Progettazione e realizzazione di un'applicazione per il controllo degli accessi in Android

Guido Davide Dall'Olio

Indice

Introduzione

L'analisi delle vulnerabilità, il rischio annesso, le minacce e gli attacchi fanno parte di quella branca dell'informatica nota come Sicurezza Informatica. Negli ultimi anni il mercato degli smartphone è cresciuto a ritmi incalzanti e la complessità dei sistemi operativi installati sui dispositivi mobili permette l'esecuzione di applicativi e software sempre più complessi. Mentre negli ultimi anni la sicurezza poteva essere relegata a elaboratori, mainframe, calcolatori, oggi è una tematica d'interesse e necessaria, la protezione dei terminali mobili.
Nel quotidiano, la scelta di uno smartphone, porta l'utente a dover interfacciarsi a differenti sistemi operativi: Android, iOS, Firefox OS, Ubuntu for phone, Tizen, Bada, etc. In questo progetto ci concentreremo sull'OS mobile più diffuso, ovvero Android, cercando di implementare un sistema di controllo degli accessi. Android, essendo già piuttosto evoluto, predispone alcune tecnologie utili a migliorare la sicurezza sul terminale.

Capitolo 1

Introduzione Android

Android è una moderna piattaforma mobile progettata per essere opensource. Le applicazioni Android fanno uso di software e hardware avanzati, così come di dati locali e remoti, per portare innovazione e valore ai consumatori attraverso la piattaforma. Per proteggere quel valore, il sistema operativo deve offrire un ambiente applicativo che garantisca la sicurezza dei dati, degli utenti, delle applicazioni, del terminale stesso e della rete.
Garantire la sicurezza di una piattaforma aperta è una sfida e necessita di una architettura solida e di programmi ben strutturati. Android è stato progettato con una sicurezza multi livello che fornisce la flessibilità necessaria per una piattaforma aperta. I controlli sono stati progettati per ridurre agli sviluppatori l'onere di implementare metodologie e nuove architetture, persino chi si approccia per la prima volta allo sviluppo ottiene già una protezione basilare in automatico. Gli utenti hanno visibilità e controllo sulle applicazioni, prima ancora di installare nuovo software potranno scegliere se i permessi richiesti soddisfino le esigenze dell'utente.

1.1 Parti Fondamentali

I blocchi logici fondamentali della piattaforma sono:

- Hardware del dispositivo: eseguendo su una vasta gamma di configurazioni hardware, Android sfrutta alcune funzionalità di sicurezza intrinseche in alcuni processori, come quelle presenti nell'architettura ARM v6;

- Sistema Operativo Android: basato su kernel Linux;

- Android Application Runtime: le applicazioni Android sono principalmente scritte nel linguaggio di programmazione Java ed eseguite dalla macchina virtuale Dalvik. Molte applicazioni e servizi sono nativi e sfruttano librerie native, ma sia la macchina Dalvik sia le applicazioni native eseguono all'interno dello stesso ambiente di sicurezza, contenuto all'interno della sandbox Android. Ad ogni applicazione viene dedicata una porzione del file system, in cui può scrivere dati privati, compresi database e dati raw.

Le applicazioni Android estendono il sistema operativo di base Android. Ci sono due tipologie principali:

- Applicazioni pre-installate: Android include applicazioni pre-installate quali telefono, email, calendario, browser web e contatti. Queste eseguono in modalità utente e garantiscono funzionalità chiave che possano essere accedute da applicazioni terze. Tali applicazioni pre-installate possono esser parte del sistema operativo Android o sviluppate da un OEM per il dispositivo specifico;

- Applicazioni utente: Android supporta applicazioni sviluppate da terzi e ne permette l'installazione tramite il Play Store ufficiale.

Viene fornito anche un insieme di servizi cloud-based, disponibili per qualsiasi dispositivo, i principali:

- Google Play: un insieme di servizi che consente agli utenti di scoprire, installare e acquistare applicazioni tramite il loro dispositivo o tramite

il web. Il Play Store facilita gli sviluppatori nel raggiungere gli utenti finali, aggiungendo la possibilità di recensire, valutare, controllare licenze, sicurezza applicativa e altri servizi di sicurezza;

- Aggiornamenti Android: il servizio di aggiornamento offre nuove funzionalità e aggiornamenti di sicurezza attraverso il web o over-the-air (OTA);

- Servizi Applicativi: framework che consentono alle applicazioni Android di usare funzionalità cloud quali il backup dei dati e delle impostazioni.Questi servizi non fanno parte del progetto opensource, ma sottostanno alle stesse regole di sicurezza.

1.2 Programma di Sicurezza Android

Nella fase iniziale di sviluppo, il team di sviluppo Android ha riconosciuto che un modello di sicurezza robusto era necessario per dar vita a un vigoroso ecosistema di applicazioni e dispositivi costruiti attorno la piattaforma Android e supportati dai servizi cloud. Come risultato, costantemente durante il periodo di sviluppo, l'os è stato sottoposto ad un programma di sicurezza professionale. Il team di sviluppo ha avuto la possibilità di osservare come altre piattaforme mobili, desktop e server prevenissero e reagissero a problemi di sicurezza, in modo così da affrontare le possibili minacce causate dai vari punti deboli.
I componenti chiave del programma di sicurezza di Android includono:

- Design Review: il processo di sicurezza comincia presto nel ciclo di vita di sviluppo, con la creazione di un modello di sicurezza ricco e configurabile. Ogni caratteristica principale è preso in esame da ingegneri, con controlli di sicurezza appropriati integrati nell'architettura del sistema;

- Penetration Testing and Code Review: durante lo sviluppo della piattaforma Android, sia i componenti generati sia quelli opensource sono

soggetti a controlli di sicurezza rigorosi. Tali test sono eseguiti dall'Android Security Team, dalla squadra di Sicurezza Informatica Google e da consulenti esterni. L'obiettivo di queste analisi è quello di individuare i punti deboli e possibili vulnerabilità ancora molto prima che la piattaforma sia aperta al pubblico e di simulare i tipi di revisioni che verranno eseguite da esperti in sicurezza esterni al momento del rilascio;

- Open Source e Community Review: Android sfrutta anche tecnologie opensource già sottoposte a revisioni e sicurezza, come ad esempio il kernel Linux;

- Incident Response: anche con tutte queste precauzioni, alcuni problemi di sicurezza possono presentarsi anche dopo il rilascio al pubblico ed è per questo che il progetto Android ha creato un completo processo globale di risposta di sicurezza. A tempo pieno il team di sicurezza monitora costantemente le comunità in rete di discussione su potenziali vulnerabilità. Dopo la scoperta di legittime problematiche, si attua un rapido processo di risposta che permette la rapida mitigazione delle vulnerabilità al fine di ridurre al minimo il rischio per tutti gli utenti. La politica di protezione si attua aggiornando la piattaforma Android sul dispositivo mediante aggiornamenti Over The Air (OTA), rimuovendo applicazioni dal servizio Google Play e persino eliminando applicazioni dai dispositivi medesimi.

1.3 Architettura della Piattaforma di Sicurezza Android

Android cerca di essere il sistema operativo più sicuro e fruibile per le piattaforme mobili cercando di indirizzare i controlli del sistema operativo e proteggendo:

- Dati utente;

- Risorse di sistema;
- Fornire isolamento dell'applicazione.

Per raggiungere questi obiettivi, fornisce queste funzionalità di sicurezza fondamentali:

- Sicurezza robusta a livello OS tramite il kernel Linux;
- Sandbox obbligatoria per tutte le applicazioni eseguenti;
- Comunicazione tra i processi sicura;
- Firma digitale delle applicazioni;
- Permessi a livello di applicazione e utente.

La sezione seguente descrive questi e altre funzionalità di sicurezza della piattaforma.

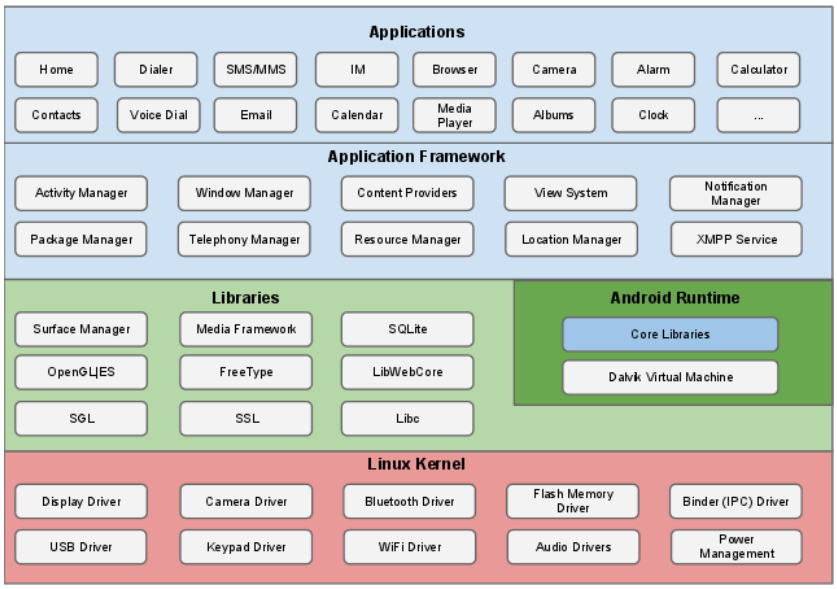

Figura 1.1: Architettura e Organizzazione

Capitolo 2

Livello di Sicurezza Kernel e di Sistema

A livello di sistema operativo, la piattaforma Android offre la sicurezza del kernel Linux, così come una sicura comunicazione tra processi (IPC) per consentire la comunicazione sicura tra le applicazioni in esecuzione in processi diversi. Queste funzioni di sicurezza a livello di sistema operativo assicurano che anche un codice nativo sia vincolato alla sandbox dell'applicazione. Sia che il codice sia il risultato del comportamento delle applicazioni native o uno sfruttamento di una vulnerabilità nell'applicazione, il sistema cerca di impedire che l'applicazione maligna danneggi altre applicazioni o il dispositivo stesso.

