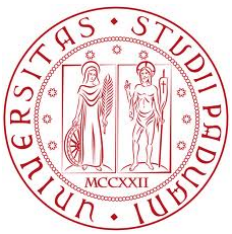

UNIVERSITÀ DEGLI STUDI DI PADOVA
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

SISTEMA DI AUTENTICAZIONE A RICONOSCIMENTO BIOMETRICO SUPPORTATO DA TELEFONO

Laureando: *Marco Nunes 580201-IF*

Relatore: *Ch.mo Michele Moro*

22 Febbraio 2011

Anno Accademico 2010 / 2011

Sommario

Questo progetto ha intenzione di portare il suo contributo nel campo dei dispositivi mobili sviluppando un'applicazione per smartphone in cui sia presente il sistema operativo *Android*. L'applicazione si interfaccerà con un sistema a riconoscimento biometrico e farà uso del nuovo paradigma ad *agenti mobili*, questo perché il classico paradigma client-server non riesce più a stare al passo con l'espansione di Internet e quindi non soddisfa al meglio i requisiti che le moderne applicazioni operanti in rete richiedono.

Il contesto operativo è quello dell'autenticazione dell'individuo tramite impronte digitali dove il dispositivo fungerà da repository di dati, non solo per quanto riguarda il template biometrico ma anche per i dati identificativi del suo possessore e per le sue chiavi pubbliche e private.

Gli strumenti utilizzati saranno ovviamente uno smartphone di ultima generazione, uno scanner di impronte e due postazioni che simuleranno un server contenente la banca dati e un terminale che farà iniziare l'autenticazione.

I risultati ottenuti sono certamente d'aiuto per lo sviluppo futuro di sistemi di sicurezza sempre più affidabili, dove il controllo degli accessi a particolari luoghi è prioritario.

Ringraziamenti

Volevo ringraziare in primo luogo il mio relatore, il Prof. Michele Moro, per l'aiuto concessomi durante la stesura di questo mio lavoro di tesi.

Un ringraziamento particolare lo devo ovviamente ai miei genitori, Luis e Margherita, e a mia sorella Sara, perché il raggiungimento dei miei risultati è solamente merito dei loro grandi sforzi e sacrifici.

Volevo poi ringraziare la mia fidanzata Erika che mi ha sempre sostenuto ed incoraggiato, soprattutto nei momenti di difficoltà.

Infine un pensiero va a tutti gli amici e a tutti i pendolari, che sono stati giorno per giorno i fedeli compagni di questo percorso.

INDICE

ELENCO DELLE FIGURE

CAPITOLO 1

INTRODUZIONE

In questi primi 10 anni del terzo millennio sta iniziando una seconda grande rivoluzione, dopo la diffusione di Internet, legata all'utilizzo di dispositivi mobili. Quello che inizialmente era un telefono cellulare ora è uno strumento in grado di fornire servizi di vario genere. È possibile conoscere in ogni momento la propria posizione, inviare e ricevere mail, acquisire immagini e filmati, ecc...

Questi dispositivi stanno diventando veri e propri PC portatili in cui la funzione di telefono, sebbene fondamentale, è solo una di tante funzioni e l'ampia richiesta che stanno avendo questi telefoni intelligenti fa sì che per gli sviluppatori si stia aprendo un nuovo orizzonte: quello della creazione e dello sviluppo di applicazioni che sfruttino le caratteristiche di questi dispositivi.

L'ampia diffusione di tecnologie informatiche ha portato anche ad un incremento dell'attività di ricerca nel campo dei sistemi distribuiti. Questo perché i classici modelli client-server, che sono stati cardine fondamentale di questa branca dell'informatica, non riescono più a rispondere in modo soddisfacente alle problematiche che si possono incontrare in tali ambiti, e hanno iniziato a mostrare i propri limiti riguardanti scalabilità, la tolleranza ai guasti, la sicurezza e la flessibilità che le nuove applicazioni di rete richiedono.

Una risposta a questo problema emersa negli ultimi anni è stata l'adozione di un nuovo paradigma di programmazione, detto ad *agenti mobili*, che prende spunto proprio dalle due parole che lo compongono. In sintesi si tratta di entità software autonome, gli agenti appunto, che hanno la capacità di spostarsi da un calcolatore all'altro per eseguire diversi tipi di compiti usufruendo delle risorse disponibili nel calcolatore in cui si trovano.

Questo nuovo paradigma porta con sé vari vantaggi tra cui:

- *efficienza*: si può spedire un processo direttamente al client in modo che vi operi localmente senza interagire continuamente con il server;
- *flessibilità*: le applicazioni che si trovano nel server vengono scaricate e installate localmente nei client quando ne necessitano;
- *occupazione di banda*: scaricare il codice quando necessario fa risparmiare in maniera ragionevole l'occupazione di banda della rete;

Un ruolo fondamentale ai fini di questa tesi lo avrà il telefono perché c'è da sottolineare il fatto che un altro elemento molto importante a vantaggio degli agenti è la possibilità di eseguire i loro compiti anche all'interno di questi dispositivi mobili, dotati di risorse generalmente limitate, sia per quanto riguarda la disponibilità di memoria sia per la potenza di calcolo.

L'integrazione quindi, tra l'utente finale, ovvero il reale possessore del telefonino, ed il sistema centrale, risulta di facile attuazione ed è per questo che nasce l'idea di un sistema di autenticazione che, unito alle conoscenze nel campo del matching biometrico, può essere il trampolino di lancio per lo sviluppo di applicazioni successive.

1.1 Obiettivi della tesi

L'obiettivo di questa tesi è quello di creare un sistema di autenticazione biometrico in cui il template dell'utente è salvato sul telefonino ed il processo di matching biometrico avviene sul server.

Le caratteristiche principali di questo sistema sono l'utilizzo di un'infrastruttura ad agenti mobili, di un parametro biometrico per effettuare l'autenticazione dell'utente e l'uso di uno smartphone di ultima generazione con sistema operativo Android in cui verrà installata l'apposita applicazione.

1.2 Prerequisiti per la lettura del testo

Non sono necessarie particolari conoscenze preliminari per una comprensione efficace di questa tesi, né per quanto riguarda il campo informatico, né per quanto riguarda il campo della biometria.

Si presuppone, tuttavia, che il lettore abbia una certa dimestichezza con le nozioni di base dell'informatica. In particolare di:

- buona conoscenza del linguaggio di programmazione Java[1] in quanto è il linguaggio con cui è stata scritta l'applicazione;

- alcune nozioni tipiche dei sistemi distribuiti: paradigma client-server, crittografia, ecc. . . ;

1.3 Prerequisiti per l'utilizzo del software

Per poter testare ed utilizzare in modo completo tutto il software sviluppato è necessario disporre di:

- sistema Linux distribuzione Ubuntu 10.04[2];
- Java Development Kit 1.6[3] o superiore;
- libreria JAI - Java Advanced Imaging[4];
- server MySQL 5.05 o superiore[5];
- JADE v4.0.1[6] con l'aggiunta di JADE-Leap e dell'add-on JADE-Android v1.2;
- scanner biometrico modello Biometrika Fx2000[7] comprensivo di driver v2.0 o superiore;
- smartphone con sistema operativo Android v2.1[8];

Per la descrizione dei passi necessari al funzionamento dei programmi si rimanda il lettore interessato al capitolo 6.

CAPITOLO 2

PARAMETRI DI AUTENTICAZIONE BIOMETRICA

In questo capitolo si esporranno i concetti basilari della biometria, elencando i parametri principali con particolare attenzione alle impronte digitali, in quanto il sistema proposto si basa sul loro confronto. Si spiegheranno i metodi e gli strumenti per la loro acquisizione descrivendo anche come avviene tale processo.

2.1 Introduzione alla biometria

Con il termine *sistema di autenticazione* si intende l'azione di associare un identità ad un determinato individuo. Esistono tre metodi[9] di identificazione personale e sono:

- *Knowledge-based*: identifica la persona grazie a qualcosa che conosce e che altri non possono e non dovrebbero conoscere come ad esempio un codice segreto o una password;
- *Token-based*: identifica la persona grazie a un oggetto in suo possesso e non facilmente clonabile come ad esempio la tessera del codice fiscale o il badge universitario;
- *Biometric-based*: identifica la persona grazie alle sue caratteristiche personali che sono in larga parte uniche in ogni persona e quindi consentono un riconoscimento univoco dell'identità;

Ciascuno di questi approcci presenta vantaggi e svantaggi di cui bisogna essere consapevoli in fase di progettazione del sistema di autenticazione. Per esempio una

password è facilmente trattabile in ambito informatico con il rischio, però, che può essere dimenticata. Come pure un badge può essere smarrito o peggio rubato. Una caratteristica personale invece è adatta a identificare una persona perché non può essere persa o trafugata, non può cambiare nel tempo o essere dimenticata, quindi rappresenta decisamente il modo migliore per distinguere una persona autorizzata da un impostore.

La biometria è il settore che ha come oggetto di studio proprio la misura di queste variabili personali tipiche degli esseri viventi e l'unico mezzo per poter identificare univocamente un individuo, abbiamo detto, è quello derivante dall'acquisizione di tali caratteristiche. Per poter essere usate con successo, una caratteristica biometrica deve rispettare una serie di requisiti quali:

- *Invariabilità*: devono essere costanti nel tempo;
- *Misurabilità*: devono essere rilevate velocemente;
- *Singolarità*: devono essere uniche e tali da permettere la distinzione di una persona dalle altre;
- *Accettabilità*: devono essere accettate da una vasta percentuale della popolazione;
- *Riducibilità*: devono essere ridotte al fine di una facile gestione;
- *Affidabilità*: la loro acquisizione deve garantire un elevato grado di affidabilità;
- *Privacy*: devono mantenere inviolata la privacy dell'individuo;

La conformazione dell'apparato scheletrico e le dimensioni del cranio, in termini di proporzione fra le sue componenti, sono state le prime variabili studiate nella storia della biometria. Ai giorni nostri invece le tecniche per l'identificazione biometrica riguardano la valutazione di due grandi categorie di parametri: quelli *fisiologici*, riguardanti il corpo, e quelli *comportamentali*, riguardanti, appunto, il comportamento.

Per quanto riguarda la prima categoria le più famose sono:

- *l'immagine del volto*;
- *il disegno dell'iride*;
- *la sagoma della mano*;
- *l'impronta digitale*;

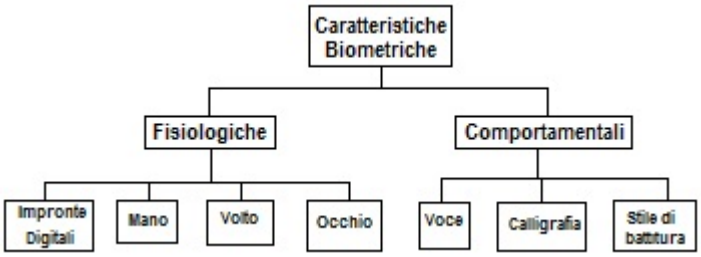

Figura 2.1: Classificazione delle caratteristiche biometriche

invece per la seconda categoria le più diffuse sono:

- *la voce*;
- *la calligrafia*;
- *lo stile di battitura*;

Figura 2.2: Esempi di parametri biometrici

La scelta del parametro biometrico da utilizzare, tenendo conto dei requisiti da rispettare e dell'obiettivo di questa tesi, è caduta sull'*impronta digitale* che, oltretutto, è la caratteristica maggiormente utilizzata oggigiorno per i metodi di autenticazione nei più svariati ambiti.

2.2 Impronte digitali

Un'impronta digitale è la riproduzione dell'epidermide del polpastrello di una delle dita della mano, ottenuta premendo il dito contro una superficie levigata.

Le caratteristiche strutturali più evidenti sono le *creste*, delle linee in rilievo, e le *valli*, gli spazi tra quelle linee, che scorrono in flussi paralleli tra di loro e che saltuariamente si intersecano o si interrompono determinando così i noti disegni dell'impronta digitale che noi tutti conosciamo.

A livello globale, lo schema di creste-valli esibisce una o più regioni caratterizzate da una forma particolare che vengono definite *regioni singolari* e la loro presenza determina la classificazione dell'intera impronta in una delle cinque classi[10], che sono:

- *Right Loop*: impronte in cui una o più creste entrano dal lato destro, si ripiegano, e escono dallo stesso lato;
- *Left Loop*: come le precedenti, ma piegate dal lato opposto;
- *Arch*: impronte in cui le creste entrano da un lato, crescono verso il centro e scendono per poi uscire dal lato opposto;
- *Tented Arch*: impronte che hanno lo stesso andamento di quelle precedente, ma in cui le creste formano un angolo o una piega al centro;
- *Whorl*: impronta a figura chiusa con forma circolare, ellittica o a spirale;

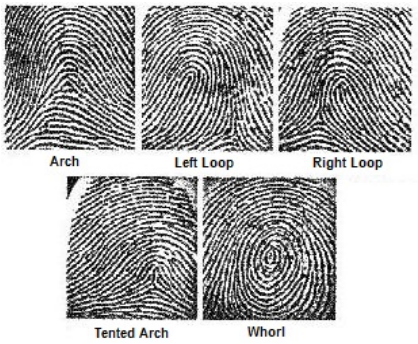

Figura 2.3: Esempi di singolarità

A livello locale invece, le discontinuità delle creste vengono chiamate *minuzie*: esse possono essere fatte corrispondere semplicemente alle terminazioni o alle biforcazioni delle creste, anche se in dettaglio la loro forma può essere descritta in modo più preciso: esistono minuzie a forma di punto o isola, lago, biforcazione e speroni, incroci. Sono proprio queste caratteristiche che vengono utilizzate per i confronti tra impronte e consistono in coincidenze spaziali di un certo numero di minuzie, variabili da 10 a 20: questo confronto è l'unica prova valida per l'identificazione di un individuo, questo perché è basata sulle basilari premesse

di invariabilità, secondo la quale le caratteristiche delle impronte non cambiano attraverso il tempo, e singolarità, la quale afferma che l'impronta è unica da individuo a individuo.

Figura 2.4: I 7 tipi più comuni di minuzie

Un sistema biometrico che faccia uso di impronte digitali è detto anche AFIS (*Automatic Fingerprint Identification System*), ma per essere tale richiede che siano presenti un sensore di impronte digitali (fingerprint reader), un template database ed una unità di calcolo.

