoginActivity

4.4.4 - SyncActivity

Una delle due attività che fa uso del protocollo HL7, utilizza tutte le primitive di comunicazione con server HL7, come abbiamo già detto in precedenza, incapsulando tutti i meccanismi di comunicazione di basso livello. L'Activity

riceve come parametro extra dall'Intent l'istanza (serializzata) del motore di crittografia creato dalla MainActivity, questo servirà a criptare il messaggio che verrà scritto su memoria di massa dopo l'avvenuto download. L'Activity dovrà istanziare tutti gli oggetti necessari a completare la pila di comunicazione di HL7.

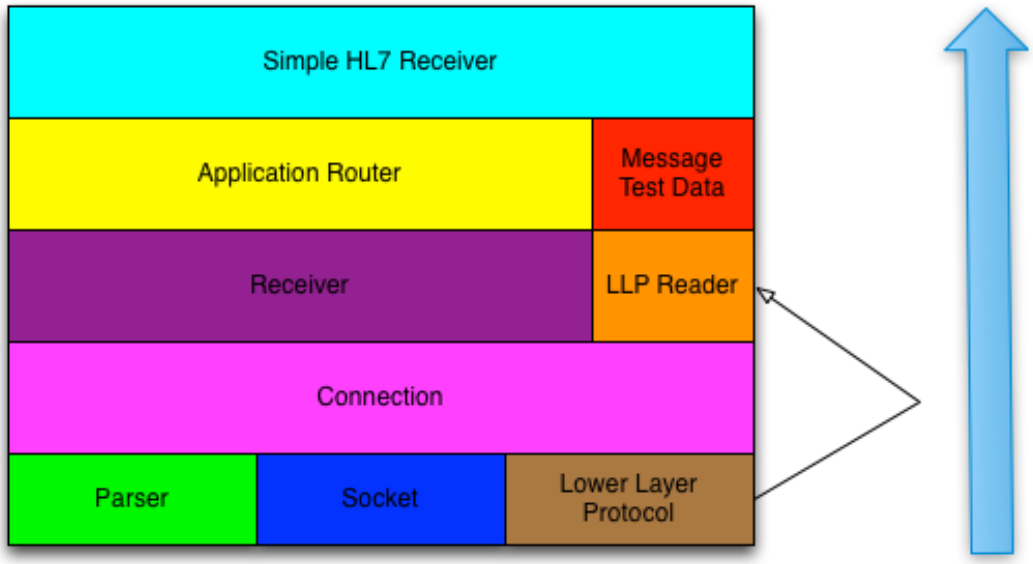

Figura 77 - La pila HL7 per la comunicazione da parte del client

Tutti gli oggetti presenti nella pila (vedi figura 26) vanno istanziati esplicitamente per una corretta ricezione e instradamento del messaggio, dall'istanza del *SimpleHL7Receiver* è possibile ricavare il contenuto del *Message* risultato dalla comunicazione con il server; questo messaggio verrà successivamente scritto su memoria di massa tramite un parser xml che tramite la *encodeDocument(Message message)* codificherà il documento secondo la grammatica xml definita dallo standard XML. Una volta fatto questo il file verrà criptato, grazie all'istanza del motore ricavata all'inizio e verrà cancellato il file XML di partenza.