2.1 Sicurezza Linux

La base di partenza del sistema operativo è il kernel Linux, ampiamente utilizzato da anni, è sfruttato in milioni di ambienti sensibili alla sicurezza. Attraverso la sua storia, costantemente oggetto di ricerca, è continuamente attaccato e aggiornato da migliaia di sviluppatori, diventando così un kernel estremamente stabile e affidabile per moltissime aziende e professionisti.

Garantisce ad Android differenti funzionalità chiave in ambito di sicurezza, tra cui:

- Modello permessi definito dall'utente;
- Isolamento tra i processi;
- Solida sicura comunicazione IPC;
- Capacità di rimuovere parti inutili o potenzialmente insicure del kernel.

Come per un sistema operativo multiutente, l'obiettivo fondamentale di sicurezza del kernel Linux è l'isolamento delle risorse utente tra i vari utenti. La filosofia di sicurezza è quella di proteggerle dall'accesso di altri utilizzatori della piattaforma. Così, Linux:

- Previene che l'utente A legga i file dell'utente B;
- Assicura che l'utente A non esaurisca la memoria di B;
- Assicura che l'utente A non esaurisca le risorse computazionali dell'utente B;
- Assicura che l'utente A non blocchi l'accesso alle interfacce accessorie del dispositivo all'utente B.

2.2 Sandbox a livello di Applicazione

La piattaforma Android si avvale della protezione basata sulla medesima tipologia fornita alla sicurezza utente nel kernel Linux. Il meccanismo di identificazione utente viene sfruttato per l'isolamento delle risorse dell'applicazione. Il sistema ad ogni applicazione Android assegna un ID utente univoco (UID) e la esegue come un processo separato. Questo approccio è differente rispetto ad altri sistemi operativi, in cui di norma più applicazioni vengono eseguite con le stesse autorizzazioni e lo stesso utente.

Ciò instaura una Sandbox a livello kernel. Viene inoltre rafforzata la sicurezza tra le applicazioni e il sistema a livello di processo tramite le strutture standard di Linux, come ad esempio gli ID di gruppo. A default, le applicazioni non possono interagire tra loro e hanno accesso limitato al sistema operativo. Se l'applicazione A tenta di eseguire qualcosa di dannoso, come leggere i dati dell'applicazione B o comporre un numero di telefono, il sistema interviene individuando immediatamente i privilegi non adeguati all'esecuzione. La Sandbox è un meccanismo semplice, verificabile e basato sulla separazione utente in Unix e sui permessi dei singoli file.
Dal momento in cui la Sandbox applicativa è realizzata nel kernel, questo modello di sicurezza si estende facilmente al codice nativo e alle applicazioni del sistema operativo. Tutto il software al di sopra del kernel, come in Figura 1.1, comprese le librerie del sistema operativo, il framework applicativo, l'ambiente di esecuzione applicativo e tutte le altre applicazioni eseguono all'interno della Sandbox. Su alcune piattaforme, gli sviluppatori sono costratti a un quadro specifico di sviluppo, a un set specifico di API o un particolare linguaggio per rafforzare la sicurezza. Su Android, invece, non sono presenti alcune restrizioni su come un'applicazione possa essere scritta, in questo modo il codice nativo è sicuro tanto quello interpretato.
In alcuni sistemi operativi, gli errori di corruzione della memoria portano generalmente a compromettere completamente la sicurezza del dispositivo. Questo non avviene in Android grazie alla applicazioni e le risorse sotto la Sandbox a livello di OS. Un'alterazione della memoria permetterà solo l'esecuzione di codice arbitrario in quel particolare contesto di quella applicazione, con le sole autorizzazioni previste dal sistema operativo.
Come tutte le caratteristiche di sicurezza, la Sandbox dell'applicazione non è indistruttibile. Per poterla violare è necessario violare il kernel Linux.

2.3 Partizione di Sistema e modalità Safe

La partizione di sistema contiene il kernel di Android, così come le librerie del sistema operativo, applicazioni runtime, il framework applicativo e le

applicazioni. Questa partizione è impostata in sola lettura. Quando un utente avvia in modalità Safe, solo le applicazioni native Android saranno disponibili. Ciò assicura che l'utente possa avviare il telefono in un ambiente libero da software di terze parti.

2.4 Crittografia

Android fornisce una serie di API di crittografia a supporto delle applicazioni. Tali API includono implementazioni standard e comuni di tecniche e primitive quali AES, RSA, DSA e SHA. Inoltre esistono ulteriori primitive a supporto di SSL e HTTPS.

2.5 Root del dispostivo

Per impostazione predefinita, su Android solo un piccolo sottinsieme di applicazioni native e il kernel stesso eseguono con permessi di root. Android non impedisce ad un utente o a un'applicazione con tali permessi di modificare il sistema operativo, il kernel o qualsiasi altra applicazione. In generale, l'utente root ha pieno accesso a tutte le applicazioni e tutti i dati delle applicazioni, bypassando in questo modo qualsiasi Sandbox o partizione protetta. Gli utenti che modificano le autorizzazioni sul dispositivo, per acquisire tali privilegi, si espongono maggiormente a possibili applicazioni dannose e potenziali attacchi.
La possibilità di modificare un dispositivo Android è molto importante per gli sviluppatori. Su molti dispositivi esiste la possibilità di sbloccare il bootloader per consentire l'installazione di un OS alternativo. Tali sistemi operativi possono consentire al proprietario di acquisire i privilegi di superutente a scopo di debug o accedere a ulteriori funzionalità.
Su alcuni dispositivi, una persona con controllo fisico sul dispositivo e un cavo USB è in grado di installare un nuovo sistema operativo che fornisca privilegi root. Per proteggerei dati utente esistenti, il meccanismo di sbloc-

co del bootloader richiede l'eliminazione di tutti i dati personali dell'utente come parte della procedura stessa. L'acquisizione dei privilegi mediante un bug del kernel o di sicurezza, può scavalcare questa protezione ulteriore.
Cifrare i dati con una chiave memorizzata sul dispositivo, non protegge i dati delle applicazioni da utenti root. Le applicazioni possono aggiungere un livello di protezione dei dati con crittografia mediante una chiave memorizzata al di fuori del telefono stesso, ad esempio un server o una password utente. Tale approccio può fornire una protezione temporanea, ma non esclude le potenzialità dell'utente root.
Un approccio più robusto alla protezione dati da utenti privilegiati è attraverso l'utilizzo di soluzioni hardware. Gli OEM possono scegliere di implementare soluzioni hardware che limitino l'accesso a determinati tipi di contenuti, come DRM per la riproduzione video o NFC associato al Google Wallet.
Nel caso di furto o smarrimento, la cifratura completa del filesystem su dispositivi Android sfrutta la password dell'account utente per proteggere la chiave di cifratura, quindi la modifica del bootloader o del sistema operativo non sono sufficienti per ottenere l'accesso ai dati utenti.

2.6 Cifratura del Filesystem

Da Android 3.0 in poi è presente la possibilità di cifrare il filesystem completo, in modo che tutti i dati utente possano essere criptati, mediante l'implementazione di AES128 con CBC e ESSIV:SHA256. La chiave di crittografia è protetta con AES128 mediante la password dello stesso account utente. Per fornire resistenza agli attacchi sistematici (es. rainbow tables o brute force), la password è combinata con un sale casuale e ne viene effettuato un hash ripetuto con SHA1 utilizzando lo standard PBKDF2 prima di essere utilizzata per decifrare la chiave del filesystem. Per fornire ulteriore resistenza contro i dizionari di password, Android aggiunge regole sulla complessità delle password che possono essere attivate dall'amministratore del dispositivo e rafforzate dal sistema operativo stesso.
Android può essere configurato per verificare una password fornita dall'u-

tente prima di fornire l'accesso a un dispositivo. Oltre a prevenire l'uso non autorizzato del dispositivo, questa password protegge la chiave di crittografia per la cifratura del filesystem stesso.

2.7 Memorizzazione delle Credenziali

Per impostazione predefinita, Android include una serie di autorità di certificazione predefinite (CA) che siano sicure per le operazioni quali creazione di connessioni SSL all'interno del browser. Da Android 4.0 in poi, gli utenti possono disabilitare CA preinstallate tramite le impostazioni di sistema. Si possono persino inserire nuove CA o certificati di sistema. Da Android 4.1 è anche possibile aggiungere, per gli OEM, un portachiavi basato su hardware che leghi le chiavi al dispositivo utilizzato.

Capitolo 3

Sicurezza Applicazioni Android

Le applicazioni Android sono perlopiù scritte nel linguaggio di programmazione Java e eseguite sulla macchina virtuale Dalvik, ma possono essere scritte anche in codice nativo. Tutte vengono installate tramite un file singolo con estensione Apk.
Le parti fondamentali di un'applicazione sono:

- AndroidManifest.xml: file di controllo che spiega al sistema cosa fare con tutti i componenti di primo livello (in particolare quali attività, servizi e fornitori di contenuti di seguito descritti) in un'applicazione. Specifica anche le autorizzazioni richieste all'utente;

- Activity: rappresenta tipicamente il codice di un singolo, task orientato all'interazione con l'utente. Comprende di norma la visualizzazione di un'interfaccia e di norma è il punto d'ingresso in un'applicazione;

- Service: è una parte di codice che viene eseguita in background, può essere eseguita in un processo o nel contesto di un altro processo applicativo. Altri componenti effettuano un 'bind' su un Service e ne invocano metodi attraverso chiamate a procedura remota. Un esempio è il riproduttore musicale, mantiene la riproduzione anche quando l'utente esce dall'interfaccia grafica principale;

- Broadcast Receiver: oggetto che viene creato quando un Intent, derivante dal meccanismo IPC, viene inoltrato al sistema operativo da un'applicazione. Le applicazioni possono registrarsi in ascolto di vari eventi e cambiare il proprio comportamento in base alle informazioni di contesto ricevute.