2.2.1 Metodi di acquisizione delle impronte

L'acquisizione di un impronta digitale può essere effettuata *offline* e si parla di *inked fingerprint*, oppure *online* e si parla di *live scan fingerprint*:

- *inked fingerprint*: viene definita così l'immagine dell'impronta digitale ottenuta tramite l'impressione del dito su un mezzo intermedio. Generalmente si ottengono passando la superficie del dito su di un tampone contenente inchiostro e premendolo su della carta. Oppure esiste un tipo particolare di impronta offline, che è quella latente: usualmente, sulla superficie del dito, è presente uno strato di sudore e grasso che può lasciare un'impressione dell'impronta digitale sugli oggetti con un semplice tocco. Con opportune tecniche è possibile rilevare tali impronte e ricavarne un'immagine cartacea;

- *live scan fingerprint*: viene definita così l'immagine dell'impronta acquistata direttamente, senza l'uso di un mezzo intermedio. In commercio esistono una serie di sensori in grado di operare un'acquisizione online in tempi molto brevi e a costi molto contenuti;

2.2.2 Strumenti per l'acquisizione delle impronte

I tipi di sensori presenti in commercio sono in grado di acquisire informazioni relative all'impronte digitali e di porle in una forma adatta a essere rappresentate da un calcolatore.

Questi sensori possono essere di tipo:

- *ottico*: focalizzano l'immagine su un sensore di visione sfruttando la legge di riflessione per produrre immagini ben contrastate;
- *capacitivo*: piastrine di silicio contenente una matrice di piccoli sensori che leggono le informazioni sulla corrispondente porzione di impronta;
- *ultrasonico*: impiegano onde sonore che colpiscono la superficie del dito e dall'eco estrapolano le informazioni sulle creste e sulle valli;
- *termico*: sfruttano il differenziale termico tra creste e valli, sono il tipo meno diffuso;

Nella nostro progetto faremo uso di sensori di impronte digitali del primo tipo, in particolare dello scanner ottico su interfaccia USB FX2000 by Biometrika.

Figura 2.5: Scanner FX2000 by Biometrika

2.3 Processo di matching

Il processo di confronto tra due impronte digitali, come avviene per il resto dei parametri biometrici, si basa su quattro fasi fondamentali, ovvero acquisizione del parametro, estrazione di un modello, costituzione di un template e comparazione di quest'ultimo con un'altro dello stesso tipo con valutazione del risultato ottenuto.

Il processo inizia, quindi, con l'*acquisizione dell'impronta*, una fase attraverso la quale si rileva, con le metodologie esposte nei paragrafi precedenti, l'immagine dell'impronta dell'individuo da autenticare.

Si passa poi all'estrazione di un modello, che in questo caso si particolarizza nell'*estrazione delle singolarità*. In questa fase si ricercano, all'interno dell'immagine le minuzie base che sono costituite da biforcazioni e terminazioni. Questa fase influisce pesantemente sul risultato finale, in quanto maggiore è la precisione nell'estrazione delle minuzie maggiore sarà la qualità del template finale. In primo luogo si applicano algoritmi per la riduzione del rumore e per un miglioramento della sua qualità. Poi si procede a selezionare la parte centrale e a binarizzarla: si parte dall'immagine in scala di grigi e si decide se i punti di quest'ultima appartengono o no a una cresta. Questa decisione si trasforma in una mappa di bianco e nero. Infine si passa all'estrazione vera e propria, in cui viene processata l'immagine binarizzata e si identificano i pattern di pixel che corrispondono alla biforcazione o alla fine delle creste, ovvero alle minuzie base.

La terza fase del processo consiste nella *costituzione del template*: in questa fase viene costruito un modello (il template) che contiene le informazioni necessarie all'algoritmo di matching, e che simmetricamente elimini quelle non rilevanti, generalmente al fine di limitare la dimensione complessiva del modello.

L'ultima fase è il *matching* vero e proprio, qui si sfrutta il lavoro dei passi precedenti per determinare se l'autenticazione è da considerarsi eseguita con successo oppure no. Questo avviene confrontando i template delle due impronte, ricercando la presenza di minuzie e di una loro possibile sovrapposizione. Qui entra in gioco la scelta della soglia di discriminazione: è infatti necessario stabilire un numero minimo di minuzie corrispondenti tra le due impronte, che permetta di affermare che quest'ultime provengano dalla stessa persona.

CAPITOLO 3

AGENTI MOBILI E LA PIATTAFORMA JADE

In questo capitolo si introdurrà il paradigma ad agenti mobili, spiegandone i concetti base, elencando le sue principali caratteristiche e approfondendone due molto importanti per il nostro progetto quali la sicurezza e la mobilità. Poi si darà una visione generale della piattaforma JADE che ci fornisce le basi essenziali per lo sviluppo di questa tesi.

3.1 Agenti mobili: una breve descrizione

Il concetto di *agente mobile* è relativamente recente quindi non esiste una definizione universalmente accettata anche se sta assumendo un significato preciso solo in questi ultimi anni. Si tratta di un'entità che ha la capacità di muoversi tra le varie macchine di una rete, che lo rende appunto mobile, e di conservare il suo stato, presupposto essenziale affinché esso sia un agente mobile, perché quando l'agente si sposta, il processo e le sue risorse si spostano assieme lui. In particolare possiamo associarlo al termine processo, e non ad un programma, attenendosi alla terminologia tipica dei sistemi operativi dove per processo si intende un programma in esecuzione.

Questa entità è dotata di un certo grado di autonomia, intesa come capacità di prendere decisioni in modo indipendente e sulla base delle condizioni ambientali circostanti. Queste ultime possono includere ad esempio: informazioni ricevute o inviate verso altri agenti, output ricevuti da altri processi in esecuzione concorrente, risultati di query effettuate su un database, ecc...

Le principali caratteristiche[11] che contraddistinguono un agente sono:

- *autonomia*: possibilità di prendere decisioni autonome in base alle sollecitazioni ricevute dall'ambiente esterno senza l'intervento diretto di umani o altri soggetti, avendo sempre il controllo sia delle sue azioni sia del suo stato interno;

- *reattività*: percepisce il proprio ambiente e risponde in modo tempestivo ai cambiamenti che si verificano;

- *pro-attività*: non solo reagisce in risposta al proprio ambiente ma è in grado di esibire comportamenti orientati all'obiettivo di sua spontanea iniziativa;

- *abilità sociale*: abilità dell'agente di comunicare e scambiare informazioni, attraverso una serie di linguaggi riconosciuti e condivisi, con altri agenti o con l'utente definite interazione agente-agente e agente-utente;

- *persistenza*: in quanto software intelligente, un agente è in grado di mantenere un proprio stato interno che garantisca una continuità logica nel lavoro di esecuzione dell'agente, anche in presenza di migrazioni o interruzioni momentanee dell'ambiente di lavoro;

- *mobilità*: possibilità di un agente di migrare da un host all'altro all'interno di una rete mantenendo integre tutte le proprietà;

- *adattività*: si adatta al proprio ambiente ed ai desideri dell'utente;

- *fidatezza*: fornisce la sicurezza che non comunicherà deliberatamente false informazioni;

- *benevolenza*: prova sempre ad eseguire ciò che gli viene richiesto;

- *cooperazione*: coopera con altri umani o altri agenti al fine di svolgere il proprio compito;

È da specificare che un agente che possiede le prime sei caratteristiche è detto *debole*, altrimenti se le possiede tutte è detto *forte*.

L'utilizzo reale di tutte le precedenti proprietà sarebbe vanificato se ad un agente non fosse associata anche un'ulteriore caratteristica in grado di garantirne l'affidabilità nell'esecuzione e la protezione dei dati in esso contenuti. Va quindi aggiunto un ulteriore aspetto forse considerabile, a ragione, più importante: la *sicurezza*.

3.2 La sicurezza

Il software realizzato per questo lavoro si basa su un sistema sicuro ad agenti mobili. Risulta quindi significativo soffermarsi a discutere dell'aspetto della sicurezza in quanto questo nuovo paradigma possiede un carattere fortemente distribuito, bisognoso di adottare tutte le contromisure necessarie per garantire la sicurezza e l'integrità del sistema dove gli agenti operano con informazioni strettamente riservate come ad esempio dati anagrafici, password, ecc...

In particolare, quando di parla di sicurezza riferita ad agenti mobili, si intende l'attuazione di metodi di prevenzione verso quelle che sono le principali e conosciute tipologie d'attacco attualmente applicabili. Tra queste assumono particolare rilevanza le seguenti minacce:

- *Agent Killer*: ha lo scopo di forzare la terminazione di un agente prima che questo abbia portato a termine il proprio lavoro;

- *Sniffing*: ovvero l'intercettazione del contenuto dei messaggi scambiati tra due entità con lo scopo di acquisire o modificarne il contenuto;

- *Denial of Service (DoS)*: è la tipologia di minaccia più diffusa e consiste nel sovraccarico di un servizio fornito o di una funzione, in modo da bloccarne l'utilizzo da parte di altre entità;

Come si può quindi dedurre dall'elenco sopra riportato, è necessario prevenire e arginare tali tipi di attacchi attraverso politiche di gestione della sicurezza sfruttando tre aspetti principali universalmente riconosciuti in questo ambito ovvero l'*autenticazione* dei componenti della piattaforma, l'*autorizzazione* per effettuare certe operazioni da parte di questi componenti e la *confidenzialità* dei dati presenti e scambiati all'interno della piattaforma.

3.2.1 L'autenticazione

Generalmente, il primo problema da risolvere, riguarda l'individuazione delle entità autorizzate a partecipare ad un determinato sistema, ma nell'ambito degli agenti mobili questo argomento viene interpretato in maniera differente dagli sviluppatori delle piattaforme. In genere, comunque, lo scopo di tale operazione sta nel fatto di associare ogni agente al proprio utente proprietario e responsabile.

JADE sviluppa proprio questo concetto, in quanto pone la condizione che gli utenti, per autenticarsi, devono farsi riconoscere tramite un Id e una password preimpostati per interagire con le entità.

3.2.2 L'autorizzazione

Come diretta conseguenza dell'autenticazione, che nega l'ingresso di entità sconosciute o indesiderate, tutto quello che ha accesso alla piattaforma è noto al sistema e ha l'autorizzazione all'esecuzione. Come seconda cosa, quindi, bisogna rivedere eventuali limiti imposti alla libertà di azione delle singole entità, definendo i confini dentro al quale ognuno è autorizzato ad agire.

La libertà d'azione non è necessariamente uniforme, ma può variare da agente ad agente realizzando meccanismi che riescono a limitare le loro azioni, tramite norme personali e regole prefissate e diversificate: basti pensare a come, all'interno di una rete, gli agenti definiti dall'amministratore del sistema possiedano libertà d'azione e movimento negate alle entità definite dagli altri utenti comuni.

3.2.3 La confidenzialità

Dopo aver garantito, all'interno della piattaforma, la presenza di entità conosciute al sistema, ognuna delle quali con la propria libertà d'azione e i propri ruoli, non rimane altro che stabilire il terzo ed ultimo livello di sicurezza dell'agente: un sistema che garantisca la confidenzialità dei dati.

Questo sistema deve essere in grado di assicurare che, nel caso avvenga un'intercettazione da parte di un'entità ostile, questa non riesca ad interpretare i dati contenuti o che non riesca a manipolarli in qualsiasi modo. Tale scopo viene raggiunto attraverso particolari meccanismi quali:

- *la cifratura*: rende determinati dati non comprensibili ad utenti privi della necessaria autorizzazione tramite algoritmi crittografici che possono essere sia a chiave simmetrica che asimmetrica;
- *l'hash*: consiste nel calcolo di una particolare funzione che ha come input proprio il messaggio stesso e permette di verificare in modo immediato se il messaggio è stato modificato da una terza parte oppure no, il tutto con una ragionevole certezza;
- *la firma digitale*: il suo scopo è quello di certificare l'identità del mittente quindi di evitare il problema del ripudio;

3.3 La mobilità

Una delle caratteristiche di un agente, abbiamo detto, è la sua capacità di muoversi in nodi differenti all'interno del sistema distribuito in cui opera. È subito intuibile, quindi, quante alternative questi agenti possono offrire da un punto di vista architetturale, offrendo una soluzione totalmente diversa rispetto ad un

sistema basato su paradigma client-server. Detto ciò il primo passo per comprendere meglio tale caratteristica e dare una visione più chiara di tale concetto consiste nel spiegare come un software possa muoversi da un calcolatore ad un'altro.

Per programma si intende un oggetto software che può essere caricato nella memoria volatile (RAM) di un calcolatore elettronico ed eseguito in un nuovo processo conferendogli ciò che viene detto stato. Nello stato vengono immagazzinate informazioni sia riguardo i dati utilizzati che gli servono per proseguire nella sua esecuzione, sia riguardo le caratteristiche interne al processo (program counter, stack, instruction pointer, ecc...). Basandosi su questi fatti si possono riscontrare tre diverse tipologie di mobilità:

- *Weak Mobility* (mobilità debole): un agente dotato di mobilità debole effettua la migrazione tra i nodi portando con sé solo il codice che descrive il suo programma;
- *Not-So-Weak Mobility* (mobilità semi forte): un agente se è dotato di mobilità semi forte migra con tutto il suo codice e con una parte del suo stato;
- *Strong Mobility* (mobilità forte): infine, se l'agente ha mobilità forte, durante la fase di migrazione, trasferisce sia il codice del programma sia il suo stato completo;

I diversi gradi di mobilità sono strettamente legati al linguaggio di programmazione che ha permesso di fatto di creare gli agenti stessi. Come abbiamo già accennato il linguaggio con la quale è stato sviluppato il nostro sistema è Java, che permette di implementare una mobilità di tipo *Not-So-Weak*.