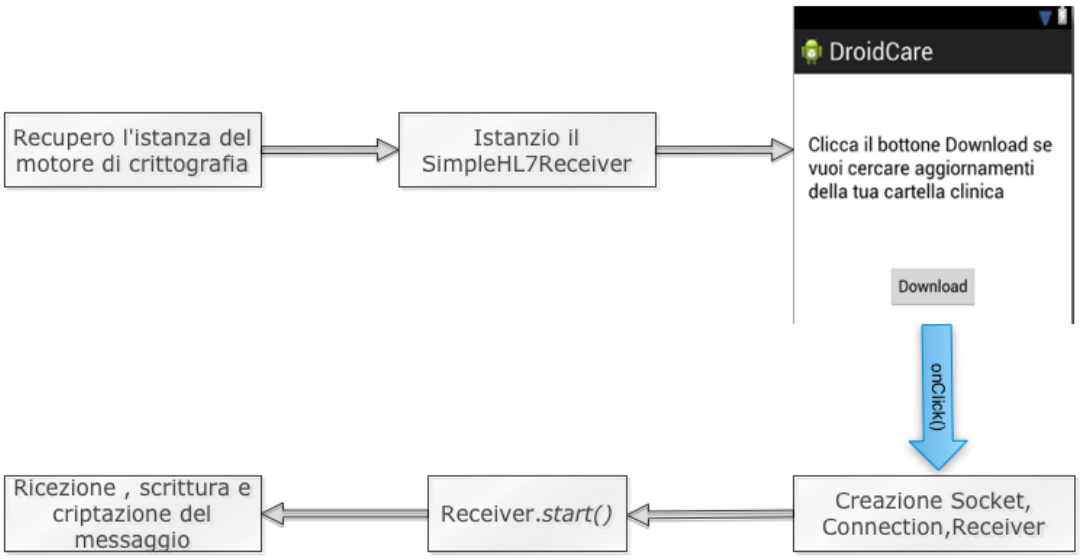

Figura 78 - Il workFlow completo dell'Activity

4.4.5 - DTRenderActivity

La DTRenderActivity si occupa di mostrare a video, tramite generazione dinamica dell'interfaccia grafica, il contenuto del messaggio; questo è possibile istanziando la classe primitiva definita dal messaggio (per capire la versione dell'istanza della classe dobbiamo ricorrere alla *Messagge.getVersion()* che restituirà una stringa con la versione del messaggio, noi abbiamo utilizzato un messaggio di versione 2.4 ed istanziato un oggetto di tipo *DT* del package *ca.uhm.hl7v2.model.v24.datatype*). Una volta istanziato l'oggetto è stata istanziata una TextView per ogni proprietà di questo aggiungendolo alla ScrollView precedentemente inserita in fase di design. E' stato anche inserito un bottone che all'atto del click (*onClick()*) terminerà l'attività restituendo il controllo all'attività principale.

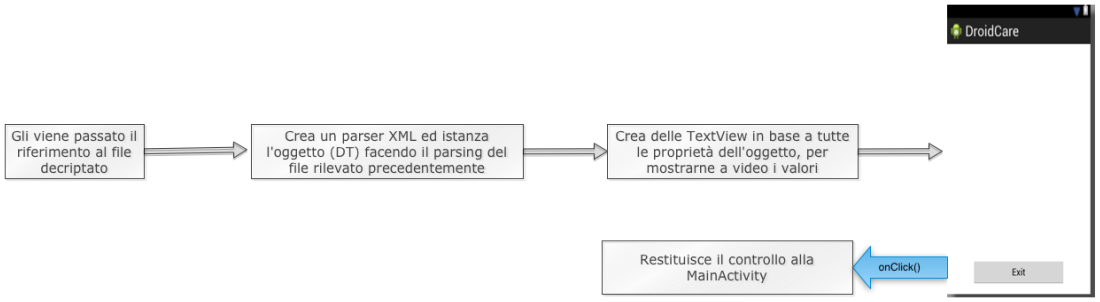

Figura 79 - Il workflow della DTRenderActivity

Questa Activity è stata definita più a livello esemplificativo, l'applicazione può difatti prevedere che venga generata un'Activity di rendering per ogni tipo di dato, sia esso primitivo o complesso; il dato da istanziare andrebbe però ricavato tramite la reflection sulla versione del messaggio e sul tipo di dato da voler mostrare. Le TextView non riportano solo il semplice valore della proprietà dell'oggetto, ma anche il nome della proprietà come "intestazione" della View.

Conclusioni

L'obiettivo principale di questo progetto consiste nello sviluppare un'applicazione per l'accesso remoto a Personal Health Record utilizzando come piattaforma di riferimento Android. L'accesso a questi dati risolve molte problematiche, di carattere burocratico e logistico all'utente finale; l'accesso a questi dati, che sono sempre aggiornati in quanto risiedono su un server remoto su cui vengono caricati i risultati e le cure in tempo reale, permette un'interazione più efficace tra medico e paziente. Le problematiche principali sono legate all'elevato livello di privacy di questi dati, il che ci ha portato ad utilizzare meccanismi di crittografia messi a disposizione dalla piattaforma di riferimento. Oltre alla sicurezza dei dati risiedenti sulla piattaforma, abbiamo affrontato un'altra problematica che riguarda l'accesso a questi dati tramite il software da un utente diverso dal possessore del dispositivo; questa problematica è stata risolta adottando un meccanismo di autenticazione tramite NFC. Un altro aspetto su cui ci siamo soffermati riguarda l'utilizzo di meccanismi per lo scambio di messaggi e modello di dati, abbiamo scelto infatti di utilizzare lo standard HL7.