3.1 Sistema dei Permessi: Accesso a API protette

Tutte le applicazioni su Android possono accedere a risorse presenti sul dispositivo, previa un'autorizzazione. Il sistema gestisce gli accessi delle applicazioni alle risorse che, se usate illecitamente o non correttamente, potrebbero ripercuotersi negativamente sull'esperienza utente o dati salvati.
Queste restrizioni vengono implementate in forme diverse. Alcune funzionalità sono limitate intenzionalmente, mancando API per alcune periferiche e alcune azioni (ad esempio, non vi è alcuna API per manipolare direttametne la scheda SIM). In alcuni casi, la separazione dei ruoli fornisce una misura di sicurezza, come l'isolamento tra le applicazioni dei direttori di salvataggio. In altri casi, le API sensibili sono destinate all'uso da applicazioni attendibili e protette da autorizzazioni.
Le API protette:

- funzioni Fotocamera;
- Dati di localizzazione (GPS);
- funzioni Bluetooth;
- funzioni di Telefonia;
- funzioni SMS/MMS;
- connessioni di rete/dati.

Queste risorse sono unicamente accessibili attraverso il sistema operativo. Per sfruttare le API protette, un'applicazione deve definire nel proprio manifest ciò di cui necessità. Quando ci si prepara a installare un nuovo applicativo, il sistema visualizzerà una finestra di dialogo per l'utente indicante i vari permessi richiesti. Sarà poi l'utente a decidere l'attendibilità di tale applicazione e valutare se fornire i permessi richiesti all'esecuzione. Tali richieste possono essere solo accettate in toto, completamente, oppure negandole. Non è possibile decidere di accettare solo alcuni permessi richiesti.
Una volta concesse, le autorizzazioni vengono applicate a tale software finché rimane installato sul dispositivo. Per evitare confusione all'utente, il sistema non notifica ad ogni avvio le autorizzazioni concesse. Le applicazioni native non richiedono alcun interazione con l'utente riguardo i permessi, essendo già preinstallate sul dispositivo.
All'interno delle impostazioni del dispositivo, gli utenti sono in grado di visualizzare le autorizzazioni per le applicazioni che hanno già installato. È anche possibile disattivare alcune funzionalità a livello globali, quali GPS, radio, Wi-Fi.
Nel caso in cui un'applicazione tenti di utilizzare una funzione protetta, che non è stata dichiarata nel manifest, verrà generata un'eccezione di sicurezza di sistema. La verifica di API protette di sistema prende luogo a livello di sistema più basso possibile, cercando quindi di evitare elusioni. Un esempio di seguito:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.
    ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
<uses-permission android:name="android.permission.
    AUTHENTICATE_ACCOUNTS" />
```

Ci sono alcune funzionalità del dispositivo, come ad esempio la possibilità di inviare SMS, che non sono disponibili alle applicazioni di terze parti, ma possono essere sfruttate da applicazioni preinstallate sul dispositivo. Queste autorizzazioni utilizzano il permesso 'signatureOrSystem'.

3.2 Come gli utenti si approcciano alle applicazioni terze

Android si sforza di far comprendere all'utente che sta interagendo con un'applicazione di terze parti e informarlo sulle funzionalità di queste. Ancor prima dell'installazione di qualsiasi applicazione, all'utente viene proposta una finestra con i differenti permessi che il software necessita per l'esecuzione corretta. Dall'installazione in poi, non saranno più richieste autorizzazioni. Un esempio di finestra di autorizzazioni lo si trova nella Figura 3.1. che si traduce in questa finestra di richiesta all'atto dell'installazione: Ci sono molte ragioni

Figura 3.1: Permessi Richiesti

per mostrare i permessi immediatamente prima l'installazione. Prima di tutto è il momento in cui l'utente sta esaminando attivamente le informazioni sull'applicazione, sviluppatore e funzionalità per determinare se soddisfano le proprie esigenze e aspettative. La visione di Android è quella di avere utenti che cambino senza problemi da un'applicazione all'altra. In aggiunta

molti studi sulle interfacce utente hanno mostrato come la continua generazione di finestre di richiesta di permessi, porti l'utente a iniziare a scegliere una risposta affermativa per ogni finestra visualizzata, sorvolando quindi sul concreto problema di sicurezza. Uno degli obiettivi principali è quindi quello di trasmettere in modo efficace importanti informazioni di sicurezza per l'utente; ciò non può essere fatto utilizzando varie finestre di dialogo, bensì una soltanto quando l'utente è propenso a valutare. Inattese richieste di autorizzazioni possono portare gli utenti più smaliziati a porre domande critiche sulle funzionalità richieste e condividere le proprie preoccupazioni in luoghi come Google Play, dove sono visibili a tutti gli utenti.

3.3 Informazioni personali

Android ha posizionato API che forniscono accesso ai dati utente sotto il set di API protette. Con un uso normale, i dispositivi Android potranno anche accumulare i dati degli utente all'interno delle applicazioni di terze parti installate dagli utenti. Le applicazioni che scelgono di condividere queste informazioni possono utilizzare il sistema di controllo dell'OS al fine di proteggere i dati da applicazioni terze. I content provider che potrebbero contenere informazioni personali, quali contatti e calendario, sono stati creati con autorizzazioni chiaramente identificate. Questa granularità dà all'utente una visione chiara dei tipi di informazione che possono essere fornite. Durante l'installazione, un'applicazione terza, potrebbe quindi richiedere l'accesso a queste risorse. Dal momento in cui viene accettata e installata, tale applicazione potrà accedere ai dati in qualsiasi momento. Tutte le applicazioni che raccolgono dati personali saranno, a default, identificate come uniche e avranno accesso limitato. Se un'applicazione sceglie di rendere i dati condivisibili attraverso un meccanismo IPC, può forzare il sistema a garantirli anche nello scambio di messaggi.

3.4 Applicazioni firmate

La firma di un'applicazione permette agli sviluppatori di identificare univocamente un autore e aggiornare l'applicazione senza creare complicate autorizzazioni e interfacce. Ogni software eseguente sul sistema Android, deve essere firmato dallo sviluppatore. Applicazioni non firmate sono respinte direttamente dal Google Play e dal programma di installazione dell'OS.
Sul Google Play, la firma collega la fiducia che Google ha con lo sviluppatore con la fiducia che ha lo stesso sviluppatore con l'applicazione. Gli sviluppatori conoscono il proprio applicativo e come è presentato, sono di conseguenza responsabili per il comportamento delle loro app.
Su Android, la firma di un applicativo è il primo step d'affrontare per poter posizionarlo nella Sandbox nativo. Il certificato di firma definisce quale id utente è associato all'applicazione; di conseguenza diverse applicazioni sono eseguite con ID utente differenti. La firma garantisce in aggiunta che le varie applicazioni non possano accedere a qualsiasi altra informazione, se non attraverso i meccanismi ben definiti di IPC.
Quando un'applicazione (mediante il file Apk), viene installata su un dispositivo, il Package Manager verifica che sia correttamente firmata con il certificato incluso all'interno del file. Se il certificato (più precisamente la chiave pubblica del certificato), corrisponde alla chiave usata per firmare qualsiasi altro Apk sul dispositivo, il nuovo potrà avere la possibilità di specificare nel manifest di condividere un UID.

Capitolo 4

Analisi Progetto

L'attività che vedremo qui di seguito cerca di mettere in pratica alcuni strumenti e tecniche viste a lezione e presenti nativamente nelle ultime release di Android OS e Java. Il progetto si basa su una interazione client server su connessione sicura di tipo SSL-TLS.
Il client Android, dopo essere stato identificato dal dispositivo come autorizzato, cerca di instaurare un collegamento sicuro per poter accedere ad alcuni file presenti sul server. Tali file simuleranno un servizio di storage di dati appartenenti a più account dell'utente associati a servizi online quali Dropbox, Amazon, Facebook. Il cliente richiedendo al server l'accesso, otterrà una lista modificabile di servizi memorizzati e relativi dettagli d'accesso (username e password) per poter accedere ai servizi elencati.
Per la realizzazione sono stati utilizzati:

- Smartphone Android;
- Android OS 4.2.2;
- PC desktop;
- Eclipse ADT v22;
- Java 1.7 release 21;

- Librerie Bouncy Castle;
- tool Keytool presente in Java;
- Java Cryptography Extension (JCE) Unlimited Strength;
- Apache Commons codec;
- Android-SDK v22.

Partiamo quindi dalla realizzazione del client Android.

4.1 Autenticazione in Android - Account Manager

Al fine di identificare in maniera univoca sulla piattaforma Android l'utente si potevano adottare diverse soluzioni.
Poteva essere sufficiente un file in locale cifrato sul quale salvare gli utenti, ma tale procedura sarebbe stata rischiosa nel caso di utenti root.
Si è quindi optato per un meccanismo già presente sull'OS Android, che garantisce isolamento tra le diverse applicazioni e supporta i dati di accesso dei vari utenti: Account Manager.