3.3.1 Java: il linguaggio ideale per la mobilità

Per eseguire programmi scritti in linguaggio Java ci si affida ad una macchina virtuale, la cosiddetta JVM (*Java Virtual Machine*), che esegue a sua volta file con un'estensione `.class`. Questi file sono composti dal *bytecode*, codice compreso solo dalla JVM, e sono il risultato della compilazione dei sorgenti Java. La forza di questo linguaggio sta proprio in questo: come proclama lo slogan *write once, run everywhere* (scrivi una volta, esegui dappertutto), è possibile scrivere un codice sorgente che possa essere eseguito su un qualsiasi calcolatore sulla quale sia presente una JVM. Aggiungiamo inoltre che la JVM è caratterizzata da quattro punti fondamentali[12]:

- *program counter*: contiene la prossima istruzione del bytecode che verrà eseguita dalla JVM;
- *execution stack*: area di lavoro specifica per ogni thread dove il processo mantiene tutte le informazioni di esecuzione;

- *heap*: è costituito da un'area di memoria comune a tutti i thread e contiene le istanze degli oggetti creati durante l'esecuzione della JVM stessa;
- *method area*: anche quest'ultima è condivisa tra tutti i thread, ha come compito quello di memorizzare le varie strutture associate ad ogni classe, come ad esempio il codice dei metodi;

Avendo spiegato brevemente il funzionamento interno della JVM si può facilmente intuire come si possa procedere per implementare la mobilità: è sufficiente permettere, oltre ad una copia del bytecode, una copia dell'heap su diversi flussi di uscita come ad esempio file, byte stream o socket. Per far ciò si sfrutta la *serializzazione* ovvero quel processo che permette di convertire un oggetto in forma binaria, in modo tale che possa essere trasmesso (ad esempio lungo una connessione di rete) e, all'arrivo, possa essere ricostruito tramite il processo contrario che prende il nome di *deserializzazione*. L'operazione di serializzazione è resa possibile attraverso l'uso dell'oggetto ObjectOutputStream e, in maniera del tutto simmetrica, all'arrivo per mezzo dell'oggetto ObjectInputStream.

3.4 JADE - Java Agent Development Framework

Per la realizzazione di questa tesi si è utilizzato JADE (*Java Agent Development Framework*), un framework sviluppato completamente in Java che supporta lo sviluppo di applicazioni distribuite e basate sul paradigma di programmazione ad agenti mobili. È nato alla fine del 1998 presso i laboratori di ricerca Telecom Italia situati a Torino e poco dopo si decise di rilasciare il software sotto licenza LGPL[1] rendendolo di fatto open-source. Da quel momento una gran comunità di sviluppatori ha contribuito all'espansione di questo sistema partecipando in maniera attiva al suo sviluppo fornendo versioni aggiornate e creando nuovi add-on. Per esempio, tra i numerosi add-on sviluppati per JADE c'è né uno specifico, chiamato JADE-S[2], dedicato al problema della sicurezza affrontato nel sottocapitolo 3.2. La forza di questo meccanismo ha reso JADE un framework estremamente interessante nel panorama dei sistemi distribuiti e uno strumento potente e flessibile agli occhi degli utenti che, grazie alle sue caratteristiche garantisce una forte portabilità sia su ambienti server sia su dispositivi a risorse limitate.

3.4.1 JADE-Leap e JADE-Android

Abbiamo già detto che questa piattaforma estremamente versatile fornisce numerose versioni alla sua comunità di utilizzatori. Tra tutte, JADE-Leap merita

[1] Lesser General Public License
[2] Abbreviazione di JADE-Security

Figura 3.1: Logo della piattaforma JADE

un'attenzione particolare per il nostro progetto, in quanto permette di eseguire JADE su dispositivi a risorse limitate quali palmari o cellulari. Inizialmente, per soddisfare questa esigenza, è stato necessario adattare il framework in quanto, nel progetto originale l'occupazione di memoria del runtime di JADE era troppo elevata, richiedeva una JVM 1.4 o superiore (mentre nella maggioranza dei dispositivi portatili è presente solo KVM[3]) e perché i collegamenti wireless hanno caratteristiche diverse rispetto alla loro controparte cablata che devono essere prese seriamente in considerazione (come ad esempio alta latenza, scarsa banda, connettività ad intermittenza e assegnamento di IP dinamico).

Recentemente è stato rilasciato un add-on di JADE, chiamato JADE-Android[13], che fornisce il supporto per l'utilizzo di JADE-Leap su sistema operativo Android. L'importanza di questo add-on verrà spiegato con maggiori dettagli nel capitolo 4.

3.4.2 Lo standard FIPA

Oltre al rilascio sotto licenza LGPL, la piattaforma JADE aderisce allo standard FIPA[14] (*Foundation for Intelligent and Physical Agents*), un'organizzazione senza fini di lucro formata nel 1996 e legata alla IEEE Computer Society, che promuove la tecnologia basata sugli agenti e l'interoperabilità dei suoi standard con altre tecnologie. In quel tempo però la maturazione di questa tecnologia software non era ancora ad un livello tale per integrarsi con il mondo commerciale essendo conosciuta solo nell'ambiente accademico. Si decise perciò di produrre uno standard che sarebbe andato a formare la spina dorsale di questo sistema favorendo il suo ingresso nell'industria aumentando così il suo utilizzo in varie applicazioni.

Figura 3.2: Foundation for Intelligent and Physical Agents

[3]una JVM specifica per hardware limitato

In particolare, lo standard FIPA coinvolge i seguenti cinque ambiti:

- *Applications*: è costituito da un insieme di esempi di applicazioni, per ognuna delle quali si propone un insieme minimo di servizi che l'agente dovrebbe essere in grado di fornire;
- *Abstract Architecture*: definisce gli elementi essenziali per la realizzazione di una piattaforma per agenti software, le relazioni tra di essi ed fornisce alcune linee guida per la loro implementazione;
- *Agent Communication*: riguarda un insieme di specifiche, numerosi protocolli e schemi di interazioni per la comunicazione e lo scambio di informazioni tra agenti basato su un linguaggio, elaborato da FIPA, chiamato ACL (*Agent Communication Language*);
- *Agent Management*: è formato da una serie di linee guida per la gestione e il controllo degli agenti e del loro comportamento;
- *Agent Message Transport*: definisce delle modalità per il trasferimento e la rappresentazione delle informazioni attraverso reti con differenti protocolli di trasporto;

CAPITOLO 4

SMARTPHONE E SISTEMA OPERATIVO ANDROID

In questo capitolo si fornirà una panoramica generale degli smartphone di ultima generazione spiegando cosa sono le app. Poi si descriveranno i vari sistemi operativi disponibili sul mercato con attenzione particolare alle due piattaforme principali: Android e iOS. Si procederà spiegando l'architettura di Android e il ruolo della Dalvik Virtual Machine. Infine si spiegherà il ruolo di JADE-Leap all'interno del sistema operativo Android.

4.1 App e sistemi operativi

In italiano potremmo tradurlo con *cellulare intelligente*, dove si abbinano le funzionalità di un telefono cellulare al concetto di mobilità intesa come gestione dei dati personali, che siano essi musica, immagini o documenti di qualsiasi genere. Questi telefoni di alta gamma, i cosiddetti *smartphone*, sono ormai a tutti gli effetti piccoli computer. In quanto tali, la gran parte delle loro funzionalità è legata non tanto alle caratteristiche tecniche, quanto ai programmi che vi si possono far girare: questi programmi sono le cosiddette *applicazioni*, a cui gli appassionati si riferiscono con il termine di *app*. Negli ultimi anni, le quote di mercato che riguardano la telefonia sono quasi del tutto state conquistate proprio da questa nuova tipologia di telefono e dalla possibilità di sviluppare e installare nuove applicazioni.

Un *app* è un programma pensato per funzionare su un cellulare: alcune servono semplicemente per navigare su internet, gestire le mail o consultare mappe ma rispetto ai programmi che usiamo sui pc, queste sono state ripensate per adattarsi alle dimensioni ridotte dello schermo dei telefoni. Le statistiche parlano chiaro,

l'utente medio scarica 9 applicazioni al mese, di cui 2 a pagamento, e le utilizza per 80 minuti al giorno.

Elenchiamo i più importanti sistemi operativi per dispositivi mobili presenti oggi sul mercato:

- *iOS*: sistema operativo sviluppato da Apple per iPhone, usa kernel Mach e Darwin;
- *Android*: sistema operativo sviluppato da Google, sta letteralmente spopolando in giro per il mondo, imponendosi come alternativa di iPhone e Blackberry, è basato su kernel Linux;
- *Blackberry*: prodotto dalla società canadese RIM, rappresenta, oltre che un prodotto davvero superiore, uno status symbol non indifferente;
- *Windows Phone 7*: sistema operativo sviluppato da Microsoft, una novità piuttosto recente;
- *Palm*: sviluppato dalla americana PalmSource e che caratterizza per lo più palmari un pò retrò;
- *Symbian*: sistema operativo sviluppato da Symbian Foundation, installato principalmente su dispositivi Nokia;

Le due piattaforme che per prime hanno colto l'importanza dello sviluppo di nuove applicazioni sono state le prime due della lista, ovvero *iOS* e *Android*, per cui rispettivamente troviamo oggi disponibili 300.000 e 100.000 applicazioni e che sono state già scaricate oltre 5 milioni di volte.

Come accade per i computer, anche le app sono vincolate al sistema operativo su cui girano. Un app che funziona su iOS non funzionerà su Symbian e viceversa. Molte app sono disponibili per due o più sistemi, ma solo chi le sviluppa ha deciso di crearne una versione per ciascuno. Dal punto di vista dell'utente questo significa che cambiare telefono è molto più complicato che in passato perché passare da un sistema operativo all'altro comporta la perdita di tutti i programmi che si erano acquistati per quello vecchio. Per questo motivo è importante scegliere un sistema che, oltre a funzionare bene oggi, dia anche garanzia sul futuro.

4.2 La scelta di Android

Come abbiamo già accennato, le due piattaforme che hanno riscosso più successo nel mercato della telefonia intelligente sono iOS e Android. Gli smartphone equipaggiati con questi due sistemi operativi hanno molti elementi simili, ma fatti

salvi i punti in comune in cui tracciare una differenza sarebbe piuttosto complicato, bisogna concentrarsi sulle differenze che ci hanno portato a scegliere il sistema operativo con cui operare per rendere effettivo il nostro lavoro.

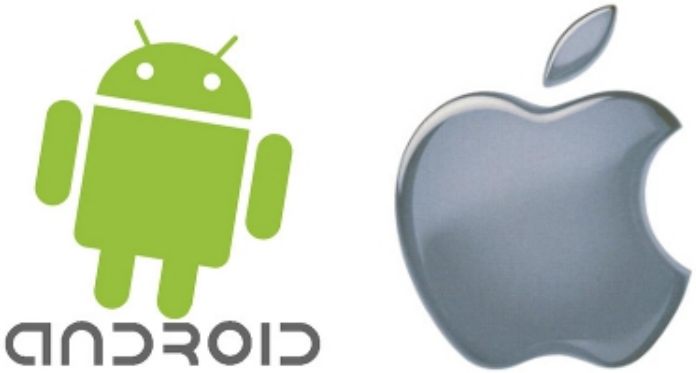

Figura 4.1: Sistemi operativi a confronto

Il dispositivo mobile che useremo per la nostra tesi monta il sistema operativo Android soprattutto perché offre la possibilità di scrivere un'applicazione in piena libertà grazie alla natura open source e perché ha disponibile una SDK dedicata che andremo ad approfondire più avanti.

Elenchiamo, comunque, tutti i punti che si sono rivelati determinanti, chi più chi meno, in questa scelta:

- la nuova release del sistema operativo di Google ha un'efficienza nettamente superiore rispetto ad iPhone e può fare girare le applicazioni in modo più veloce;
- la piattaforma di Google è avvantaggiata dalla possibilità di offrire un accesso al Web molto più simile a quello ottenibile sul computer di quanto avvenga per l'iPhone;
- Android introduce una gestione migliorata degli account di posta elettronica e per la sincronizzazione di rubrica, calendario e amministrazione remota;
- la possibilità di utilizzare il telefonino come punto di accesso Wi-Fi alle reti a banda larga estende in modo importante la competitività di Android;

- compatibilità di Android con un elevato numero di smartphone di marche differenti;
- l'ecosistema creato da Google è aperto, in contrapposizione con la piattaforma chiusa e altamente controllata di Apple;
- Google ha semplificato lo sviluppo di applicazioni attraverso un kit che semplifica al massimo la vita ai developer e apre le porte anche a chi vuole provare a scrivere software, al contrario di Apple che ha una politica molto stringente sulla certificazione dei programmi pubblicabili;

4.3 Android: concetti fondamentali

Come sappiamo Google non è solo il nome che si associa al celeberrimo motore di ricerca, ma è una fonte inesauribile di idee. Nonostante la grande crisi economica mondiale che ha suggerito al gigante della rete di accantonare alcuni progetti meno essenziali, in questi ultimi mesi Google ha comunque continuato a sviluppare le sue idee per il mercato difficilissimo che è rappresentato dai dispositivi mobili, in cui concorrenti come Apple, Nokia, Microsoft e molti altri la fanno da padrone.

Sta quindi sempre più maturando Android che, come vedremo meglio, non è un microbrowser e nemmeno una applicazione installabile sul proprio telefono, ma si tratta di qualcosa di più complesso che parte dal sistema operativo fino ad una Virtual Machine per l'esecuzione delle applicazioni mobili.

Caratteristica fondamentale di tutto ciò è l'utilizzo di tecnologie open source a partire dal sistema operativo che è Linux con il Kernel 2.6, fino alla specifica virtual machine per l'esecuzione di applicazioni che si chiama Dalvik. Il tutto è guidato dalla OHA (*Open Handset Alliance*) ossia da un gruppo di 47 aziende (numero in continua crescita), il cui compito è quello di studiare un ambiente evoluto per la realizzazione di applicazioni mobili.

4.3.1 Architettura di Android

Quando si introduce Android si utilizza sempre una famosa immagine che ne descrive l'architettura (Figura 4.2). In questa paragrafo utilizzeremo l'immagine per inquadrare quello che è l'elemento forse più importante per noi sviluppatori ossia la *Dalvik Virtual Machine*. Notiamo come essa faccia parte dell'ambiente di runtime e sia costruita sui servizi offerti dal kernel e dalle diverse librerie native.