Cercando di voler soddisfare tutti i requisiti, in fase di sviluppo abbiamo cercato alcuni compromessi per non andare a inficiare sulle prestazioni dell'applicazione. La scelta più importante è stata l'algoritmo di crittografia, il quale ha un peso notevole sulle performance dell'applicazione poiché è il componente che utilizza più risorse computazionali; la decisione è stata semplificata anche dalla documentazione resa disponibile online e dalla disponibilità di librerie che implementano in modo estremamente ottimizzato i vari algoritmi. Un altro compromesso si è presentato riguardo alla scelta della versione dello standard, la scelta era ricaduta sulla terza versione la quale presentava delle evoluzioni molto interessanti; le quali consistevano in una semantica di comunicazione più ricca (messaggi incapsulati in file XML), interoperabilità del protocollo con qualsiasi livello di trasporto da TCP/IP ad http fino ad https e tls. Questa versione però mancava del supporto e della documentazione necessaria per produrre un client in grado di svolgere anche le funzioni più elementari, quindi abbiamo deciso di utilizzare la versione

precedente; questa usufruisce di un supporto a tutt'ora molto attivo e di uno sviluppo continua. La versione da noi scelta, anche se non nativamente, può interoperare con tutti i livelli di trasporto a patto di utilizzare librerie ed API extra. Lo step della realizzazione vera e propria dell'applicazione, utilizzando i moduli creati per le parti suddette, invece non ha presentato alcun problema, questo grazie all'abbondanza di documentazione e supporto reso disponibile da Google e dalla comunità di sviluppatori.

Il software, come già detto, presenta delle possibilità di estensioni che spaziano dall'aumento del livello di sicurezza all'interfacciamento con server di livello enterprise più complessi. L'aumento del livello di sicurezza sarà possibile grazie alla continua evoluzione dei dispositivi mobili, in termini di prestazioni dei processori e più in generale dell'aumento delle risorse; questo consentirà di utilizzare algoritmi sempre più sicuri andando ad impattare sempre meno sull'autonomia. Un altro sviluppo possibile riguarda l'utilizzo della tecnologia NFC, recentemente infatti questa tecnologia sta prendendo piede nel campo dei pagamenti facendo si che il dispositivo mobile si interfacci in maniera sicura con un dispositivo permettendone l'autenticazione (tramite scambio dei dati sensibili, quali il numero della carta di credito) su POS (*Point Of Sale*) abilitati; sfruttare questa modalità di autenticazione sicura permetterebbe di delegare a questo meccanismo tutta la parte di login, inoltre potrebbe essere una modalità di scambio di dati con il proprio medico senza dover passare per altri meccanismi potenzialmente insicuri. Riguardo invece l'estensione dell'utilizzo del protocollo si può implementare l'invio e la ricezione di messaggi come un interfacciamento con un servizio basato su JMS (*Java Message Service*); in questa maniera il software sarebbe integrato maggiormente con l'ecosistema software ospedaliero e più in generale con l'infrastruttura sanitaria.

Bibliografia

P. Bellavista, <<Piattaforme per sviluppo di servizi mobili: panoramica, confronto, J2ME, Android,>> [Online], Avaliable: http://lia.deis.unibo.it/Courses/sm1213-info/lucidi/04-platforms(1x).pdf

B. Elgin, «Google Buys Android for Its Mobile Arsenal,» 2005. [Online]. Available: http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm.

O. H. Alliance, «Industry Leaders Announce Open Platform for Mobile Devices,» 5 Novembre 2007. [Online].Available: http://www.openhandsetalliance.com/press_110507.html.

C. Ziegler, «Android: A visual history,» 7 Dicembre 2011. [Online]. Available: http://www.theverge.com/2011/12/7/2585779/android-history#section_5.

«The WebKit Open Source Project,» [Online]. Available: http://trac.webkit.org/wiki.

M. Carli, ANDROID 3: Guida per lo sviluppatore, APOGEO, 2011.

D. Bornstein, «Dalvik VM Internals,» Maggio 2008. [Online]. Available: https://sites.google.com/site/io/dalvik-vm-internals.