Tale modulo nasce principalmente per venire in aiuto a tutti quei problemi comuni di autenticazione all'interno delle applicazioni. Ricordare l'utente che ha effettuato un login in un sistema, caricare le stesse impostazioni anche in un dispositivo differente, risultano immediati tramite l'utilizzo di Account Manager. Allo stesso è poi possibile richiedere la lista di account presenti su un dispositivo, per cercare di effettuare nuovamente la procedura di login.
Può essere sfruttato principalmente in due modi:

- Gestendo localmente la registrazione degli account;
- Facendo gestire remotamente la registrazione degli account, ma salvandosi localmente il token di autorizzazione.

Nel progetto è stata utilizzata la prima strada.
Vediamone un semplice grafo esplicativo, vedi Figura 4.1. Principalmente l'Account Manager si comporta da registro centralizzato degli account. L'utente inserisce le credenziali una volta e automaticamente il tutto viene salvato su un database locale e a sola lettura del Manager. Tale database, per il momento non è cifrato, ma la lettura è impedita a qualsiasi applicazione differente dall'account manager. Tuttavia un utente privilegiato, può comunque navigare tra i file di sistema e alterarli, come potrebbe anche leggere il suddetto file: per questo motivo, si è scelto di cifrare i contenuti salvati all'interno del database manualmente.
La versatilità di tale registro è data anche dal fatto che garantisca la sola gestione della memorizzazione dei dati, lasciando la logica di autenticazione in mano allo sviluppatore, non imponendo alcun vincolo in merito. Per

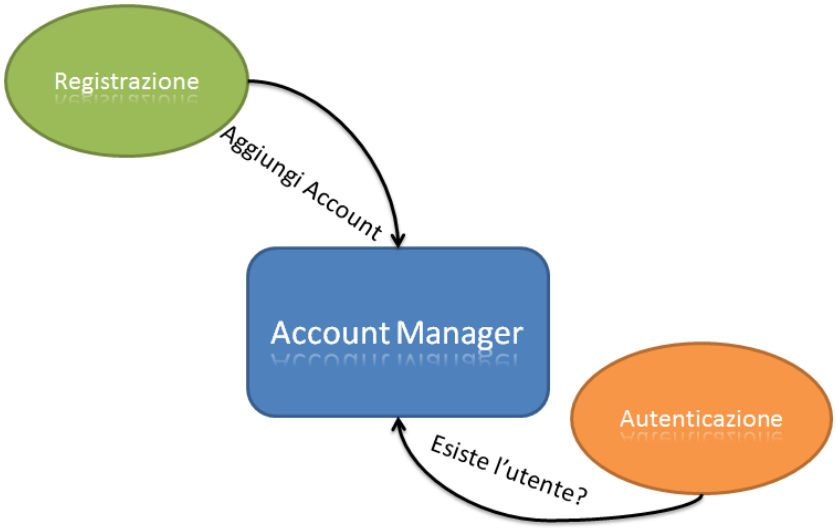

Figura 4.1: Utilizzo Account Manager

poter gestire Account differenti da quelli di Google è necessario implementare un'Activity specializzata, che realizzerà la vera logica di autenticazione. A questo punto è semplice poter integrare, mediante la Activity di autenticazione, differenti metodologie di riconoscimento dell'utente, dalla semplice password fino a parametri biometrici. Uno dei pochi punti dolenti, attualmente nella realizzazione di questo modulo, è il file di database locale. Ad

oggi tale file non è cifrato e la lettura è impedita da parte di applicazioni che non posseggano la stessa firma di quelle di registrazione. La scelta di non criptare il file lascia aperta una vulnerabilità verso tutti quegli utenti root che potrebbero essere attivati e che quindi potrebbero prelevare le credenziali di ogni account utente, semplicemente aprendo il file. Android sorvola su questa problematica, poiché spesso le applicazioni vengono strutturate con una logica di gestione account più centralizzata e remota, in mano ad un server. Tipicamente Account Manager è sfruttato mediante token di autorizzazione di tipo OAuth. L'impiego di tali, avrebbe però sviato completamente l'architettura logica, mettendo più in risalto la realizzazione server e impedendo quindi un approfondimento locale al sistema operativo mobile.
L'Account Manager svolge quindi un ruolo centrale nella gestione, reperimento, fornitura di credenziali all'interno di Android.

4.2 Client Android

Il client svolge una interazione semplice con il server, ma supportata da un sicuro accesso alle risorse remote mediante un layer di sicurezza. Volendo semplificare, una volta avviata l'applicazione '1stPass' l'utente è portato a inserire le proprie credenziali per poter accedere al servizio remoto. Tale client, una volta effettuata la richiesta di login, verifica se sul dispositivo l'utente è autorizzato (mediante l'Account Manager), stabilisce una connessione su socket sicura(una volta verificato il certificato del server), chiede l'accesso al server (il quale verifica l'autenticità dell'utente), risponde a un meccanismo semplice di sfida-risposta e infine ottiene la lista di servizi salvati. Una volta ricevuta la lista, potrà prendere visione dei dettagli, aggiornali, modificarli e aprire automaticamente la pagina web relativa al servizio. Mediante un semplice pulsante di sincronizzazione, potrà salvare le modifiche alla lista sul server remoto. La lista non sarà quindi mai salvata in locale su file, ma sempre e unicamente remotamente su un file cifrato. Ciò quindi impedirà qualsiasi vulnerabilità imputabile all'attivazione di utenti root.
La parte iniziale dell'interazione è visibile nella Figura 4.2. Successivamente

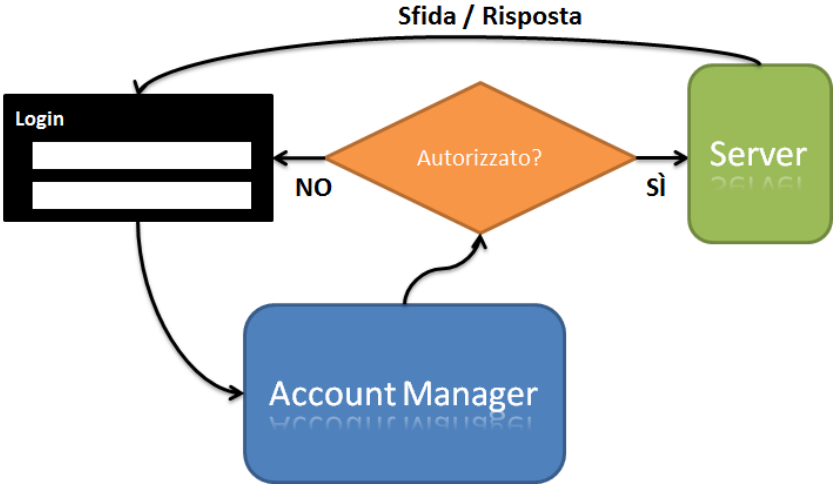

Figura 4.2: Interazione iniziale Client

il cliente per poter leggere le liste di password inviate dal server deve decifrarle tramite lo stesso numero randomico sfruttato mediante il meccanismo di sfida risposta. Quindi solo se l'utente verrà correttamente riconosciuto dal dispositivo, dal server potrà leggere il file ricevuto; dimostrando quindi la sua identità.

4.3 Server

Lato server l'interazione si espleta in maniera duale. Il server è in attesa del client, una volta ricevuta la richiesta verifica l'autenticità dell'utente e lo sfida (sempre su socket sicura), successivamente se è autorizzato apre i file contenenti i servizi salvati e, una volta cifrati, li invia al client android mediante una lista opportunamente creata. Il server oltre a verificare l'autenticità dell'utente, mediante il riconoscimento delle credenziali, si occuperà anche di registrare gli utenti attivi mediante un Monitor deputato a gestire la concorrenza. In questo modo si potrà evitare attacchi di replica, ipotizzando sempre un possibile intruso che sia riuscito a "bucare" il layer TLS.

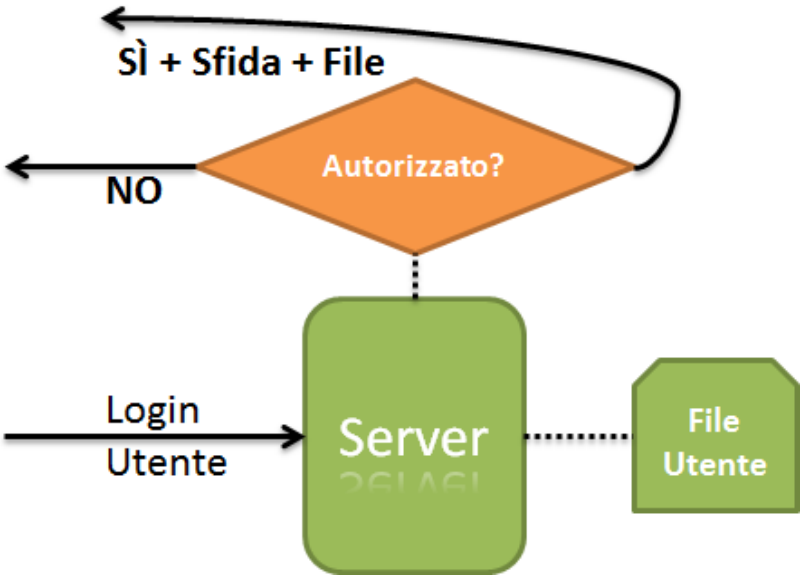

Figura 4.3: Interazione Server

Capitolo 5

Realizzazione Progetto

Di seguito vedremo come è stata affrontata la realizzazione del Progetto e come sono state formulate le interazioni tra i vari componenti.

5.1 TLS e certificati

A supporto dell'interazione tra Client e Server, si è deciso di sfruttare socket sicure. Transport Layer Security (TLS) è stato il protocollo di cifratura utilizzato per la comunicazione.
Tale meccanismo si basa principalmente sull'utilizzo di due metodologie di cifratura: una iniziale asimmetrica, durante la fase di autenticazione, e una simmetrica durante l'efettiva comunicazione. Al fine di poter usufruire di un supporto sicuro, i due intermediari devono effettuare un breve protocollo di handshake (con C si intende il Cliente, mentre con S il Server):

1. C invia a S la propria versione TLS, le impostazioni di crittografia, dati specifici di sessione e altre informazioni utili alla comunicazione;

2. S in risposta a C, invia la propria versione TLS, le impostazioni di crittografia, dati specifici di sessione e altre informazioni utili alla co-

municazione. In questo momento S invia anche il proprio certificato (chiave pubblica);

3. C utilizza le informazioni ricevute per verificare l'autenticità di S. Se S non è autorizzato, C è avvisato e a questo punto la connessione sicura non può avere luogo. Se S invece può essere autenticato, C procede;

4. C, mediante tutti i dati scambiati sinora dall'handshake (con collaborazione di S a seconda del cifrario in uso), genera una prima chiave simmetrica "pre-master" per la sessione, la cifra mediante la chiave pubblica di S (ottenuta al punto 2) e la invia a S;

5. S, mediante la propria chiave privata, decifra la chiave simmetrica "pre-master" e, attraverso una serie di procedure ben definite (nello stesso momento le stesse che C sta eseguendo), genera a partire dalla "pre-master" la vera e propria chiave "master";

6. Entrambi C e S, utilizzano la chiave "master" per generare le chiavi di sessione, nuovamente chiavi simmetriche usate per cifrare e decifrare le informazioni di sessione scambiate e per verificare l'integrità;

7. C invia un messaggio a S informandolo che i messaggi futuri saranno cifrati e conclude l'handshake inviando un messaggio cifrato specifico da protocollo;

8. S invia un messaggio a C informandolo che i messaggi futuri saranno cifrati e conclude l'handshake inviando un messaggio cifrato specifico da protocollo.

Da questo momento in poi la sessione può considerarsi sicura.
Tipicamente il certificato (in cui è inclusa la chiave pubblica) del Server viene scaricato dal Client prima di instaurare la comunicazione. Tale certificato viene verificato tramite una Certificate Authority fidata (di norma il dispositivo ha una lista nativa di Authority fidate).
Nel progetto, tale parte non compare e di fatto sia il Client sia il Server posseggono già a priori un proprio Key e Trust Store in cui sono salvate le

chiavi. In questo modo tramite il TrustStore il Client Android può verificare la chiave pubblica inviata dal Server e valutarne l'autenticità, mentre il Server, tramite il proprio KeyStore, aver cognizione sia della propria chiave pubblica sia della chiave privata per poter decifrare i messaggi a lui indirizzati. Tramite tale meccanismo non è quindi necessaria alcuna autorità esterna che validi l'identità dei due intermediari.
Per realizzare i certificati si è proceduto in questo modo:

1. Generazione lato S, su PC, del certificato S e inserimento in un KeyStore privato;

2. Esportazione chiave pubblica S;

3. Importazione chiave pubblica S nel TrustStore privato di C su Android.

Che si traducono in queste sequenze di istruzioni (mediante il tool nativo java keytool e la libreria BouncyCastle per la generazioni di TrustStore di tipo BKS per Android):

1. Genera coppia chiavi privata e pubblica;