Ciò che l'architettura non mette in evidenza è il ruolo di Java in tutto questo. In una architettura Android non esiste alcuna virtual machine Java e non viene eseguito alcun bytecode Java. Parliamo di Java semplicemente perché per sviluppare applicazioni per Android si utilizza proprio questo linguaggio. Non si è utilizzata

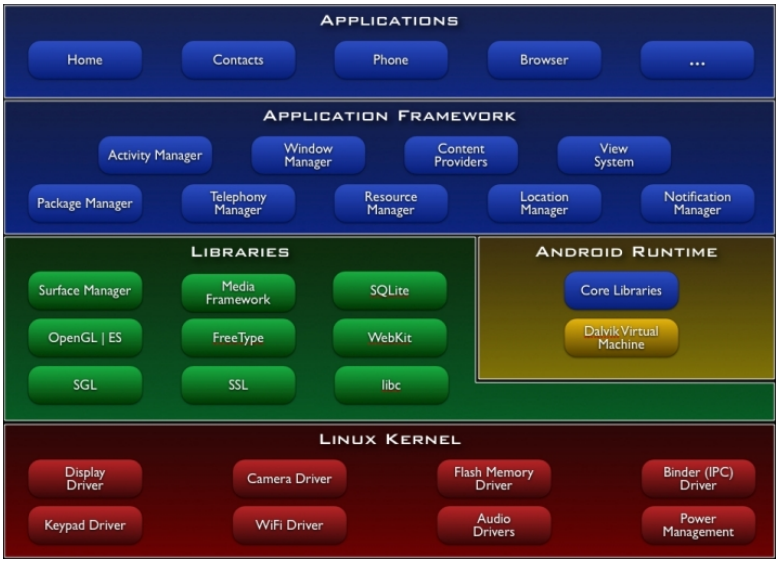

Figura 4.2: Architettura di Android

la piattaforma Java ME[1] perché, purtroppo, non ha avuto lo stesso successo delle sorelle Java SE[2] e Java EE[3] in quanto non ha mantenuto la promessa del *write once, run everywhere*. I dispositivi mobili in grado di eseguire delle applicazioni per J2ME sono moltissimi e molto diversi tra di loro. Realizzare delle applicazioni che funzionano bene e sfruttano a dovere le caratteristiche di ciascun dispositivo si è rivelata cosa impossibile. Un dispositivo può promettere di eseguire applicazioni allo stesso modo di un altro ma può avere dimensioni, risoluzione, modalità di input diverse. Per questo motivo si è portati a realizzare applicazioni specifiche per classi di dispositivi. La J2ME non è quindi stata pensata appositamente come piattaforma mobile ma come un'insieme di API che funzionano anche in dispositivi mobili.

La OHA e Google hanno quindi deciso che la cosa migliore per la realizzazione di una piattaforma mobile fosse quella di creare subito qualcosa di ottimizzato e specifico senza dover reinventare i componenti principali. Le possibilità erano quelle di creare un nuovo linguaggio oppure di utilizzarne uno esistente ereditando una community di sviluppatori e soprattutto un insieme maturo di strumenti di sviluppo. La scelta è caduta su Java non solo per questi motivi ma anche per la possibilità di avere a disposizione un bytecode di specifiche conosciute.

[1] J2ME: è un runtime e una collezione di API per lo sviluppo di software dedicato a dispositivi a risorse limitate

[2] J2SE: versione della Virtual Machine del linguaggio di programmazione Java

[3] J2EE: versione della piattaforma Java pensata per la realizzazione di applicazioni di enterprise e mission critical.

4.3.2 DVM - Dalvik Virtual Machine

Le applicazioni per Android, una volta scritte in linguaggio Java, vengono eseguite tramite la *Dalvik Virtual Machine*, una macchina virtuale adattata per l'uso su dispositivi mobili. La DVM è stata progettata da Dan Bornstein, dipendente Google, ed è uno dei componenti principali di Android. Grazie ad un utilizzo intelligente dei registri di sistema permette una maggiore ottimizzazione della memoria in dispositivi con bassa capacità, consente di far girare diverse istanze della macchina virtuale contemporaneamente e nasconde al sistema operativo sottostante la gestione della memoria e dei thread. Ovviamente la DVM non esegue bytecode Java ma un qualcosa che si può ottenere da esso e che prende il nome di Dalvik bytecode. Le applicazioni per Android si sviluppano in Java per poi trasformare il bytecode Java in bytecode Dalvik. Ecco che su un dispositivo Android non girerà alcun bytecode Java ma un bytecode le cui specifiche sono descritte dal formato `.dex`. Una curiosità: il nome Dalvik deriva dal villaggio di pescatori di cui la famiglia di Bornstein è originaria.

4.3.3 Framework per applicazioni

Gli sviluppatori hanno pieno accesso alle stesse framework API usate dalle applicazioni di base. L'architettura delle applicazioni è progettata per semplificare il riutilizzo dei componenti; ogni applicazione può rendere pubbliche le sue capacità e tutte le altre applicazioni possono quindi farne uso (sono soggette ai limiti imposti dalla sicurezza del framework). Questo stesso meccanismo consente all'utente di sostituire i componenti standard con versioni personalizzate.

Alla base di ogni applicazione si trova un set di servizi e sistemi[15], tra cui:

- *System View*: un gruppo ricco ed estensibile di strumenti che possono essere usati per costruire un'applicazione. Contiene liste, caselle di testo, pulsanti e addirittura un browser web integrato;
- *Content Providers*: permettono alle applicazioni di accedere a dati da altre applicazioni, come i contatti, o di condividere i propri dati;
- *Resource Manager*: offre l'accesso a risorse non-code come strings localizzate, grafica, files di layout;
- *Notification Manager*: permette a tutte le applicazioni di mostrare avvisi personalizzati nella status bar;
- *Activity Manager*: gestisce il ciclo di vita delle applicazioni e fornisce un backstack di navigazione comune;

4.4 JADE e l'SDK di Android

L'SDK (*Software Development Kit*) di Android è stato rilasciato dalla OHA per la prima volta a novembre 2007 fornendo negli anni diversi aggiornamenti. Essa include gli strumenti di sviluppo, le librerie, un emulatore del dispositivo, la documentazione (in inglese), alcuni progetti di esempio, tutorial e altro. È installabile su qualsiasi computer che usi come sistema operativo Windows XP, Vista, Mac OS X (dalla versione 10.4.8) o Linux. L'IDE (*Integrated Development Environment*) ufficialmente supportato per lo sviluppo di applicazioni per Android è Eclipse, per cui è fornito un plug-in progettato per fornire un potente ed integrato ambiente in cui costruire le applicazioni. L'SDK è basato sul linguaggio di programmazione Java che fornisce una serie di API specifiche per mezzo del quale è possibile interagire con il sistema operativo Android, controllare l'hardware del dispositivo e lo sviluppo dell'interfaccia grafica.

Figura 4.3: Esempio di emulatore del dispositivo

Come avevamo introdotto nel paragrafo 3.4.1, è stato recentemente rilasciato un add-on di JADE, chiamato JADE-Android. Al fine di essere compatibile con la DVM e di far fronte adeguatamente alle limitazioni e ai vincoli dei dispositivi mobili e reti wireless, l'add-on consente l'utilizzo di JADE-Leap sul sistema operativo Android. JADE-Leap, lo ricordiamo, permette di eseguire agenti JADE su dispositivi mobili con ridotte capacità di calcolo. Questo consente di limitare le comunicazioni attraverso il collegamento wireless, per quanto possibile, e di ottenere una comunicazione molto veloce tra peers mobili.

La possibilità di combinare la flessibilità del paradigma ad agenti mobili di JADE con la potenza della piattaforma di Android, porta un forte stimolo nello sviluppo di applicazioni innovative basate su modelli sociali e paradigmi P2P[4]. Grazie a JADE-Android un'applicazione Android può facilmente incorporare un agente JADE e pertanto diventare parte di un più ampio sistema distribuito, in cui

[4]peer-to-peer

siano eventualmente presenti anche altri dispositivi mobili (non necessariamente Android). Più in dettaglio l'add-on fornisce un'interfaccia che permette all'applicazione di avviare un agente locale, innescare behaviour e più in generale scambiare oggetti. È quindi possibile scoprire computer remoti, effettuare eventualmente con loro complesse conversazioni, sfruttare il supporto JADE per gestire i messaggi strutturati, insomma, approfittare di tutte le caratteristiche della piattaforma JADE.

CAPITOLO 5

SISTEMA SVILUPPATO

Lo scopo principale di questo lavoro è quello di sviluppare un sistema di autenticazione biometrico basato su agenti mobili, dove per autenticazione si intende un processo di verifica dell'identità di un soggetto, e per biometrico s'intende l'utilizzo delle tecniche tipiche della rilevazione di caratteristiche fisiologiche e comportamentali di cui si è già parlato nel capitolo 2.

Come già sottolineato, nonostante il movimento legato agli agenti mobili sia in grande fermento, in letteratura non si trovano molti esempi di un suo utilizzo legato a scopi di autenticazione anche se, all'interno del nostro gruppo dipartimentale, sono state svolte diverse ricerche in questo campo.

Partendo da questi presupposti si è pensato alla possibilità di allargare tale metodologia anche al mondo dei dispositivi mobili di ultima generazione, ovvero gli smartphone. Questi, oggigiorno, si stanno diffondendo in modo massiccio, e la creazione di nuove applicazioni da far girare al loro interno sta aprendo un nuovo orizzonte per gli sviluppatori.

L'innovazione introdotta consiste nel creare un sistema per l'autenticazione di utenti i quali dispongano, come strumento di interazione verso il sistema, non solo di macchine fisse messe a disposizione ad hoc dal soggetto controllante ma anche di *mobile device* di proprietà dell'utente.

Per lo sviluppo del sistema sviluppato in questo lavoro di tesi si è ipotizzato che l'utente possegga uno smartphone con sistema operativo Android 2.1 nel quale possa essere installato non solo un runtime di JADE-Leap Android, ma anche del software che permetta di utilizzare algoritmi propri della crittografia, sia a chiave pubblica sia a chiave privata, che risulteranno fondamentali per garantire la segretezza e la fidatezza delle informazioni scambiate durante la fase di autenticazione.

La demo realizzata utilizza il dispositivo mobile come deposito per il template dell'utente mentre la fase di matching viene seguita all'interno di un server. Ma prima di procedere ad una descrizione dettagliata del funzionamento e dell'architettura interna del progetto sviluppato, è necessario definire in modo chiaro gli aspetti legati all'autenticazione e capire il perché ci si orienta verso metodologie biometriche.

5.1 Autenticazione a più fattori

Un fattore di autenticazione può essere un'informazione o un processo usati per verificare l'identità di una persona a scopi di sicurezza. Quando un soggetto vuole procedere alla fase di autenticazione presenta, quindi, delle credenziali che possono essere classificate in:

- qualcosa che l'utente *conosce*;
- qualcosa che l'utente *possiede*;
- qualcosa che l'utente *ha in sé e può fisicamente esplicitare*;

Si parla di *strong authentication* (autenticazione a due o più fattori) quando vengono utilizzati almeno 2 fra le tipologie di credenziali sopra riportate.

5.1.1 Qualcosa che l'utente *conosce*

Questo tipo di credenziale è caratterizzata dalla conoscenza di un *PIN* o di una *password*, generalmente alfanumerica e di lunghezza variabile. L'utente deve essere l'unico depositario di questa informazione che fornirà al sistema per dimostrare la sua identità. Sono varie le problematiche derivanti dall'impiego di tale metodologia:

- l'utente deve essere conscio che l'inserimento della propria password su un sistema comporta un atto di fiducia nei confronti di quest'ultimo, in quanto potrebbe essere gestito da un soggetto maligno il cui unico scopo è ottenere le informazioni personali;
- l'utente poi può dimenticare questa informazione visto ormai l'enorme uso che ne si fa per molteplici servizi e risulta quindi facile confondersi. D'altra parte la sua annotazione scritta porta ad altri problemi come la copia o il furto;
- infine il segreto può essere indovinato o attraverso metodologie di forza bruta, che possono essere combattute usando password alfanumeriche molto lunghe, o attraverso tecniche basate su dizionari o attraverso stringhe create sulla

base del profilo del soggetto, per esempio si tende ad usare come password la propria data di nascita oppure il nome dei familiari o una loro combinazione;

5.1.2 Qualcosa che l'utente *possiede*

Credenziale basata sul solo possesso di un oggetto non facilmente clonabile che garantisce all'utente di effettuare l'autenticazione. Il più delle volte si tratta di possedere un *badge magnetico* oppure una *chiave meccanica* o un qualche dispositivo elettronico, come ad esempio una *chiave USB* o una *smartcard*.

È una tipologia eterogenea, nel senso che sono molteplici gli oggetti che possono costituire credenziale, e il livello di protezione è fortemente dipendente dall'oggetto scelto. Ad esempio una smartcard con capacità crittografiche fornirà una protezione maggiore rispetto ad un badge magnetico. In ogni caso c'è da risolvere il problema legato allo smarrimento e/o alla sottrazione dell'oggetto.

5.1.3 Qualcosa che l'utente *ha in sé e può fisicamente esplicitare*

In questo caso ci si trova nell'ambito della verifica di una caratteristica fisiologica o comportamentale del soggetto che richiede di essere autenticato, ovvero di una sua *caratteristica biometrica*. Tra le varie caratteristiche (presentate nel capitolo 2) quella maggiormente utilizzata è l'impronta digitale.

Questo tipo di credenziale presenta alcune problematiche come il costo necessario al rilevamento della caratteristica, l'invasività che per alcuni utenti potrebbe essere percepita come lesiva della propria privacy, l'accuratezza e la verifica dei vari sistemi di acquisizione che sono disponibili sul mercato e la non sostituibilità del parametro, nel senso che, se il parametro si danneggia, si ha un numero limitato di possibilità di sostituirlo.

I lati positivi, però, sono certamente maggiori, in quanto si risolve facilmente il problema della *sottraibilità*: le credenziali non possono essere smarrite, dimenticate o sottratte, anche se tuttavia questo ultimo punto non va dato per scontato e dovrebbe essere valutato a seconda della caratteristica biometrica e della tecnologia utilizzata. Non è poi da sottovalutare la comodità in quanto l'utente non deve più ricordarsi password od oggetti ma porta sempre con sé le credenziali di accesso.

5.2 Sistemi di autenticazione ibrida

Cercando di rispondere alle richieste della *strong authentication* sono stati sviluppati dei sistemi nei quali vanno a fondersi le varie tipologie di credenziali

appena descritte, al fine di creare un sistema che offra un adeguato livello di sicurezza.