C. Haseman, <<Android Essentials,>> Berkeley: Apress, 2008.

S. Dave e F. Geoff, <<Android recipes: a problem-solution approach,>> Berkeley: Apress, 2011.

F. A. W., S. Robi, K. Chris e A. Frank, <<Android in Action,>> Manning , 2011 .

S. James e T. Nelson, The Android Developer's Cookbook: Building Applications with the Android SDK: Building Applications with the Android SDK, Addison-Wesley, 2010.

Monica Murero, Ronald E. Rice, «The Internet And Health Care: Theory, Research, And Practice.»

G Eysenbach, « JMIR--What is e-health?, » [Online], Avaliable: http://www.jmir.org/2001/2/e20/

Fondazione Bruno Kessler, «E-Health,» [Online], Avaliable: http://ehealth.fbk.eu/en/home

Fondazione Bruno Kessler, «Research,» [Online], Avaliable: http://ehealth.fbk.eu/en/research

S.Price, R.Summers, «CLINICAL KNOWLEDGE MANAGEMENT AND M-HEALTH,» Second Joint EMBSBMES Conference, Ottobre 23-26, 2002

Pindter Medina J, Gonzalez Villarruel J E, Tovar Corona B., <<Proposal for an m-Health System,>> 2009, Electronics, Robotics and Automotive Mechanics Conference

<<Il Fasciolo Sanitario Elettronico,>> [Online], Avaliable: http://support.fascicolo-sanitario.it/cose-il-fascicolo-sanitario-elettronico-fse/

<<Tutela della Privacy,>> [Online], Avaliable: http://support.fascicolo-sanitario.it/cose-il-fascicolo-sanitario-elettronico-fse/tutela-della-privacy/

<<I Documenti Contenuti,>> [Online], Avaliable: http://support.fascicolo-sanitario.it/cose-il-fascicolo-sanitario-elettronico-fse/quali-documenti-contiene/

<<Servizi Sanitari Online,>> [Online], Avaliable: http://support.fascicolo-sanitario.it/servizi-sanitari-on-line/

<<Tecnologie,>> [Online], Avaliable: http://www.progetto-sole.it/pubblica/index/tecnologie

<<Servizi Attivi,>> [Online], Avaliable: http://www.progetto-sole.it/pubblica/index/serviziattivi

Simon Singh, <<Codici & segreti. La storia affascinante dei messaggi cifrati dall'antico Egitto a Internet, >> BUR, 2001, pp. 407
Caterina Marrone, <<I segni dell'inganno. Semiotica della crittografia, >> Viterbo: Nuovi Equilibri, 2010

<<Crypto++, a free C++ class library of cryptographic schemes,>> [Online], Avaliable: http://www.cryptopp.com

<< Stanford Javascript Crypto Library,>> [Online], Avaliable: http://crypto.stanford.edu/sjcl/

<<javax.crypto,>> [Online], Avaliable: http://docs.oracle.com/javase/7/docs/api/javax/crypto/package-summary.html

<< The Legion of the Bouncy Castle,>> [Online], Avaliable: http://www.bouncycastle.org/java.html

Mohsen Toorani, Ali Asghar Beheshti Shirazi, << LPKI – A Lightweight Public Key Infrastructure for the Mobile Environments,>> 2008, IEEE Paper

<<Spongy Castle by rtyley,>> [Online], Avaliable: http://rtyley.github.io/spongycastle/

<<ULTRA Android Library,>> [Online], Avaliable: http://www.netlawsrl.com/prodotti-e-servizi-5/ultra-android-library

Jung Ha Paik, Seog Chung SEO, Yungyu Kim, HwanJin Lee, Hyun-Chul Jung, Dong Hoon Lee, << An Efficient Implementation of Block Cipher in Android Platform,>> 2011, Fifth FTRA International Conference on Multimedia and Ubiquitous Engineering

<<Google Play Store,>> [Online], Avaliable: http://play.google.com

<<Android SDK and ADT Plugin,>> [Online], Avaliable: http://developer.android.com/sdk/index.html

<<The Open NFC Project, The Reference Open Source NFC software stack,>> [Online], Avaliable: http://open-nfc.org/wp/

Nicolas Courtois, Josef Pieprzyk, <<Cryptanalysis of Block Ciphers with Overdefined Systems of Equations,>>

Joan Daemen, Vincent Rijmen, <<The Design of Rijndael: AES - The Advanced Encryption Standard,>> 2002, Springer-Verlag