```
keytool -genkeypair -keystore storeServer -alias serverCertificate
```

2. Esporta la chiave pubblica dell'alias nel file server.crt;

```
keytool -export -keystore storeServer -alias serverCertificate -
    file server.crt
```

3. Importa la chiave pubblica nello store Android apposito di tipo BKS.

```
keytool -import -alias serverCertificate -file server.crt -keystore
     storeAndroid -storetype BKS -providerClass org.bouncycastle.
    jce.provider.BouncyCastleProvider -providerpath bcprov-jdk15on
    -148.jar
```

5.2 Supporto alla comunicazione

Il protocollo appena visto è supportato da una famiglia di socket, in Java, detta "SSLSocket". La creazione di una socket sicura richiede alcuni passi e, per questo motivo, sono state introdotte classi di supporto per semplificare la creazione e l'utilizzo.
Per il client Android, Figura 5.1, mostra la classi di supporto: Per poter

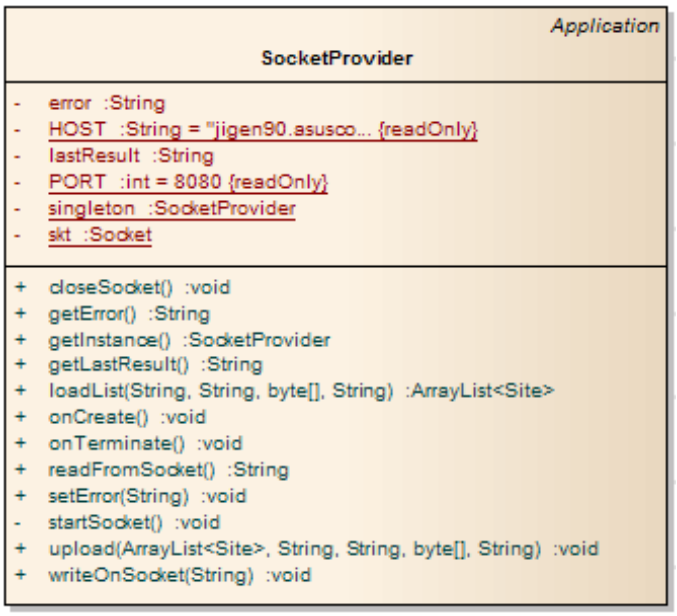

Figura 5.1: La classe SocketProvider

avviare una socket di questo tipo, si deve:

1. Caricare da file il TrustStore e immetterne la password relativa;

```
KeyStore keystore = KeyStore.getInstance("BKS");
keystore.load(this.getApplicationContext().getResources().
    openRawResource(R.raw.storeAndroid), passphrase);
```

2. Avviare un gestore chiavi;

```
KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance
    (KeyManagerFactory.getDefaultAlgorithm());
keyManagerFactory.init(keystore, passphrase);
```

3. Impostare il contesto SSL;

```
SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
```

4. Ottenere un KeyManager e un TrustManager;

```
KeyManager[] keyManagers = keyManagerFactory.getKeyManagers();
TrustManager[] trustManagers = new TrustManager[]{
  new X509TrustManager() {
    public java.security.cert.X509Certificate[] getAcceptedIssuers
        ()
    {
      return null;
    }
    public void checkClientTrusted(java.security.cert.
        X509Certificate[] certs, String authType)
    {
    }
    public void checkServerTrusted(java.security.cert.
        X509Certificate[] certs, String authType)
    {
    }
  }
};
```

5. Avviare il contesto SSL;

```
sslContext.init(keyManagers, trustManagers, new SecureRandom());
```

6. Ottenere la socket factory;

```
SSLSocketFactory sslSocketFactory = (SSLSocketFactory) sslContext.
    getSocketFactory();
```

7. Infine, ottenere la socket.

```
Socket skt = (SSLSocket) sslSocketFactory.createSocket(HOST, PORT);
```

Allo stesso modo lato server per avviare la stessa comunicazione. Di fondamentale importanza i file contenenti i certificati.

5.3 Gestione utente Android

Come si è già visto, un metodo efficace e corretto per poter gestire le credenziali utente e l'autenticità dello stesso è mediante il servizio Account Manager, nativo in Android. Volendo sfruttarlo con una tipologia di account, differente da quella Google, è quindi necessario l'impiego di una classe Account Authenticator.
Seguiamo gli step per poter utilizzare Account Manager, inserendo un Account ad-hoc:

1. Creare il nostro Authenticator il cuore della nostra operazione;
2. Creare l'activity responsabile della effettiva verifica dell'utente (a nostro totale piacimento) - qui l'utente inserirà le credenziali;
3. Creare il service, attraverso cui possiamo comunicare con l'Authenticator.

Vediamo i compiti dei vari componenti.
AccountAuthenticator: un modulo per gestire un tipo specifico di account. L'Account Manager trova l'AccountAuthenticator appropriato, interagisce e esegue tutte le operazioni specifiche per quel tipo di account. L'Authenticator sa quale activity mostrare all'utente per fargli inserire le credenziali e dove salvarle (nel nostro caso all'interno dello stesso Account Manager). Ciò può essere comune a molti servizi sotto un unico tipo di account; per esempio l'autenticatore di Google su Android autorizza GMail, insieme ad altri servizi quali Calendar, Drive.
AccountAuthenticatorActivity: classe di base per i servizi di registrazione e accesso, chiamata dall'autenticatore quando l'utente ha bisogno di identificarsi. Tale activity si occupa della vera logica di autenticazione, per poi salvare i dati all'interno dell'Account Manager.
Volendo integrare completamente il nostro account di tipo "1stPass" all'interno degli account disponibili su Android, si è proceduto alla creazione del service relativo. Una volta definito, automaticamente Android nel momento in cui si selezionano gli account disponibili, trova la tipologia inserita e

tramite il servizio richiama l'AccountAuthenticator relativo, come si evince dalla Figura 5.2. I passi nel codice:

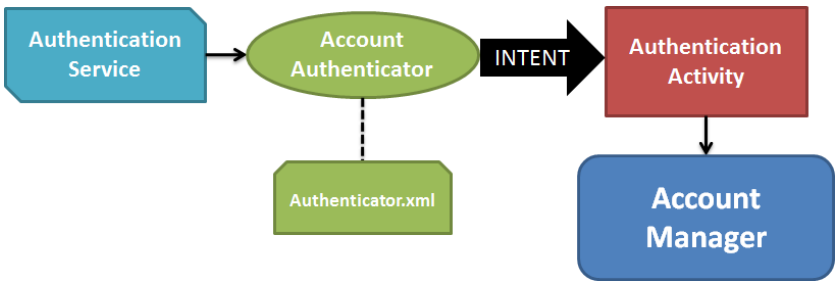

Figura 5.2: Funzionamento meccanismo Account Authenticator

1. Service relativo;

```
public class AuthenticationService extends Service {

  @Override
  public IBinder onBind(Intent arg0) {
    // TODO Auto-generated method stub
    return new AccountAuthenticator(this).getIBinder();
  }

}
```

2. AccountAuthenticator;

```
public class AccountAuthenticator extends
    AbstractAccountAuthenticator {
  @Override
  public Bundle addAccount(AccountAuthenticatorResponse response,
      String accountType,
      String authTokenType, String[] requiredFeatures, Bundle
          options)
          throws NetworkErrorException {
    // TODO Auto-generated method stub
    final Bundle result;
    final Intent intent;
    intent = new Intent(this.myCtx, AuthenticationActivity.class);
    intent.putExtra(AccountManager.KEY_ACCOUNT_TYPE, accountType);
    intent.putExtra(AccountManager.
        KEY_ACCOUNT_AUTHENTICATOR_RESPONSE, response);

    result = new Bundle();
    result.putParcelable(AccountManager.KEY_INTENT, intent);
```

```
    return result;
  }
}
```

3. L'activity che effettua la logica.