I casi più frequenti di autenticazione ibrida sono:

- qualcosa che l'utente *conosce* e qualcosa che l'utente *possiede*: è il classico caso basato su carta magnetica e PIN numerico come ad esempio avviene per il sistema Bancomat/POS. Anche in questo caso la sicurezza è fortemente legata alle caratteristiche dell'oggetto posseduto: una smartcard crittografica rende il sistema molto più sicuro che non una tessera magnetica. La prima infatti è quasi impossibile da duplicare illecitamente, la seconda invece rappresenta soltanto un supporto di memorizzazione in cui non è effettuato alcun controllo sull'accesso ai dati, è quindi duplicabile quasi quanto una password. Ne sono una prova i vari episodi di clonazione delle tessere Bancomat;

- qualcosa che l'utente *possiede* e qualcosa che l'utente *possa esplicitare*: sistemi di questo tipo accoppiano informazioni biometriche che possono essere memorizzate su un particolare dispositivo. Il passaporto e la carta d'identità nazionale ne sono un esempio;

Durante lo sviluppo di questo progetto di tesi è stato scelto di adottare un sistema di autenticazione a tre fattori che si possono riassumere brevemente:

1. conoscenza di un PIN per l'accesso al sistema;

2. possesso di un telefonino di ultima generazione sul quale vengono memorizzate informazioni relative all'identità dell'utente e ai suoi parametri biometrici;

3. disponibilità al rilevamento live di un parametro biometrico dell'utente che verrà confrontato con quello salvato all'interno del telefonino;

5.3 Descrizione del sistema sviluppato

In questa sezione verranno descritti tutti gli scenari applicativi che sono stati sviluppati per questo lavoro di tesi. La presentazione qui si concentrerà sugli aspetti progettuali, in quanto l'analisi del codice e il manuale d'uso saranno esposti nei capitoli successivi. D'ora in poi quando si parlerà di parametro biometrico si intenderà un'impronta digitale rilevata tramite uno scanner.

L'algoritmo di matching utilizzato è basato sul confronto delle minuzie ed effettua quindi una ricerca al fine di trovare la migliore sovrapposizione tra esse tenendo conto sia delle possibili traslazioni sia delle possibili rotazioni verificabili. Nello sviluppo poi sono state utilizzati gli algoritmi RSA per la crittografia a chiave asimmetrica ed AES per la cifratura a chiave simmetrica.

5.3.1 Fase di Enrollment

La registrazione dell'utente è una parte estremamente importante al fine di un corretto funzionamento dell'intero sistema in quanto consiste nell'acquisizione dei dati personali dell'individuo, compresa la sua impronta digitale dalla quale poi sarà estratto un template.

Per il salvataggio dei dati dell'utente si è utilizzato un database risiedente su un server dove fosse presente un DBMS MySQL. Dal momento che si devono salvare solo informazioni relative a singoli soggetti (ovvero informazioni 1 a 1) il tutto è codificato all'interno del database *UserData* per mezzo di una singola tabella di nome *Clients*. I dati che vengono richiesti al momento della registrazione dell'utente sono:

- nome;
- cognome;
- numero di telefono;
- indirizzo e-mail;
- PIN;

Vengono generate una chiave pubblica ed una privata, da utilizzare per funzioni crittografiche a chiave asimmetrica, ed un'altra chiave privata da utilizzare invece per funzioni crittografiche a chiave simmetrica. Viene poi richiesto di rilevare l'impronta digitale (che può essere ad esempio sempre l'indice destro) e da quest'ultima si estrae un pattern che viene memorizzato all'interno del database.

5.3.2 Demo

In questa demo si fa uso dello smartphone come repository di dati, non solo per quanto riguarda il template biometrico ma anche per i dati identificativi del cliente e per le sue chiavi pubbliche e private. A livello generale questa demo permette di eseguire un'autenticazione biometrica di un utente al sistema: l'utente avvia sul proprio telefonino l'applicazione la quale, dopo aver richiesto a video l'inserimento del PIN personale dell'utente per l'accesso al sistema, crea un agente mobile sulla piattaforma nell'host client. A questo punto l'utente può richiedere alla postazione client, dove è presente lo scanner per le impronte, di iniziare l'autenticazione. Il server richiederà in automatico al telefonino di fornirgli i dati dell'utente per confrontarli con quelli presenti nel database MySQL e, se trovata una corrispondenza, verrà richiesto l'invio del template dell'utente e di quello live ricavato tramite lo scanner nella postazione client. A questo punto il server può comunicare l'esito dell'autenticazione.

Le operazioni step-by-step eseguite dal programma sono le seguenti:

1. nella postazione server è presente un'applicazione che attende richieste da parte di una applicazione client;

2. nella postazione client sono presenti lo scanner biometrico e un'applicazione che, per mezzo di una GUI, permette di richiedere l'avvio del processo di autenticazione (alla fine mostrerà il risultato del processo);

3. un utente che voglia avviare il processo di autenticazione deve avviare sul proprio smartphone, presso la postazione client, l'apposita applicazione. In primo luogo viene richiesto un PIN numerico di 5 cifre (che verrà utilizzato più avanti) e poi si collega al server JADE, crea un container speciale (di nome "User Phone Agent-"+ indirizzo logico server JADE + "-" + numero progressivo) e avvia un agente al suo interno di nome "User Phone Agent". A questo punto l'utente può richiedere l'avvio del processo di autenticazione biometrica cliccando sul pulsante dell'applicazione client;

4. l'applicazione client genera una richiesta di avvio del processo di autenticazione all'applicazione server;

5. appena giunta una richiesta viene instaurata una connessione SSL[1] tra le parti;

6. l'applicazione client genera un numero casuale (ID) e lo invia al server. Genera un container all'interno dello JADE Server con il nome "CLIENT-" + ID;

7. l'applicazione server, una volta ricevuto l'ID, genera un container all'interno dello JADE Server con il nome "SERVER-" + ID ed avvia un agente mobile al suo interno di nome "Biometric_User_Authenticator-" + ID;

8. l'agente del server crea un messaggio e lo salva all'interno di un Message-Container: quest'ultimo è un oggetto che può contenere un messaggio (come array di byte) detto *payload* ed un *challenge* (un valore float). La creazione del messaggio in questa fase iniziale prevede solo la generazione di un numero casuale come nuovo challenge e payload nullo;

9. l'agente cifra con la chiave AES del server il messaggio e poi lo firma digitalmente con la propria chiave privata;

[1]protocollo crittografico che permette una comunicazione sicura e una integrità dei dati su reti TCP/IP

10. viene stanziato un agente mobile dal nome “Mobile_Agent-” + n + “-” + ID (dove n è un numero progressivo che parte da 0 e viene incrementato ogni qual volta viene creato un agente) il quale prende il messaggio generato dal server, migra nel container del client ed invia il messaggio all’agente nel telefonino;

11. l’agente all’interno del telefono riceve il messaggio, ne controlla la firma digitale usando la chiave pubblica del server (nel caso in cui non fosse valida manda un messaggio di fallimento all’agente mobile che fa terminare l’intero processo di autenticazione segnalando l’avvenuto fallimento), decifra il messaggio usando la chiave AES del server, aggiorna il challenge (sommando 1 al precedente), inserisce come payload del messaggio la propria chiave pubblica, cifra il messaggio con la chiave AES del server, ne calcola la firma digitale con la propria chiave privata e invia il messaggio all’agente mobile nel container client;

12. quest’ultimo ritorna al server e consegna il messaggio dopodiché termina;

13. l’agente nel server riceve il messaggio dall’agente mobile lo decifra con la chiave AES del server ed estrae la chiave pubblica dell’utente (che è salvata come payload del messaggio). A questo punto può controllare la firma digitale del messaggio usando appunto la chiave pubblica dell’utente (nel caso in cui non fosse valida manda un messaggio di fallimento all’agente mobile che fa terminare l’intero processo di autenticazione segnalando l’avvenuto fallimento) dopodiché cifra la chiave pubblica del client con la chiave AES del server e la salva in memoria (questo per non lasciare in memoria dati sensibili in chiaro). Estrae il challenge, ne valuta la correttezza controllando che sia uguale al valore del challenge precedente più 1 (nel caso in cui non fosse valido manda un messaggio di fallimento all’agente mobile che fa terminare l’intero processo di autenticazione segnalando l’avvenuto fallimento) ed infine lo aggiorna (sommando uno). Ora azzera il payload, cifra il messaggio con la chiave AES del server e poi lo firma digitalmente con la propria chiave privata. Stanzia un nuovo agente mobile (il cui nome avrà il contatore aumentato di 1 rispetto al valore precedente) ed invia il messaggio a quest’ultimo;

14. il nuovo agente mobile migra nel container client ed invia il messaggio all’agente nel telefonino;

15. l’agente nel telefono riceve il messaggio ne controlla la firma digitale usando la chiave pubblica del client (nel caso in cui non fosse valida manda un messaggio di fallimento all’agente mobile che fa terminare l’intero processo di autenticazione segnalando l’avvenuto fallimento), decifra il messaggio con la chiave AES del server, estrae il challenge e lo aggiorna, imposta come

payload del messaggio il proprio valore di hash (presente nella SDCard dello smartphone sotto forma di file) ed il PIN inserito all'avvio dell'agente. Cifra il messaggio con la chiave AES del server e poi lo firma digitalmente con la propria chiave privata. Alla fine lo invia all'agente mobile in attesa nel container client;

16. quest'ultimo ritorna al server e consegna il messaggio dopodiché termina;

17. l'agente nel server riceve il messaggio dall'agente mobile, decifra usando la chiave AES del server la chiave pubblica dell'utente che aveva salvato cifrata in memoria e la usa per verificare la firma digitale del messaggio (nel caso in cui non fosse valida manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento). Decifra il messaggio con la chiave AES del server, effettua un controllo sul valore del challenge (nel caso in cui non fosse valido manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento) e poi lo aggiorna. Estrae il digest[2] ed il PIN, si collega al server MySQL e verifica la presenza dell'utente mediante l'hash (nel caso in cui non fosse un utente valido manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento). Controlla che il PIN nel database sia uguale a quello ricevuto (nel caso in cui non fosse un utente valido manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento) e in caso affermativo richiede al database anche la chiave AES dell'utente. Azzera il payload del messaggio, lo cifra con la chiave AES dell'utente (poi la cifra con la chiave AES del server e la salva in memoria) e poi lo firma digitalmente con la propria chiave privata. Stanzia un nuovo agente mobile (il cui nome avrà il contatore aumentato di 1 rispetto al valore precedente) ed invia il messaggio a quest'ultimo;

18. il nuovo agente mobile migra nel container client ed invia il messaggio all'agente nel telefonino;

19. l'agente nel telefono riceve il messaggio ne controlla la firma digitale usando la chiave pubblica del client, decifra il messaggio con la propria chiave AES, estrae ed aggiorna il challenge, imposta come payload del messaggio il proprio valore di fingerprint (presente nella SDCard dello smartphone sotto forma di file) cifra il messaggio con la propria chiave AES e poi lo firma digitalmente con la propria chiave privata. Alla fine lo invia all'agente mobile in attesa nel container client;

[2] calcolo di una particolare firma del dato

20. quest'ultimo ritorna al server e consegna il messaggio dopodiché termina;

21. l'agente nel server riceve il messaggio dall'agente mobile, decifra usando la chiave AES del server la chiave pubblica dell'utente che aveva salvato cifrata in memoria e la usa per verificare la firma digitale del messaggio (nel caso in cui non fosse valido manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento). Decifra il messaggio con la chiave AES dell'utente ed effettua un controllo sul valore del challenge (nel caso in cui non fosse valido manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento) e poi lo aggiorna, azzera il payload e salva il messaggio in memoria. Estrae il fingerprint e lo salva direttamente in memoria (non serve cifrarlo in quanto già cifrato) dopodiché crea un nuovo messaggio (quindi anche un nuovo challenge) il cui payload è la chiave pubblica del server, crea un nuovo agente mobile (il cui nome avrà il contatore aumentato di 1 rispetto al valore precedente) ed invia il messaggio a quest'ultimo;

22. l'agente mobile migra nel client ed ottiene il live fingerprint dell'utente tramite il lettore biometrico, lo cifra usando la chiave pubblica del server (che era salvata come payload del messaggio che portava con sé) e poi ritorna al server consegnando il messaggio al server. A questo punto termina;

23. l'agente nel server decifra il live fingerprint usando la propria chiave privata, decifra usando la chiave AES del server la chiave AES dell'utente salvata in memoria, la usa per decifrare il fingerprint dell'utente salvato in memoria e poi effettua l'operazione di match. Estrae il messaggio dalla memoria (salvato al punto 21) e imposta come payload il risultato del match. Cifra il messaggio con la chiave AES dell'utente e lo firma digitalmente con la propria chiave privata. Stanzia un nuovo agente mobile (il cui nome avrà il contatore aumentato di 1 rispetto al valore precedente) ed invia il messaggio a quest'ultimo. Invia la risposta all'applicazione server la quale tramite la connessione SSL la invierà all'applicazione client. A questo punto l'agente server può terminare ed anche il container viene eliminato;

24. l'agente mobile migra nel container client, invia il messaggio all'agente nel telefonino dopodiché termina ed il container client viene eliminato;

25. l'agente all'interno del telefono riceve il messaggio, ne verifica la firma digitale (in caso di errore avvisa l'utente) con la chiave pubblica del server, lo decodifica con la propria chiave AES e stampa a video il risultato dell'autenticazione. A questo punto può terminare;

5.3.3 Considerazioni sulle scelte implementative

Per questa dimostrazione, dal momento che fa uso di un dispositivo mobile dotato di una potenza di calcolo limitata, si è cercato di limitare al massimo l'utilizzo della crittografia a chiave pubblica preferendole quella a chiave privata, la quale permette di ottenere un ottimo livello di sicurezza con un costo computazionale notevolmente inferiore. Come si deduce infatti dalla lettura delle varie operazioni, la crittografia asimmetrica trova utilizzo solamente nell'instaurazione della comunicazione SSL tra il client ed il server, che sono macchine desktop dotate di potenza di calcolo sufficiente allo scopo, e per la firma digitale dei messaggi, dove si deve codificare solamente un digest di pochi byte. Per le altre operazioni di codifica, invece, si utilizza l'algoritmo crittografico a chiave simmetrica che prende il nome di AES. Le operazioni coinvolte sono:

- *connessione SSL*: operazioni 4, 5, 6 per l'instaurazione della connessione e per l'invio dell'ID e operazione 23 per l'invio dei risultati dell'autenticazione. Vengono svolte solamente da macchine desktop;
- *firma digitale dei messaggi e relativa verifica della loro validità*: operazioni 9, 11, 13, 15, 17, 19, 23, 25 dove alcune di esse vengono eseguite dal dispositivo mobile ma grazie alla limitata dimensione dei dati da codificare risultano essere estremamente veloci;
- *cifratura messaggio con chiave pubblica*: operazione 22 eseguita solamente su macchina desktop;
- *cifratura con chiave simmetrica*: stesse operazioni relative alla firma digitale dei messaggi. Ovviamente anche in questo caso alcune di esse vengono eseguite sul dispositivo mobile ma grazie a delle librerie opportunamente ottimizzate si ottengono tempi di esecuzione paragonabili a quelli misurabili su macchine desktop;

Dai test effettuati, grazie agli accorgimenti presi in fase di progettazione in riferimento all'uso degli opportuni algoritmi crittografici, una sessione di autenticazione viene eseguita in un tempo massimo di 5 secondi.