Xiaoyun Wang and Dengguo Feng and Xuejia Lai and Hongbo Yu, <<Cryptology ePrint Archive: Report 2004/199,>> 17 Aug 2004

Florent Chabaud, Antoine Joux, <<Differential Collisions in SHA-0,>> 1998, CRYPTO

<<Working with TCP/IP HL7>>, [Online], Avaliable: http://docs.oracle.com/cd/E19509-01/820-5508/ghadt/index.html

<<HL7v2 API Overview>>, [Online], Avaliable: http://hl7api.sourceforge.net/base/apidocs/index.html

IBM, <<IBM Websphere Message Broker>>, [Online], Avaliable: http://publib.boulder.ibm.com/infocenter/wmbhelp/v8r0m0/index.jsp?topic=%2Fcom.ibm.healthcare.doc%2Fha00010.htm

Ringraziamenti

Sono sempre stato dell'idea che un'azione valga più di mille parole, ma in questo caso non è così; quindi, giunto ormai alla fine di questo primo gradino del mio percorso universitario, è giusto che scriva due parole per ringraziare chi mi supportato, sopportato e mi abbia, in qualsiasi modo, aiutato a crescere.

Ringrazio la mia famiglia innanzitutto, che mi ha sempre supportato in questi anni; mamma e papà che mi hanno aiutato a superare i momenti più difficili, mia sorella che mi è sempre stata vicino a cui voglio un gran bene (che poi non glielo dimostro è un altro discorso…).

Ringrazio tutti i miei amici da sempre, con cui ho condiviso quasi tutte le esperienze di questi anni; con i quali, soprattutto nell'ultimo periodo, anche solo scambiarci due parole mi dava carica. Ringrazio Peppe, il mio fedele coinquilino, per tutto il sostegno che mi ha sempre dato. Ringrazio Daniele, che è sempre stato un amico fedele. Ringrazio Francesco che è sempre stato presente nel momento del bisogno. Ringrazio le donzelle Silvietta, Flavia ed Eleonora per tutto quello che hanno fatto. Ringrazio anche Jacopo, che mi sopporta da una vita (e penso che già questo sia tanto); ringrazio anche Domenico che, anche lui, ormai mi sopporta da tempo immemore. Ringrazio anche tutti gli altri ragazzi e ragazze di Pescara Giovanni, Federico, Carlo, Stefano, Silvano, Fabrizio, Augusto che ognuno a suo modo mi hanno supportato ed accompagnato in questo cammino.

Ringrazio anche Marco, Donato, Pierpaolo e Giulio che ho avuto la fortuna di conoscere e si sono dimostrati dei grandi amici; ringrazio anche il bomber (aka Domenico) per tutto, molestie incluse. Ringrazio anche il gruppo Borges che nell'ultimo periodo mi ha sopportato alla grande. Ringrazio anche tutti gli altri ragazzi che ho avuto la fortuna di conoscere ed apprezzare, come Tommy, Paolo, Gigio. Ringrazio anche Sabbia per il suo essere diretto, schietto ed essersi dimostrato sempre sincero, Adelina, Lorena, Lucilla e Sorina che mi hanno sostenuto ed aiutato durante questo lungo periodo di studio; ringrazio anche Miki per tutti gli esami studiati e passati insieme. Un grazie va anche a Manu per tutto l'aiuto che mi ha dato. Ringrazio anche Fabio (aka Sucre),

Andrea (aka Smile) che mi hanno aiutato a trascorrere più serenamente questo periodo.
Ringrazio anche Eleonora che ho conosciuto da poco tempo ed ho imparato ad apprezzare da subito e si è dimostrata/o una grandissima/o amica/o; ringrazio anche Stefania per essersi dimostrata un'ottima amica. Ringrazio anche i ragazzi che ho iniziato a conoscere da poco, ma con cui mi sono trovato da subito in sintonia e di cui ho tanta stima: Mattia, Mirko, Carmen, Francesco, Francesca. Un grazie va anche a Marika che si è dimostrata una gran persona.
Ringrazio il mio relatore il Prof. Ing. Paolo Bellavista, per avermi dato la possibilità di svolgere questo progetto; confermando la mia stima nei suoi confronti a livello personale e accademico.
Ringrazio anche chi per un motivo o per un altro è passato nella mia vita ed è andato via, lasciandomi comunque un ricordo; perché sia esso bello o brutto mi ha aiutato a crescere.