```
public class AuthenticationActivity extends
    AccountAuthenticatorActivity {
  public void onClickConfirm(View arg0){
    AccountManager acMan = AccountManager.get(this);
    final Account account = new Account(user, accountType);
    ByteObj obj = EncDecSocket.hash(password);
    Bundle bundle = new Bundle();
    bundle.putString("#S41T", EncDec.base64EncodetoString(obj.
        getIVParameterSpec()));
    acMan.addAccountExplicitly(account, EncDec.base64EncodetoString
        (obj.getEncriptedData()), bundle);
  }
}
```

Come si può notare dagli estratti del codice, tramite il metodo "addAccountExplicitly" viene inserito il nuovo account creato e l'hash della password dell'utente. Tutto ciò per evitare la possibilità che un utente root possa in qualche modo leggere il database dell'Account Manager e quindi potersi autenticare con le credenziali dell'utente. L'hash viene effettuato con "PBKDF2WithHmacSHA1" a 1024 iterazioni e con chiave di lunghezza 128 byte. Allo stesso modo quando l'applicazione dovrà inizialmente avviarsi e verificare l'utente, tramite l'Account Manager, otterrà l'hash relativo e ricalcolerà l'hash risultante dall'input utente; in questo modo la password non viene mai salvata in chiaro, ma il tutto si riduce ad un confronto tra stringhe prive di senso.

Una precisazione occorre. L'inserimento di account all'interno dell Account Manager può essere effettuato da tutte le applicazioni, basta che esse dichiarino il permesso (nel manifest) relativo. Tuttavia l'estrazione di dettagli dall'Account Manager, a partire dal nome dell'account, è vincolata ad una regola ben precisa: possono richiedere informazioni personali, tra cui anche l'hash della password, solo quelle applicazioni che hanno lo stesso UID dell'AccountAuthenticator. Quindi l'activity di login, che verificherà l'utente,

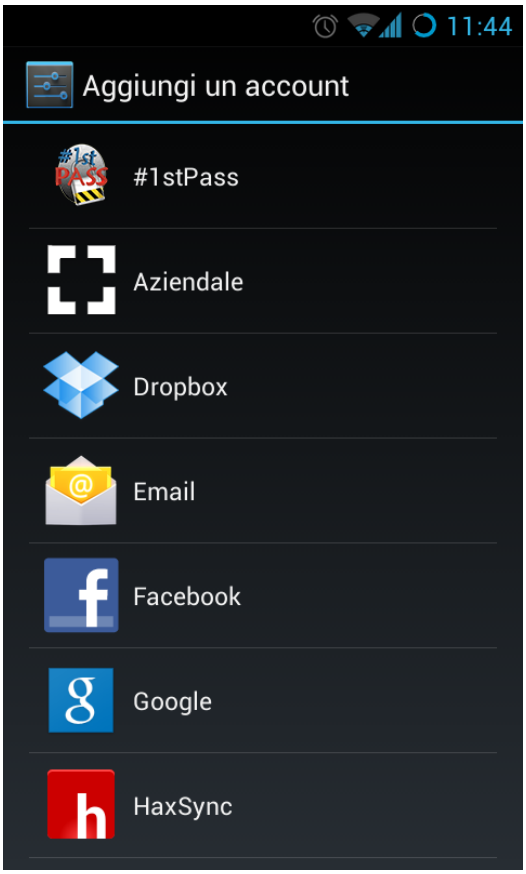

Figura 5.3: Aggiunta account

dovrà possedere lo stesso UID (insito nel manifest) per poter richiedere i dettagli.
Android, come si è già visto, permette UID unici, quindi per ogni dispositivo può esistere un solo UID di quel tipo, di conseguenza solo le applicazioni autorizzate di tipo 1stPass potranno accedere a quella tipologia di account nell'Account Manager.

Vediamo i passi che l'utente deve affrontare per essere registrato sul dispostivo:

1. Inserimento nuovo account mediante le impostazioni di Android, Figura 5.3;

2. Compilazione form generato dall'activity relativa (e salvataggio all'in-

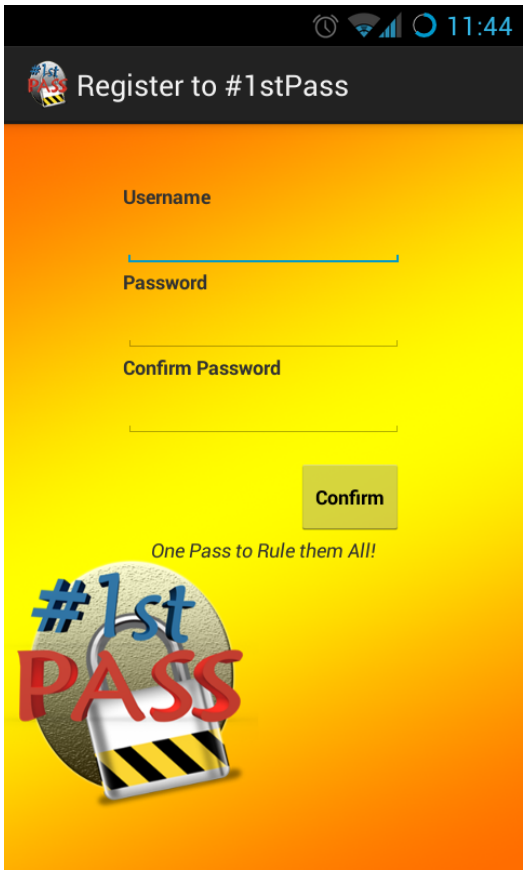

Figura 5.4: Compilazione form

terno dell'Account Manager), Figura 5.4.

In maniera del tutto trasparente, nel momento in cui l'utente si registra sul dispositivo in maniera corretta, avviene contemporaneamente la registrazione dell'utente sul server. Tramite connessione sicura, come già visto, le credenziali utente vengono passate in maniera cifrata al server, il quale inserisce l'utente nella propria lista di utenti autorizzati.

Nella Figura 5.5 notiamo le principali classi utilizzate durante la procedura di registrazione locale e remota. "HashChecker" e "ConnectingClassForRegister" racchiudono metodi e task asincroni per la registrazione locale e calcolo hash e per la connessione remota e comunicazione.

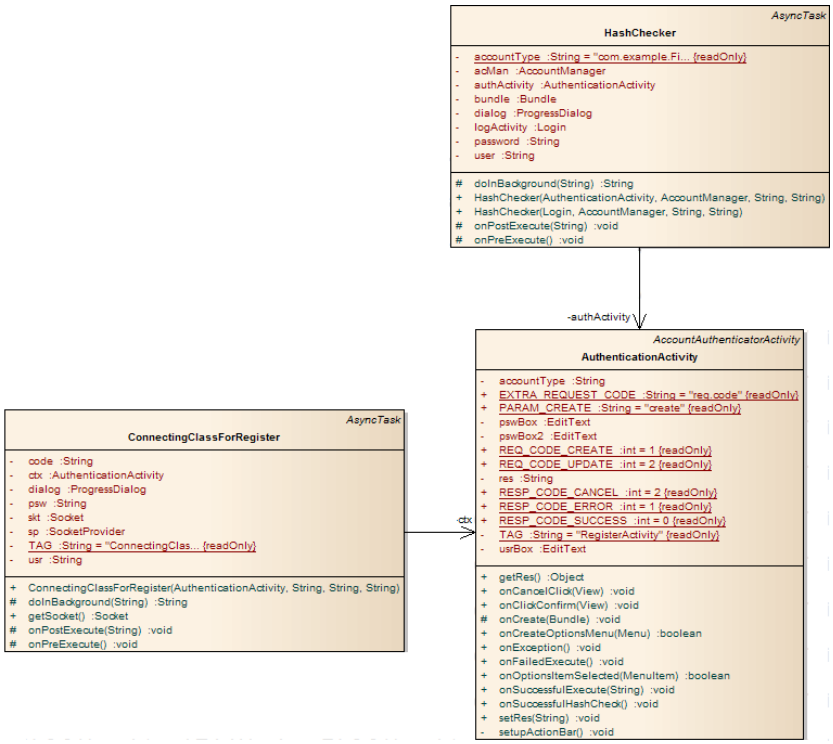

Figura 5.5: Classi per la registrazione

5.4 Client Android

La parte client è principalmente suddivisa in due componenti principali: quello di Login e quello effettivo di interazione con l'utente, 1stPass.
La parte di login è stata strutturata in questo modo, vedi Figura5.6.
L'activity di Login si avvia e si predispone all'interazione. Quando l'utente preme sul pulsante di "Login" scatena una serie di passi seguendo queste istruzioni:

1. Verifica della presenza dell'utente sul dispositivo, mediante una semplice equivalenza sul nome;

```
boolean OK = false;
Account[] accounts = acMan.getAccountsByType(accountType);
Account result = null;
for(Account a : accounts){
  if(a.name.equals(user)){
    OK = true;
```

```
    result = a;
    break;
  }
}
```

2. Verifica che gli hash (salvato e calcolato) siano identici;

```
if(OK){
  String storedHash = acMan.getPassword(result); //qui abbiamo
      salvato l'HASH
  String storedSalt = acMan.getUserData(result, "#S41T");
  String calculatedHash = EncDec.hash(password, EncDec.
      base64DecodeFromString(storedSalt));
    if(!calculatedHash.equals(storedHash)){
      return "Username or Password wrong!";
    }
    return "Logged in";
}
else
  return "User doesn't exist!";
```

3. Se abbiamo una risposta affermativa, si inizia la comunicazione con il server;

```
if(result.equals("Logged in")){
    new ConnectingClassForLogin(this, this.usrBox.getText().
        toString(), this.pswBox.getText().toString(),"#L#\n").
        execute("");
}
```

4. Se anche il server risponde correttamente, allora passiamo all'avvio dell'activity 1stPass.