CAPITOLO 6

MANUALE UTENTE

L'obiettivo di questo capitolo è quello di fornire al lettore una guida da seguire nel caso in cui si voglia testare personalmente il software oggetto di questa tesi. Verranno quindi illustrate le corrette procedure di installazione dei vari programmi necessari e le modifiche da apportare ai vari script scritti appositamente per lanciare in modo rapido le varie applicazioni.

Si suppone che l'utente abbia già installato sul proprio computer una distribuzione Linux, come Ubuntu o Fedora. Il progetto è stato sviluppato e testato su Ubuntu 10.04 LTS 64-bit, tuttavia l'esecuzione del software non è vincolata a questa particolare distribuzione.

6.1 Installazione del software necessario

La prima cosa da fare è predisporre il sistema ovvero è necessario installare tutti i software di terze parti e le librerie utilizzate dalle varie dimostrazioni:

- *Java Development Kit v1.6 o superiore*: l'installazione non risulta particolarmente problematica in quanto è sufficiente scaricare dal sito della Sun il relativo pacchetto e seguire le istruzioni proposte a video;
- *Java Advanced Image*: una volta scaricato il pacchetto dal sito della Sun si dovrà entrare nella cartella ove risiedente la JVM (su Ubuntu dopo installazione JDK 1.6 /usr/lib/jvm/java-6-openjdk/) e da lì lanciare l'eseguibile che provvederà all'installazione della libreria;
- *Smartphone Android*: per poter utilizzare l'applicazione sul proprio smartphone è sufficiente installare il file *BiometricAuthenticationSystem.apk* seguendo

il manuale d'uso del telefono. Nel caso invece si voglia utilizzare l'emulatore, è sufficiente scaricare ed installare Android SDK e la piattaforma Android v2.1 o successiva. L'installazione di queste due componenti risulta molto semplice ed intuitiva. Dopodiché sarà possibile installare l'applicazione sull'emulatore grazie al tool *adb*;

- *Server MySQL*: scaricabile dal sito ufficiale l'installazione non presenta particolari problemi. Una volta installato è necessario importare il database fornito assieme al programma. Utilizzando *MySQL Administrator* sarà sufficiente collegarsi al database come utente root e poi, cliccando su *Restore Backup*, scegliere il file che contiene le istruzioni SQL per il ripristino del database;

- *JADE*: è sufficiente copiare nell'home dell'utente tutta la cartella fornita assieme al programma. Al suo interno saranno presenti tutte le librerie, in formato JAR, necessarie per il suo utilizzo e che saranno automaticamente aggiunte nel *CLASSPATH* dagli script forniti. Si ricorda inoltre che è necessario scaricare anche gli add-on *JADE-Leap* e *JADE-Android*. Entrambi andranno poi copiate all'interno della cartella JADE;

- *Driver Biometrika*: è da premettere che per l'installazione dei driver è necessario disporre dei sorgenti del kernel in uso. Una volta disponibili i sorgenti si potrà utilizzare la funzione *build.pl* (presente all'interno della cartella dei driver) per procedere alla compilazione dei driver. Una volta compilato il modulo dovrà essere installato (si dovrà avere l'accesso root al sistema) tramite la funzione *fxdriverinstall*, alla quale dovrà essere fornito il suo percorso. Una volta installato il modulo si dovrà copiare nella cartella /etc/ di sistema la cartella e tutte le sottocartelle presenti in etc/ all'interno della cartella dei driver. Fatto questo si potrà procedere all'avvio del modulo tramite la funzione *rc/fx2000.init.udev start*. Una volta caricato il modulo (si può verificare l'avvenuto caricamento tramite il comando dmesg) si dovrà avviare lo scanner tramite i seguenti comandi: *bin/FxReset -p 0* e *bin/FxReset -i 0*. L'avvio del modulo e l'inizializzazione dello scanner sono operazioni che devono essere ripetute ad ogni avvio del sistema (si può quindi inserire almeno il caricamento del modulo nel file `etc/rc.local` per farlo in automatico);

A questo punto sono stati svolti tutti i passi preliminari e il sistema è pronto per l'uso.

6.2 Enrollment Tool

Lo script di avvio di tale software si trova nella sotto-cartella *SCRIPT* della cartella *ENROLLMENT* e ha nome *ENROLLMENT.start*. Per poter avviare

correttamente il programma sarà necessario modificare solamente la variabile *HOME* inserendo l'indirizzo completo della propria directory home tra apici.

Il software, una volta avviato, mostra la propria interfaccia grafica. La prima operazione da compiere consiste nell'impostazione dei parametri per l'accesso al server MySQL: cliccando sul menù a tendina *MySQL -> Change connection parameters* apparirà una nuova finestra nella quale inserire i dati relativi alla connessione (cliccando su *Load* verranno caricati quelli di default); una volta inseriti cliccando su *Save* verranno memorizzati su un file e d'ora in poi utilizzati per connettersi al DBMS. Una volta completato il passo di configurazione si potrà avere accesso ai dati del database (ovviamente nel DBMS devono essere impostati i diritti di acceso e/o modifica relativi all'utente se non questi non è root) e di conseguenza si potranno compiere le seguenti operazioni:

- aggiunta di un utente: dopo aver inserito i dati in forma testuale nelle relative caselle, si procederà all'inserimento del template biometrico utilizzando come sorgente una immagine `.tif` (nel qual caso si utilizzerà il pulsante *Load Fingerprint Image from File*). La finestra di log mostrerà le operazioni compiute. Per ottenere una scansione da utilizzare come sorgente per l'enrollment è sufficiente aprire un terminale e, dopo aver inizializzato lo scanner ed aver appoggiato il dito sulla parte sensibile dello scanner stesso, digitare il comando *bin/FxReset -a fingerprint.tif*;

- visionare la lista degli utenti presenti nel database: cliccando sul pulsante *List* apparirà un'altra finestra con una tabella contenente una lista di tutti i record presenti nel database. Una volta selezionata una tupla, cliccando sul pulsante *Load selected data on main menu* i dati relativi all'utente selezionato verranno automaticamente caricati (ad esclusione dell'immagine del parametro biometrico che non viene salvata nel database ma viene salvato solo il template) e potranno essere editati oppure cancellati (per cancellazione si intende l'eliminazione completa del record dal database). Per l'aggiunta di un nuovo utente sarà necessario cliccare sul menù a tendina *File -> New*;

- editare un utente presente nel database: dopo aver caricato i dati relativi dell'utente prescelto sarà sufficiente editare i valori presenti nelle caselle di testo e poi cliccare sul pulsante *Edit* per salvare i cambiamenti. Si fa notare che non è possibile sostituire il parametro biometrico. Per l'aggiunta di un nuovo utente sarà necessario cliccare sul menù a tendina *File -> New*;

- eliminare un utente presente in lista: una volta caricato i dati di un utente cliccando su *Remove* si procederà alla sua eliminazione dal database;

Accanto alle funzioni necessarie ad inserire, editare e cancellare utenti all'interno del database, sono presenti anche metodi per:

- estrazione della chiave AES: cliccando sul pulsante *Extract AES Key* apparirà una finestra che permetterà di scegliere dove e con che nome salvare il file che conterrà al suo interno la chiave privata dell'utente utilizzata nella cifratura simmetrica;
- estrazione del template biometrico: cliccando sul pulsante *Extract Fingerprint Pattern* apparirà una finestra che permetterà di scegliere dove e con che nome salvare il file che conterrà al suo interno il template biometrico cifrato con la chiave AES dell'utente;
- estrazione chiave pubblica RSA: cliccando sul pulsante *Extract RSA Public Key* apparirà una finestra che permetterà di scegliere dove e con che nome salvare il file che conterrà al suo interno la chiave pubblica dell'utente utilizzata nella cifratura asimmetrica;
- estrazione chiave privata RSA: cliccando sul pulsante *Extract RSA Private Key* apparirà una finestra che permetterà di scegliere dove e con che nome salvare il file che conterrà al suo interno la chiave privata dell'utente utilizzata nella cifratura asimmetrica;
- estrazione del digest dei dati dell'utente: cliccando sul pulsante *Extract HASH* apparirà una finestra che permetterà di scegliere dove e con che nome salvare il file che conterrà al suo interno il valore del digest dei dati dell'utente (il digest SHA-256);
- preparazione dei dati da salvare nella memoria del telefono: cliccando sul pulsante *Prepare data for PHONE* apparirà una finestra che permetterà di scegliere la cartella ove salvare i dati che dovranno essere copiati all'interno della memoria del telefonino dell'utente affinché possa autenticarsi(si ricorda che devono essere copiate anche la chiave pubblica del server e la sua chiave AES);

6.3 Biometric Authentication System

Prima di procedere all'avvio della dimostrazione è necessario compiere alcuni operazioni preliminari, ovvero modificare i seguenti file.

All'interno della cartella *CLIENT/bua*:

- *Address.dat*: vanno inseriti l'indirizzo IP della macchina su cui è in esecuzione il server (può essere la stessa su cui è in esecuzione il client), e l'indirizzo IP del server JADE;

All'interno della cartella *SERVER/appListener*:

- *Address.dat*: vanno inseriti l'indirizzo IP della macchina su cui è in esecuzione il server, e l'indirizzo IP del server JADE;

All'interno della cartella *SERVER/biometricUserAuthenticator*:

- *KEYPATH*: qui vanno inseriti i percorsi completi delle chiavi pubbliche e private RSA del server (generabili con il tool fornito *RSA Key Generator*), della chiave pubblica dell'host e della chiave AES del server (generabile con il tool fornito *AES Utility*);
- *MySQLParameters*: dove si inseriscono i dati relativi alla connessione al DBMS MySQL;
- *CLIENTCONTAINERDATA*: all'interno del quale va inserito l'indirizzo IP della macchina su cui è in esecuzione il client (può essere la stessa macchina su cui è in esecuzione il server);

Per quanto riguarda il telefono è necessario copiarvi i dati creati al momento della registrazione dell'utente. Per fare questo basta aprire l'Enrollment Tool, caricare l'utente che si intende autenticare e, dopo aver collegato il telefono al pc tramite usb, esportare i suoi dati (usando il comando *Prepare data for PHONE*) all'interno del telefono. Nel caso in cui si utilizzi l'emulatore questa operazione eseguita tramite il comando *push* del tool *adb*.

Si dovranno poi aggiungere manualmente, sempre all'interno della memoria, i dati relativi al server ovvero la sua chiave privata AES (che deve essere nominata *SERVERKEY.AES*) e la sua chiave pubblica RSA (che deve essere nominata *SERVERPublicKey.dat*).

Ora non resta che modificare gli script di avvio sostituendo in tutti e tre la variabile HOME inserendo l'indirizzo completo della propria directory home. Nello script *JADE.start* va inoltre modificato il valore associato all'opzione *-host*, inserendo l'indirizzo IP del computer su cui verrà eseguito lo script.

Ovviamente la demo può essere eseguita anche su una singola macchina in locale. In questo caso, nei file precedentemente elencati, i valori dei vari indirizzi IP coincideranno con l'indirizzo locale (memorizzato nel file `/etc/hosts`).

A questo punto si possono avviare gli script in questo ordine:

- *JADE.start*: per avviare la piattaforma contenente il main container;
- *SERVER.start*: per avviare il server;
- *CLIENT.start*: per avviare il client;

Ora è possibile avviare l'applicazione sullo smartphone (o, in alternativa, sull'emulatore).

L'utilizzo della demo è estremamente semplice: prima di tutto, dopo aver cliccato il pulsante *Launch* sul display del telefono e aver selezionato l'indirizzo IP del JADE Server, si inserisce il PIN dell'utente che richiede l'autenticazione e si clicca su *Send PIN*. Nel caso in cui il telefono in uso sia fornito di touchscreen, basterà toccare due volte all'interno della finestra di inserimento del PIN per visualizzare la tastiera virtuale. Ora non resta che appoggiare il dito sullo scanner biometrico e cliccare su *Start Biometric User Authentication* nell'applicazione client. Se i dati presenti all'interno del telefonino non sono corretti (ovvero se l'hash ed il PIN forniti coincidono con quelli di un utente presenti nel database) il sistema lo comunicherà all'utente che potrà effettuare un nuovo tentativo. Se invece essi dovessero rivelarsi corretti, dopo aver effettuato la fase di matching dell'impronta, il sistema visualizzerà il risultato dell'autenticazione sia nella finestra apposita dell'applicazione client sia sul telefonino. A questo punto si potrà eseguire un'altra autenticazione cliccando sulla freccia indietro dello smartphone. Se si riscontrassero problemi di comunicazione tra i container JADE è necessario verificare che il file `/etc/hosts` contenga:

- indirizzo locale: nella forma 127.0.0.1 *localhost*;
- indirizzo di rete: nella forma IP *networkname machinename*, come ad esempio 147.26.53.192 bio2.dei.unipd.it;

CAPITOLO 7

MANUALE TECNICO

Dopo aver descritto il funzionamento generale del sistema e aver fornito gli strumenti necessari per testare personalmente il sistema, in questa sezione verranno presentate le modalità di implementazione dei vari software riportando, quando opportuno, alcuni pezzi di codice che possano aiutare a comprenderne meglio la struttura e a favorirne lo sviluppo.