```
Intent i = new Intent(this,FirstPass.class);
i.putExtra("usr", this.usrBox.getText().toString()); //per frase di
     benvenuto
i.putExtra("salt", bytes); //per ulteriore cifratura della lista di
     password, in caso di attacco BEAST
i.putExtra("pass", this.pswBox.getText().toString()); //password
    per sbloccare il file-entry salvato sul server

this.startActivity(i); //avvio activity
```

Da notare come la password non venga mai salvata in nessuna situazione e viaggi su rete doppiamente cifrata. L'utente una volta avviata l'applicazio-

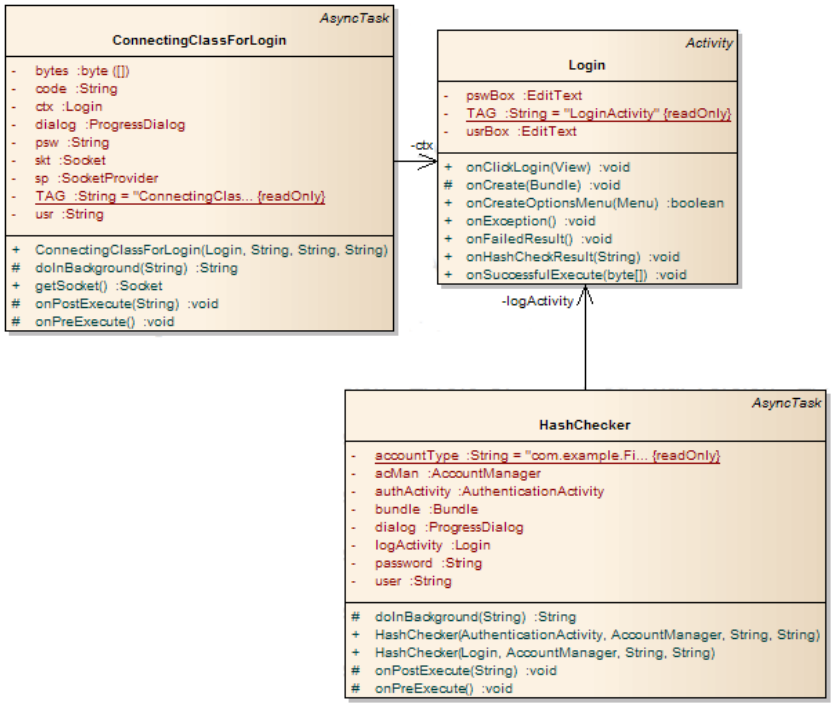

Figura 5.6: Classi per l'autenticazione

ne, riceve questa finestra di login, che gli permetterà di accedere al sistema solo se è già stato autorizzato dall'Account Manager, Figura 5.7. Ricevuto l'intent, l'activity 1stPas può quindi avviarsi. Qui è presente la logica di interazione con il server per lo scambio della lista di password, la cifratura dei contenuti e l'interazione con l'utente.
La suddivisione dei componenti, in Figura 5.8. "ConnectedClass" svolge la logica di interazione con il server, mentre "Site" è una classe di supporto all'interazione.
Qui l'interazione con l'utente, Figura 5.9. Nel momento in cui il servizio è inserito, è possibile prenderne visione. Tramite il tasto opzioni la password del servizio può essere resa visibile o copiabile. Nel momento in cui si seleziona "WEB", si verrà indirizzati direttamente all'indirizzo web salvato, Figura 5.10.

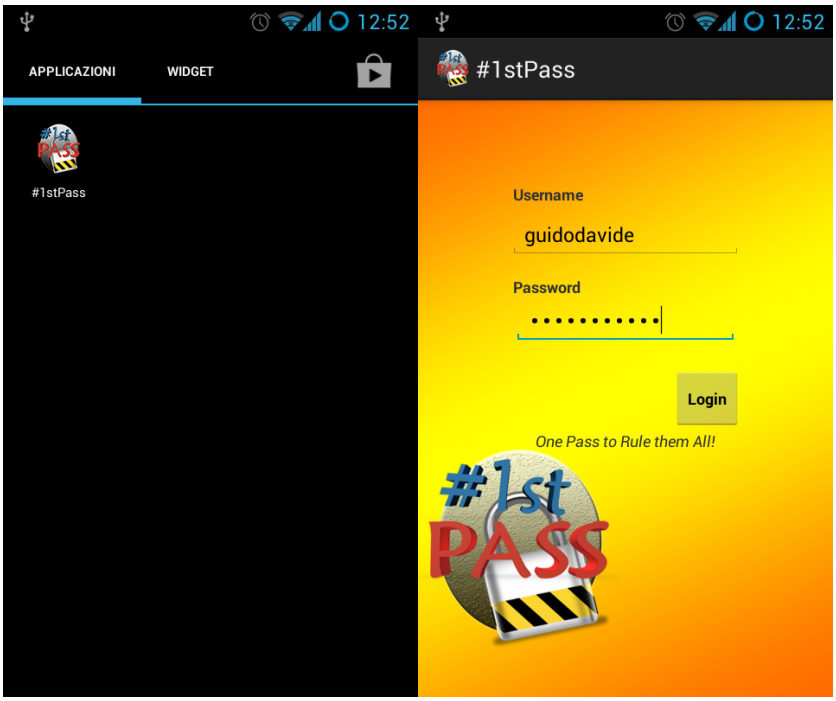

Figura 5.7: Sequenza di login

5.5 Manifest applicazione

Il manifest dell'applicazione, integrato nel pacchetto di installazione Apk automaticamente generato da Eclipse, permette di avere una visione globale della struttura del programma. Può considerarsi un macro riassunto dei vari componenti interagenti.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.firstpass"
    android:sharedUserId="com.firstpass.user"
    android:versionCode="1"
    android:versionName="1.1" >

    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.
        ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

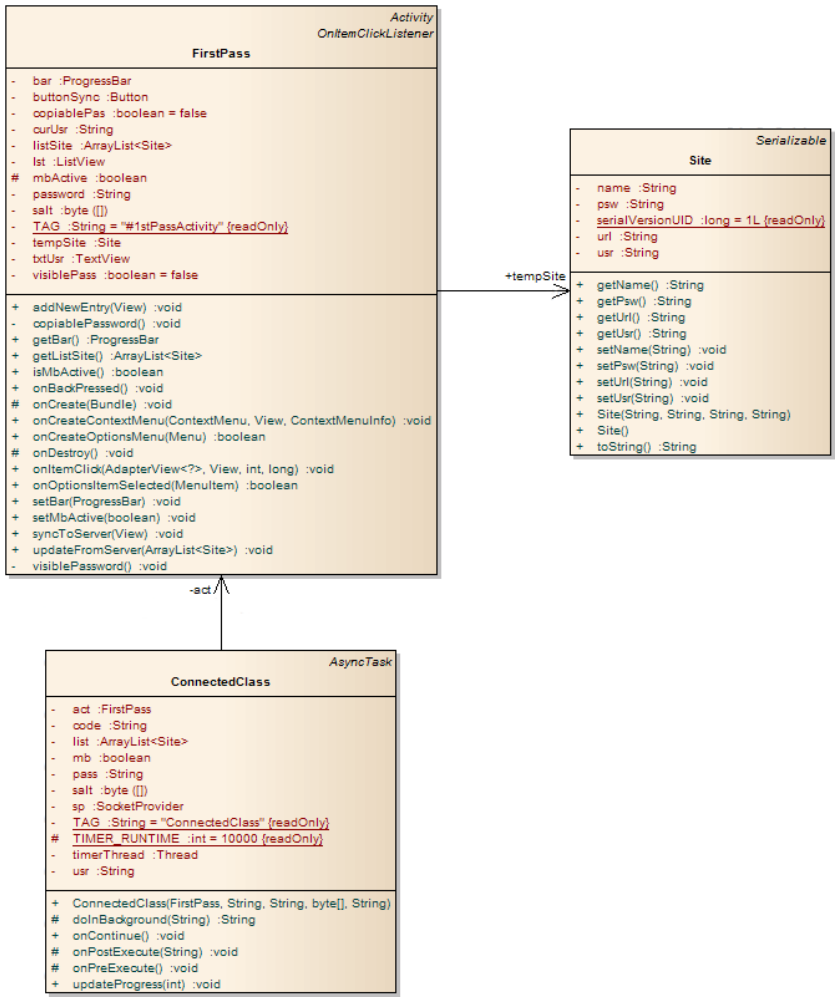

Figura 5.8: Classi 1stPass

```
  <uses-permission android:name="android.permission.USE_CREDENTIALS" />
  <uses-permission android:name="android.permission.
      AUTHENTICATE_ACCOUNTS" />

  <permission
      android:name="com.firstpass.permission"
      android:description="@string/descriptionPerm"
      android:label="Read incoming email"
      android:permissionGroup="android.permission-group.PERSONAL_INFO"
      android:protectionLevel="normal" />

<uses-permission android:name="com.firstpass.permission" />
  <application
      android:name="com.firstpass.model.SocketProvider"
```

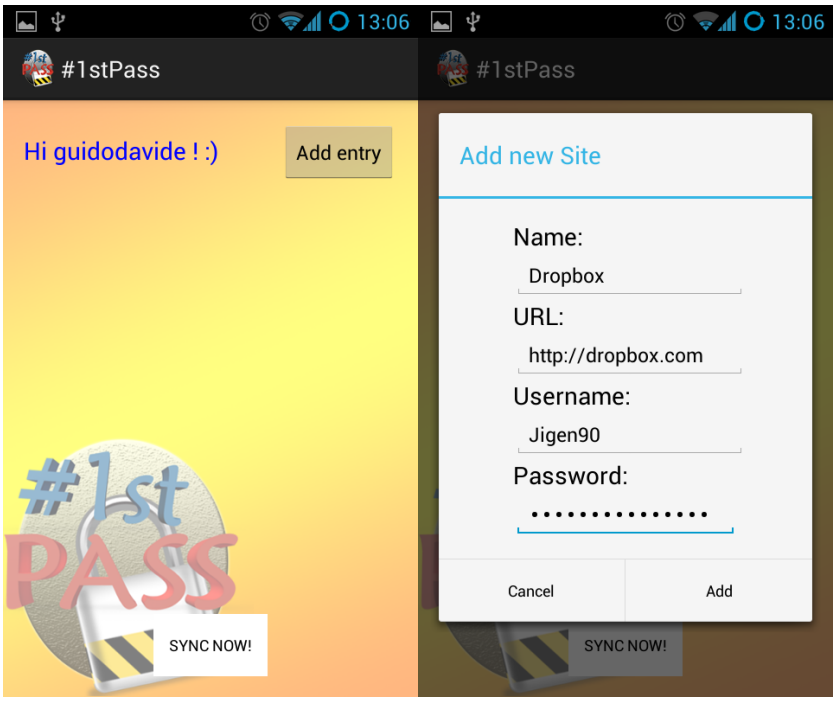

Figura 5.9: Inserimento nuova entry

```
android:allowBackup="true"
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
<service
    android:name="com.firstpass.services.AuthenticationService"
    android:exported="false"
    android:permission="com.firstpass.permission" >
    <intent-filter>
        <action android:name="android.accounts.
            AccountAuthenticator" >
        </action>
    </intent-filter>

    <meta-data
        android:name="android.accounts.AccountAuthenticator"
        android:resource="@xml/authenticator" >
    </meta-data>
</service>