Tutto il progetto è stato sviluppato in linguaggio Java. Sia la fase di sviluppo sia la fase di testing sono state condotte sia su un computer desktop con sistema operativo Linux[1], sia su macchine collegate in rete in quanto tutti i parametri necessari ad un suo funzionamento, come esposto nelle sezioni successive, sono letti da file di configurazione tranquillamente editabili; per la parte del telefonino si è utilizzato uno smartphone HTC Desire con sistema operativo Android 2.1 e come ambiente di sviluppo si è utilizzato Eclipse. Accanto al sistema sono state prodotte due librerie, *Utility* e *Utility Android*, che raccolgono le funzioni più utilizzate dai vari programmi come ad esempio l'accesso ai file, la lettura dei parametri di configurazione e la scrittura/lettura delle chiavi di cifratura.

Nella nostra demo, il software di autenticazione, come si è già avuto modo di constatare, è caratterizzato, oltre che da una piattaforma ad agenti mobili, anche da tre applicazioni:

- lato smartphone, con il nome di Biometric Authentication System;
- lato client, con il nome di Biometric User Authentication;
- lato server, che prende il nome di Application Listener;

[1] distribuzione Ubuntu 10.04

7.1 Biometric Authentication System

Come già illustrato nei capitoli precedenti, il JADE runtime disponibile per Android è una versione molto semplificata rispetto a quella che può essere eseguita su un computer desktop e il suo unico scopo è quello di collegarsi ad un host, a sua volta collegato al main container della piattaforma, e lì creare il *BackEnd* container il quale si carica del compito della registrazione presso il main container e, allo stesso tempo, creare il *FrontEnd* container sullo smartphone.

L'indirizzo di rete della postazione host in cui verrà creato il *BackEnd* potrà essere scelto e selezionato da una lista che verrà visualizzata sul display del telefono. Per l'avvio del runtime, JADE-Android prevede una propria procedura completamente diversa da quella utilizzata da JADE-Leap.I passi principali da seguire sono:

1. creare una classe Jade Agent che estenda *GatewayAgent*;
2. all'interno della classe creata al punto precedente sovrascrivere il metodo *processCommand*;
3. creare un'activity che implementi *ConnectionListener*;
4. chiamare, all'interno del metodo *onCreate* dell'Activity, il metodo JadeGateway.connect, passandogli, attraverso un oggetto di tipo *Properties*, tutti i parametri necessari per la connessione;
5. implementare il metodo onConnected(JadeGateway gateway) dell'interfaccia *ConnectionListener* per creare un'istanza dell'oggetto GatewayAgent;
6. chiamare il metodo *execute* dell'oggetto di tipo *JadeGateway* per mandare un comando all'agente;
7. chiamare il metodo *disconnect* per terminare la connessione;

Listing 7.1: Avvio ambiente JADE su telefonino

```
1  public class Start extends Activity implements ConnectionListener {
2
3          private JadeGateway gateway;
4          private BridgeJadeAndroid updater;
5
6          @Override
7      public void onCreate(Bundle savedInstanceState) {
8          super.onCreate(savedInstanceState);
9          setContentView(R.layout.start);
10
```

```
11          updater = new BridgeJadeAndroid(this);
12
13          Properties props = new Properties();
14          props.setProperty(Profile.MAIN_HOST,
15                  UserPIN.getSelectedIP());
16          props.setProperty(Profile.MAIN_PORT, getResources().
17                                  getString(R.string.port));
18          props.setProperty(JICPProtocol.MSISDN_KEY,
19                                  getResources().
20                                  getString(R.string.msisdn));
21
22          try {
23                  JadeGateway.connect(UserAuthenticatorAgent.
24                          class.getName(),
25                          null, props, this, this);
26                  }
27          catch (Exception e) {
28                  Toast.makeText(this, e.getMessage(), 5000);
29                  }
30      }
31
32          public void onConnected(JadeGateway gw) {
33                  gateway = gw;
34
35                  try {
36                          gateway.execute(updater);
37                          }
38                  catch (StaleProxyException e) {
39                          e.printStackTrace();
40                          }
41                  catch (ControllerException e) {
42                          e.printStackTrace();
43                          }
44                  catch (InterruptedException e) {
45                          e.printStackTrace();
46                          }
47                  catch (Exception e) {
48                          e.printStackTrace();
49                          }
50          }
51
52          public void onDisconnected() {
53
54          }
55
56          // Chiusura della connessione con JADE
57          protected void onDestroy() {
58                  super.onDestroy();
59
```

```
60                  try {
61                          if (gateway != null)
62                                  gateway.shutdownJADE();
63
64                  } catch (ConnectException e) {
65
66                          Toast.makeText(this, e.getMessage(),
67                                  Toast.LENGTH_LONG);
68                  }
69                          if (gateway != null)
70                                  gateway.disconnect(this);
71          }
72  }
```

7.2 Biometric User Authentication

Tale applicazione ha lo scopo di presentare una GUI per instaurare una connessione SSL con l'applicazione server, generare l'ID di sessione ed avviare l'agente mobile client. Per conoscere quale sia l'indirizzo del server con il quale richiedere l'avvio della connessione protetta il software ne legge l'indirizzo su un file, editabile con i propri dati, di nome `Address.dat`. Per la generazione della connessione sicura poi necessita del percorso dei certificati che contengono le chiavi pubbliche e private delle due entità: tali percorsi vengono letti da un file di nome `DataCertificates.dat`.

Listing 7.2: Richiesta di connessione al server e connession SSL

```
1  String [] data=new String [2]; //data=Address + Port Number
2  data[0]=utility.FileUtility.getDataFromDataFile
3          (AddressFile, "ADDRESS");
4  data[1]=utility.FileUtility.getDataFromDataFile
5          (AddressFile,"PORT");
6  socket=utility.NetUtility.getConnection
7          (utility.NetUtility.getAddress(data[0]),
8          Integer.parseInt(data[1])); //get connection
9
10 //Read ACK from Application Listener
11 if(!utility.NetUtility.readFromSocket(socket).equals("ACK")) {
12         return("An error occured. I can't bind my application to
13                 Server" +"\nTry Again...");
14     }
15
16 //Read port of SSL Connection from socket
17 int portSSLConnection=
18         Integer.parseInt(utility.NetUtility.readFromSocket(socket));
19
```

```
20 socket.close(); //close clear connection
21
22 //Create an SSL Connection with Proxy
23 //Validate trustStoreServer
24 while(sslSocket==null) {
25         sslSocket=utility.NetUtility.getClientSSLSocket(
26                     utility.FileUtility.getDataFromDataFile
27                     (DataCertificatesFile, "CERTIFICATES_PATH"),
28                     utility.FileUtility.getDataFromDataFile
29                     (DataCertificatesFile, "CLIENT_CERTIFICATE_NAME"),
30                     utility.FileUtility.getDataFromDataFile
31                     (DataCertificatesFile, "SERVER_CERTIFICATE_NAME"),
32                     utility.FileUtility.getDataFromDataFile
33                     (DataCertificatesFile, "CLIENT_PASSWORD"),
34                     utility.FileUtility.getDataFromDataFile
35                     (DataCertificatesFile, "SERVER_PASSWORD"),
36                     utility.FileUtility.getDataFromDataFile
37                     (AddressFile, "ADDRESS"),
38                     portSSLConnection );
39         }
```

L'avvio dell'agente mobile sfrutta le API messe a disposizione dalla piattaforma JADE. Prima di tutto è necessario creare un oggetto *Runtime* che permetta di accedere all'ambiente runtime JADE; poi si deve creare un profilo attraverso la creazione dell'oggetto *Profile* che permette di impostare tutti i parametri necessari al corretto avvio dell'agente come l'indirizzo della piattaforma (anch'esso reso disponibile all'interno del file `Address.dat`), il nome del container che deve essere creato, i servizi che devono essere caricati, ecc.... Alla fine, tramite l'oggetto *Runtime* si possono creare il container ed avviare l'agente al suo interno.

Listing 7.3: Avvio dell'agente su piattaforma JADE

```
1  //Create CLIENT Agent in JADE Platform
2  //Parameters for CLIENT
3  Runtime runtime=Runtime.instance(); //get runtime environment
4  Profile pClient=new ProfileImpl(); //make new profile for client
5  pClient.setParameter(Profile.MAIN_HOST,
6          utility.FileUtility.getDataFromDataFile(AddressFile,
7          "JADE_HOST")); //set JADE Host
8  pClient.setParameter(Profile.MAIN_PORT,
9          utility.FileUtility.getDataFromDataFile(AddressFile,
10         "JADE_PORT")); //set JADE Port
11 pClient.setParameter(Profile.CONTAINER_NAME, "CLIENT-" +ID);
12 ccClient=runtime.createAgentContainer(pClient);
```

Una volta avviato l'agente l'applicazione rimane in attesa del risultato dell'autenticazione.

7.3 Application Listener

Tale applicazione risiede sul server e ha il compito di ricevere le richieste da parte delle applicazioni client e di instaurare, con ognuna di esse, una connessione protetta SSL (i percorsi dei certificati sono presenti su un file di testo del tutto simile a quello cui ha accesso l'applicazione client). Una volta creato il collegamento sicuro viene stanziato un oggetto di tipo *Proxy* il quale ha il compito di avviare il container server e i relativi agenti mobili al suo interno (le modalità sono del tutto simili a quelle descritte in ambito client) e di inviare il risultato dell'autenticazione non appena è a disposizione. Al momento non è previsto che l'*ApplicationListener* crei un thread specifico per ogni connessione, quindi è ammissibile una sola connessione per volta: non appena viene restituito il risultato è possibile effettuare un'altra sessione di autenticazione.

7.4 Agenti Mobili

La parte principale della dimostrazione consiste nell'utilizzo degli agenti mobili. La programmazione di essi in ambito JADE prevede di strutturare il loro set di operazioni come una insieme di comportamenti che possono essere eseguiti nell'ordine voluto. In fase di progettazione si è scelto di strutturare l'insieme complessivo delle operazioni dell'agente come una sequenza di comportamenti, dove ogni singolo comportamento è stato implementato come classe al fine di poter riutilizzare parte del codice nelle occasioni in cui un comportamento dovesse essere utilizzato più volte.

7.4.1 BiometricUserAuthenticator

L'agente principale risiede sul server e presenta questa sequenza di comportamenti:

- *richiesta dell'hash dell'utente*: prima di tutto viene creato un oggetto di tipo *MessageContainer* formato da un array di byte che serviranno a contenere il messaggio da trasmettere, e da un'istanza dell'oggetto *Challenge* che servirà a memorizzare il valore di un challenge generato dal server che, assieme alla firma digitale, servirà come mezzo per verificare l'autenticità delle risposte ricevute. Una volta stanziato tale oggetto (al cui interno quindi è presente un nuovo challenge e una richiesta per l'hash dell'utente) viene calcolata la sua firma digitale (ovvero viene calcolato un digest che poi viene cifrato con la chiave privata del server), viene cifrato con la chiave AES del server ed il tutto viene passato come parametro ad un nuovo agente mobile, di nome *Mobile_ Agent*, il quale migra sul container client (il cui indirizzo viene

ritrovato tramite una funzione scritta ad hoc presente sulla libreria Utility che interroga l'AMS[2]) e successivamente invia il messaggio all'agente dello smartphone. L'agente legge le chiavi pubbliche e private del server da un file di testo, editabile a piacimento, di nome `KEY_PATH`;

- *attesa di una risposta*: l'agente attende che *Mobile_ Agent* ritorni sul server con le relative risposte;

- *controllo del challenge*: *Mobile_ Agent* è ora ritornato sul server e ha riportato il messaggio modificato dall'agente del telefono; quest'ultimo viene decifrato, ne viene verificata la firma digitale e poi viene controllato il valore del challenge. Se non si sono verificati errori il challenge viene aggiornato in caso contrario viene attivato il comportamento di autodistruzione e l'autenticazione termina con risultato negativo;

- *controllo della presenza dell'utente nel database*: una volta ricevuto l'hash dell'utente si crea una connessione al DBMS MySQL, e lo si interroga al fine di verificare la presenza dell'utente e, se esiste, si prosegue, altrimenti si richiama il comportamento di autodistruzione e l'autenticazione termina con risultato negativo;

- *invia Mobile_ Agent a rilevare il live fingerprint*: viene inviato al client l'agente mobile affinché rilevi il template biometrico live dell'utente;

- *attesa di una risposta*: l'agente attende che *Mobile_ Agent* ritorni sul server con le relative risposte;

- *esecuzione del match e terminazione*: una volta decifrato il contenuto del messaggio, si interroga nuovamente il database per estrarre il template dell'utente memorizzato, si stanzia la classe *FindMatch* e tramite il metodo *getTemplatesFromByteArray(byte firstT, byte secondT)*, si caricano i due pattern (quello live e quello memorizzato); infine tramite una chiamata alla funzione *matchTemplate()* si ricava uno score che se superiore alla soglia prefissata (in fase di progettazione si è deciso di scegliere 30 minuzie) permette di autenticare correttamente l'utente. A questo punto non resta che avvisare il proxy dell'avvenuta autenticazione e poi il tutto viene distrutto tramite il comportamento di autodistruzione;

[2]Agent Management System: componente che supervisiona gli accessi alla piattaforma da parte di entità esterne

Listing 7.4: Esecuzione del match ed invio risultato

```
1  //Get template from byte
2  FindMatch matcher=new FindMatch();
3
4  matcher.getTemplatesFromByteArray(userFingerPrint,
5          liveFingerPrint); //Get user name
6  String user=new String(utility.AES.BCdecode
7          (utility.AES.readKey(SERVER_AES_KEY),
8          (byte[])getDataStore().get("USER_NAME")));
9
10 //Calculate Match and print results of matching
11 //if I find more than 30 equal minutiae
12 if(matcher.matchTemplate()>30) {
13         String  message=new String("USER "+user+" AUTHENTICATED");
14         System.out.println("["+(String)getDataStore().get("MY_NAME")+
15                 "]:" +message);
16         //set message like payload of message container
17         mc.setMessage(message.getBytes());
18         //create a mobile agent that inform phone of result
19
20         BehaviourCreateMobileAgent cma=new BehaviourCreateMobileAgent
21                 (thisAgent,mc,USER_AES_KEY);
22         cma.setDataStore(this.getDataStore());
23         thisAgent.addBehaviour(cma);
24         //create a file with result
25         File result=new File("result.dat");
26         utility.FileUtility.writeFile(message.getBytes(),result);
27         //now I finished
28         return;
29         }
30 else { //NOT AUTHENTICATED
31         String  message=new String("USER "+user+" NOT AUTHENTICATED");
32         System.out.println("["+(String)getDataStore().get("MY_NAME")+
33                 "]:" +message);
34         //set message like payload of message container
35         mc.setMessage(message.getBytes());
36         //create a mobile agent that inform phone of result
37         BehaviourCreateMobileAgent cma=new BehaviourCreateMobileAgent
38                 (thisAgent,mc,USER_AES_KEY);
39         cma.setDataStore(this.getDataStore());
40         thisAgent.addBehaviour(cma);
41         //create a file with result
42     File result=new File("result.dat");
43     utility.FileUtility.writeFile(message.getBytes(),result);
44         //now I finished
45     return;
46     }
```

7.4.2 Mobile

Accanto all'agente che risiede sul server è presente un agente mobile che migra verso il client e che ha il compito di trasportare i messaggi dal server al telefono e di acquisire la live fingerprint. Esso ha un codice strutturato secondo una serie di comportamenti di cui, i più importanti, sono quelli relativi alla migrazione, in cui è necessario conoscere il container di destinazione rappresentato da un AID, e alla ricezione ed invio di messaggi ACL verso il telefono che, essendo eseguito in modalità asincrona, sfrutta l'utilizzo di un buffer. È stato implementato anche un controllo sul nome del mittente del messaggio e sulla performative ad esso associata al fine di evitare attacchi del tipo mail bombing da utenti sconosciuti in quanto tutti i messaggi che non provengono da agenti facenti parte del sistema vengono scartati (notare che questa metodologia risulta efficace in quanto ad ogni sessione di autenticazione i nomi degli agenti contengono, nella parte finale, un numero casuale valido solo per quella sessione).

Listing 7.5: Mobilità dell'agente

```
1  //Migration of agent
2  //if I have a destination AID
3  if(destination!=null) {
4          thisAgent.doMove(destination);
5          }
6  else { //I have data of container
7          ContainerID destinationContainer=new ContainerID();
8      destinationContainer.setName(destinationContainerName);
9          destinationContainer.setAddress(destinationAddressContainer);
10     destinationContainer.setPort(destinationPortContainer);
11     thisAgent.doMove(destinationContainer);
12     }
```

Listing 7.6: Send e receive di messaggi ACL

```
1  //send message
2  if(send) {
3          this.sendMessage(); //cal function that send message
4          }
5  else { //receive message
6          ACLMessage incoming=thisAgent.receive();
7          //receive incoming message
8
9      if(incoming!=null) {
10         //Control that sender is correct sender
11         if((incoming.getSender().getLocalName().equals(senderName))
12                 && ((incoming.getPerformative()==ACLMessage.REQUEST)
13                  || (incoming.getPerformative()==ACLMessage.INFORM)
14                  || (incoming.getPerformative()==ACLMessage.FAILURE)))
```

```
15                          {
16                                  received=true;
17                  //CONTROL IF IT'S A FAILURE
18                  if(incoming.getPerformative()==ACLMessage.FAILURE) {
19                          //a failure is arrived
20                          //so send a message to server and
21                      //destroyed myself
22                      ACLMessage mex=new ACLMessage(ACLMessage.FAILURE);
23                      AID server=new AID();
24                      server.setLocalName("Biometric
25                          User Authenticator");
26                      mex.addReceiver(server);
27                      thisAgent.send(mex);
28                      System.out.println("[Mobile User Data Requester-"
29                          +(String)getDataStore().get("ID")+"]:" +
30                          "an error was occured.
31                          Authentication process " +
32                          "fail.");
33                      //Failure so I must kill agent
34                      thisAgent.addBehaviour
35                          (new BehaviourKillAgent(thisAgent));
36                      return;//terminate this behaviour
37                          }
38                                  //Save the message in DataStore
39                  this.getDataStore().put("MESSAGE",
40                          incoming.getContent());
41                          }
42                  else {
43                //I have receive the message so I can exit from
44                //blocking mode
45                received=false;
46                }
47                  }
48          else {
49                  //block agent until message arrive
50          this.block();
51          received=false;
52          }
53          }
```

7.4.3 UserPhoneAgent

Anche l'agente stanziato sul telefono è stato strutturato come un sequenza di comportamenti:

- *attesa del challenge da parte del server*: l'agente rimane in attesa che nel suo buffer arrivi un messaggio proveniente dal server contenente il challenge

da modificare e rinviare al server come primo metodo per dimostrare la sua fidatezza;

- *decodifica del messaggio e modifica del challenge*: una volta arrivato il messaggio ne viene verificata la firma digitale (ovviamente facendo terminare tutto il processo di autenticazione se questa non fosse valida), viene decifrato usando la chiave AES del server (che è salvata assieme agli altri dati nella memoria del telefono), si procede all'aggiornamento del challenge e all'impostazione della propria chiave pubblica come payload del messaggio,ed infine si codifica il MessageContainer e lo si firma digitalmente con la propria chiave privata, dopodiché lo si invia all'agente in attesa sul client (si noti che nel codice si fa molto ricorso alle librerie sviluppate, in questo caso UtilityAndroid);

Listing 7.7: Elaborazione del messaggio

```
1  //get message with signature
2  byte [] messageWithSignature=getMessageByteArray();
3  //get signature
4  byte [] signature=utility.Android.RSAAndroid.
5  extractSignatureFromMessageWithSignature
6          (messageWithSignature);
7  //get message
8  byte [] message=utility.Android.RSAAndroid.
9          extractMessageFromMessageWithSignature
10                 (messageWithSignature);
11 //verify signature of message if failure stop
12 //authentication process
13 if(!verifyServerSignature(signature,message)) {
14         printMessage(R.id.TextView02,"ERROR - Signature
15         is not ok.
16     \nMessage was mod by someone
17     or it was not send by Server.
18     \nAuthentication process will stop.");
19     thisAgent.addBehaviour
20     (new BehaviourShutDownAuthentication(thisAgent));
21     }
22
23 [...]
24
25 printMessage(R.id.TextView02,"SERVER is ready to start
26         authentication proccess");
27
28 //decode message
29 byte[] messageDecoded;
30 try {
31         messageDecoded = startAES(message, false, "SERVER");
32
```

```
33      //Set new message with payload = my public key
34      byte [] replyMessage=setNewMessage(messageDecoded,
35          getMyRSAKey(true));
36      //Crypt replyMessage with ServerKey
37      byte[] replyEncryptMessage;
38
39      replyEncryptMessage = startAES(replyMessage, true, "SERVER");
40
41      //Sign message
42      byte [] signReplyMessage=utility.Android.RSAAndroid.RSASign
43          (replyEncryptMessage,
44          utility.Android.RSAAndroid.readPrivateKey
45          (getMyRSAKey(false)));
46      String replyMessageHEX=utility.Android.HexUtilityAndroid.
47          toHex(signReplyMessage)+utility.Android.
48          HexUtilityAndroid.toHex(replyEncryptMessage);
49      //Send reply
50      ACLMessage reply=new ACLMessage(ACLMessage.INFORM);
51      AID destination=new AID();
52          destination.setLocalName((String)getDataStore().
53          get("MOBILE_AGENT_NAME"));
54      reply.addReceiver(destination);
55      reply.setContent(replyMessageHEX);
56      thisAgent.send(reply);
57      printMessage(R.id.TextView03,
58          "I sent reply that I'm also ready to start");
59      //update number of mobile agent
60      int value=Integer.parseInt((String)getDataStore().
61                              get("NUM_MOBILE_AGENT"));
62      getDataStore().put("NUM_MOBILE_AGENT",""+(++value));
63      //Update name of mobile agent
64      updateNameOfMobileAgent();
65
66          }
67  catch (FileNotFoundException e) {
68          // TODO Auto-generated catch block
69          e.printStackTrace();
70          }
71
72  return; //finish
```

- *attesa della risposta da parte del server*: l'agente rimane in attesa di un messaggio proveniente dall'agente mobile sul client che si fa carico di trasportare le risposte del server;

- *invio hash e PIN su server*: una volta ricevuta la risposta da parte del server estrae dalla propria memoria l'hash dei suoi dati ed il PIN che era stato

inserito manualmente dall'utente all'avvio dell'agente (mediante interfaccia grafica) e li inserisce come payload del nuovo messaggio (ovviamente esegue sempre i controlli sulla validità della firma digitale) lo cifra e lo invia all'agente sul client;

- *attesa risposta server e invio fingerprint*: se il server ha identificato l'utente allora l'agente può inviargli il template. Da questo punto in poi i messaggi vengono cifrati con la chiave AES dell'utente anch'essa presente nella memoria del telefono (notare l'utilizzo della funzione startAES(byte [] msg,boolean encrypt,String nameOfKey) dove come nome della chiave viene passato " MY " ovvero quella relativa all'utente);

Listing 7.8: Invio template

```
1  printMessage(R.id.TextView08,"SERVER wants fingerprint");
2  //decode message
3  byte[] messageDecoded;
4  try {
5          messageDecoded = startAES(message, false, "MY");
6
7          //set new message with payload Fingerprint
8          byte [] fingerprint=getData(hash);
9          byte [] replyMessage=setNewMessage(messageDecoded,
10                 fingerprint);
11         //Crypt replyMessage with My AES Key
12         byte[] replyEncryptMessage;
13
14         replyEncryptMessage = startAES(replyMessage, true, "MY");
15
16         //Sign message
17         byte [] signReplyMessage=utility.Android.RSAAndroid.
18                 RSASign(replyEncryptMessage,utility.
19                 Android.RSAAndroid.readPrivateKey
20                 (getMyRSAKey(false)));
21         String replyMessageHEX=utility.Android.
22                 HexUtilityAndroid.toHex
23                 (signReplyMessage)+
24                 utility.Android.HexUtilityAndroid.toHex
25                 (replyEncryptMessage);
26         //Send reply
27         ACLMessage reply=new ACLMessage(ACLMessage.INFORM);
28         AID destination=new AID();
29         destination.setLocalName((String)getDataStore().
30         get("MOBILE_AGENT_NAME"));
31         reply.addReceiver(destination);
32         reply.setContent(replyMessageHEX);
33         thisAgent.send(reply);
```

```
34          //update number of mobile agent
35          int value=Integer.parseInt((String)getDataStore().
36                  get("NUM_MOBILE_AGENT"));
37          value+=2;
38          getDataStore().put("NUM_MOBILE_AGENT",""+(value));
39          //Update name of mobile agent
40          updateNameOfMobileAgent();
41          printMessage(R.id.TextView09,
42                  "I sent my fingerprint to SERVER");
43          }
44  catch (FileNotFoundException e) {
45          // TODO Auto-generated catch block
46          e.printStackTrace();
47          }
48
49  return; //finish
```

- *attesa risultato autenticazione e relativa stampa a video*: l'agente ora rimane in attesa del risultato dell'autenticazione e, dopo averlo stampato a video, richiama il comportamento di autodistruzione.

CONCLUSIONI

L'obiettivo principale della tesi era quello di sviluppare un'applicazione per smartphone che permettesse l'autenticazione biometrica con tre caratteristiche principali: l'uso degli agenti mobili, l'uso di un parametro biometrico e l'uso del sistema operativo Android per lo smartphone. Mediante l'uso della piattaforma JADE e dell'add-on JADE-Leap Android, si è potuto sviluppare un modello che andasse verso questa direzione e, dal punto di vista dimostrativo, il tutto funziona con tempi di attesa accettabili dato che una sessione di autenticazione con matching impiega circa 5 secondi.

Dal un punto di vista progettuale ci si è soffermati sulle problematiche relative alla sicurezza, aspetto molto importante in quanto c'è un alto numero di scambi di dati personali. Si sottolinea che la sicurezza delle comunicazioni è garantita dall'uso massiccio della crittografia, non solo per la cifratura dei dati ma anche per la salvaguardia della loro integrità e verifica del mittente. Si è cercato di creare, utilizzando gli strumenti attualmente a disposizione, un ambiente che permettesse di effettuare questi scambi in totale sicurezza anche mediante l'utilizzo di canali implicitamente insicuri.

La realizzazione implementativa del sistema ha portato allo sviluppo di una dimostrazione che permette di verificarne la potenzialità utilizzando strumenti e dati reali, in quanto è previsto l'uso di uno scanner biometrico per l'acquisizione delle impronte dell'utente e di uno smartphone di ultima generazione come repository dei dati. Come si è già esposto nei capitoli precedenti, questa metodologia di autenticazione non solo mostra l'efficacia degli strumenti di protezione ma, in particolar modo, la versatilità del progetto realizzato che permette di effettuare autenticazioni con l'utilizzo di dispositivi reali.

Uno sviluppo di questo progetto potrebbe essere quello di implementare il sistema in modo tale che, una volta che l'utente viene autenticato, esso possa accedere a particolari servizi.

BIBLIOGRAFIA

[1] Il linguaggio di programmazione Java, *http://www.java.com/it/*.

[2] La distribuzione Linux Ubuntu, *http://www.ubuntu-it.org/*.

[3] Il Java Development Toolkit, *http://java.sun.com/javase/downloads/index.jsp*.

[4] La libreria JAI, *http://java.sun.com/javase/technologies/desktop/media/2D/*.

[5] Il Database MySQL, *http://www.mysql.it/*.

[6] La piattaforma JADE (*Java Agent DEvelopment platform*), *http://jade.tilab.com/*.

[7] BiometriKa S.r.l., azienda produttrice dello scanner biometrico, *http://www.biometrika.it/*.

[8] Il sistema operativo Android, *http://www.android.com/*.

[9] S. Furlan, *Realizzazione di un server per identificazione mediante impronte digitali*, Tesi di laurea, Università di Padova, 2001.

[10] A. Panazzolo, *Acquisizione di un template robusto per autenticazione mediante impronte digitali*, Tesi di laurea, Università di Padova, 2004.

[11] A. Morandotti, *Un'infrastruttura per la registrazione di esami tramite autenticazione biometrica*, Tesi di laurea, Università di Padova, 2008.

[12] M. Tranquillin, *Sistema sicuro di autenticazione biometrica ad agenti mobili su Jade*, Tesi di laurea, Università di Padova, 2007.

[13] JADE-Android add-on guide, *http://jade.tilab.com/*.

[14] Lo standard FIPA, *http://www.fipa.org/*.

[15] M. Carli, *Android - Guida per lo sviluppatore*, APOGEO, 2010.