<activity
    android:name="com.firstpass.activities.AuthenticationActivity
        "
    android:label="@string/title_activity_authentication"
```

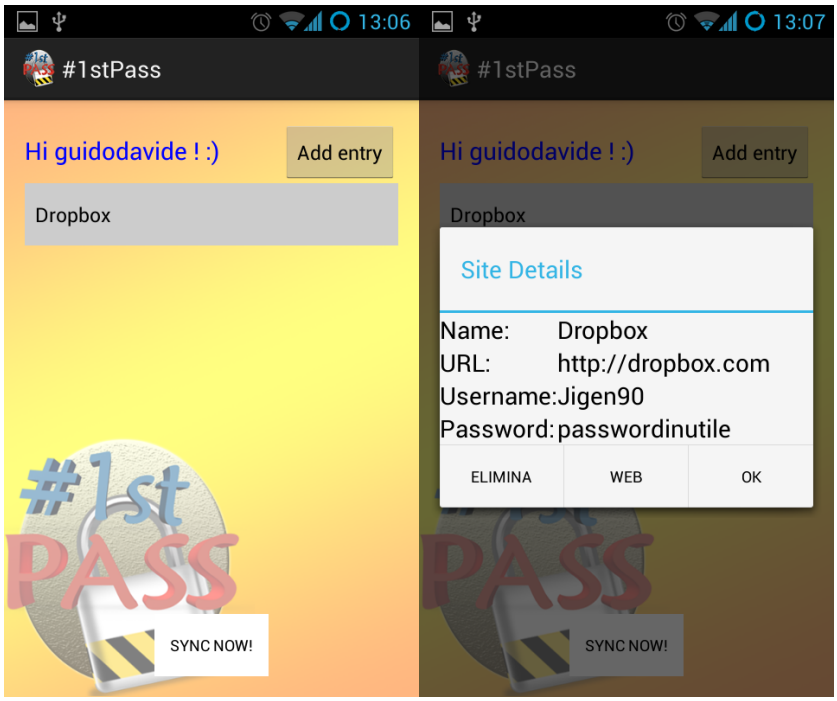

Figura 5.10: Finestra servizio

```
    android:permission="com.firstpass.permission"
    android:process="com.firstpass.process" >
</activity>
<activity
    android:name="com.firstpass.activities.Login"
    android:label="@string/app_name"
    android:process="com.firstpass.process" >
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.example.firstpass.Login" />

    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER"
            />
    </intent-filter>
</activity>
<activity
    android:name="com.firstpass.activities.FirstPass"
    android:label="@string/title_activity_first_pass"
    android:permission="com.firstpass.permission"
    android:process="com.firstpass.process" >
</activity>
```

```
        </application>

    </manifest>
```

5.6 Server Java

Per quanto riguarda la realizzazione del servizio ricevente le richieste da parte degli utenti, possiamo distinguere fin dall'inizio le varie parti fondamentali.
Il tutto è strutturato principalmente su una classe principale che accetta le richieste detta "AcceptServer" e lancia thread specifici per ogni cliente.
Ad ogni cliente viene legato immediatamente un "HandlerThread" il quale si occuperà di tutte le interazioni e delle fasi di vita dell'applicazione lato cliente.
In questo modo si rende scalabile l'architettura lato server e si rende possibile l'interazione contemporanea di più clienti al servizio.
Le classi principali sono in Figura 5.11 Ad ogni interazione viene automa-

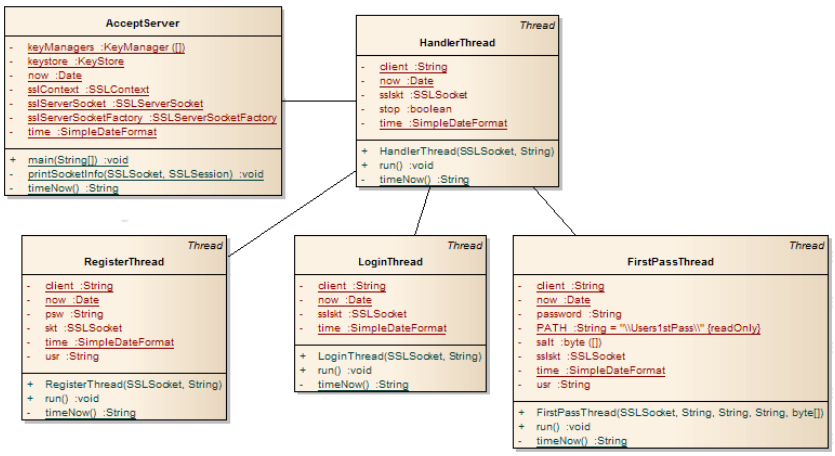

Figura 5.11: Classi server

ticamente associato un Handler, il quale in base alle differenti richieste del client decide se attivare un thread di tipo Register, Login o FirstPass.
Per evitare che qualcuno possa replicare i messaggi del client e quindi loggarsi nuovamente e replicare i messaggi (nell'ipotesi che con un attacco es. Beast

riesca a rompere la sicurezza TLS), è stato introdotto un registro utenti attivi, gestito da un monitor centralizzato.
Una volta confermata l'autorizzazione al cliente, il server deve decifrare il file relativo ai dati utente (cifrato mediante "PBKDF2WithHmacSHA1" e "AES" a 65536 iterazioni e chiave da 256 byte) con la password utente e inviare su file i dettagli dei servizi cifrandoli nuovamente con una chiave randomica.
Ad ogni richiesta di Sync, il cliente sincronizza i dettagli inseriti sul dispositivo con il server, in modo che quest'ultimo si occupi del salvataggio.
Nel log di seguito si può comprendere meglio i passi dell'interazione.

```
[2013/06/17-12:37 - AcceptServer] Avviato
[2013/06/17-12:37 - AcceptServer] Keystore inizializzato
[2013/06/17-12:37 - AcceptServer] Server Socket creata
[2013/06/17-12:37 - AcceptServer] Accettata richiesta
Socket class: class sun.security.ssl.SSLSocketImpl
   Remote address = /192.168.1.1
   Remote port = 39402
   Local socket address = /192.168.1.100:8080
   Local address = /192.168.1.100
   Local port = 8080
   Need client authentication = false
   Cipher suite = TLS_DHE_DSS_WITH_AES_128_CBC_SHA
   Protocol = TLSv1
[2013/06/17-13:06 - HandlerThread /192.168.1.1]Avviato
[2013/06/17-13:06 - HandlerThread /192.168.1.1]Richiesta Login
[2013/06/17-13:06 - HandlerThread /192.168.1.1]Attendo LoginThread
[2013/06/17-13:06 - LoginThread /192.168.1.1]Avviato
[2013/06/17-13:06 - LoginThread /192.168.1.1]Richiesta Login per
    guidodavide
[2013/06/17-13:06 - LoginThread /192.168.1.1]guidodavide entrato
[2013/06/17-13:06 - LoginThread /192.168.1.1]guidodavide è LOGGATO
[2013/06/17-13:06 - LoginThread /192.168.1.1]Attendo FirstPassThread
[2013/06/17-13:06 - FirstPass /192.168.1.1]Avviato per guidodavide
[2013/06/17-13:06 - FirstPass /192.168.1.1]Richiesta lista per
    guidodavide
[2013/06/17-13:06 - FirstPass /192.168.1.1]Invio lista per guidodavide
[2013/06/17-13:08 - LoginThread /192.168.1.1]FirstPassThread chiuso
[2013/06/17-13:08 - LoginThread /192.168.1.1]guidodavide è FUORI
[2013/06/17-13:08 - HandlerThread /192.168.1.1]LoginThread chiuso
[2013/06/17-13:09 - HandlerThread /192.168.1.1]Applicazione Client
    Terminata
[2013/06/17-13:09 - HandlerThread /192.168.1.1]si chiude
```

La verifica dell'utenza viene effettuata a partire dalla lettura di un file, contenente per ogni utente: l'identificativo, il salt sfruttato e l'hash della password. Con la stessa metodologia del client, il server verificherà l'autenticità del client rieseguendo l'hash della password trasmessa e confrontandola con quella già salvata.
Vediamo uno stralcio del metodo di verifica:

```
LoginRegisterUtils.getInstance();
Iterator<User> it = userList.iterator();
while(it.hasNext()){
    User temp = it.next();
    if(temp.getName().equals(user)){
      String hashCalculated=hash("sha", psw, temp.getSalt());
      if(hashCalculated.equals(temp.getHash()))
        return true;
      else
        return false;
    }
}
return false;
```

Capitolo 6

Conclusioni e Risultati

Mediante l'attivit progettuale e la relativa progettazione, si sono conseguiti differenti risultati.
Personalmente ho potuto apprendere meglio le tecnologie alla base del sistema operativo mobile Android e il loro impiego. Le possibilit fornite da questo sistema, permettono agli sviluppatori ampi orizzonti e idee. Non nego di essermi divertito, nel cercare di ottimizzare e perfezionare il pi possibile il progetto. Tra le varie competenze acquisite:

- Sistema di Accounting in Android, come l'utente può essere autenticato;
- Gestione delle connessioni e realizzazione di Socket sicure;
- Gestione della concorrenza e accessi contemporanei;
- Prevenzione e corrette tecniche per impedire possibili attacchi;
- Ottimizzazione delle risorse a disposizione.

Grazie al progetto stato possibile anche approfondire le vulnerabilit dei sistemi di gestione account e come possibile prevenire alcuni attacchi.
Sicuramente sono presenti margini di miglioramento e possibili sviluppi futuri, ne vorrei elencare alcuni:

- Gestione degli utenti Android tramite rilascio token OAuth da parte del server;
- Ridurre scritture su file lato server;
- Ottimizzare la gestione dei file utente, lato server, introducendo un database di supporto;
- Studiare il carico e ottimizzazione dovute a pi accessi contemporanei (a titolo di prova si provato solo un accesso simultaneo alle risorse remote da parte di due soli dispositivi);
- Studio pi preciso e realizzazione dell'UI in Android;
- Approfondimento dei possibili attacchi su TLSv1.0, quali BEAST.