

# Università degli Studi di Genova

Scuola di Scienze Matematiche Fisiche e Naturali

Dipartimento di Informatica Bioingegneria Robotica e Ingegneria dei Sistemi

Corso di Laurea Magistrale in Informatica



*Anno Accademico 2013/2014*

## Certificazione dei dispositivi Android in ambito BYOD

*Candidato*

Dott. Paolo Macco

*Relatori*

Dott. Giovanni Lagorio

Dott. Alessio Merlo

*Correlatore*

Prof. Giorgio Delzanno



# Sommario

Sommario .....	3
Ringraziamenti .....	7
1. Introduzione .....	9
2. Il paradigma BYOD .....	11
2.1. Panoramica dell'offerta attuale.....	11
2.1.1. Soluzioni MDM .....	12
2.1.2. Soluzioni Dual Persona .....	13
2.1.3. Soluzioni Remote Workplace .....	13
2.1.4. Soluzioni Virtualization Platform .....	14
2.2. Samsung KNOX .....	14
2.2.1. TIMA.....	14
2.2.2. Secure Boot .....	15
2.2.3. Trusted Boot .....	16
2.2.4. Security Enhancements (SE) .....	16
2.2.5. Attestation.....	16
2.2.6. On-Device Encryption (ODE) .....	17
2.2.7. MDM .....	17
2.2.8. Applicazioni e informazioni protette .....	18
2.3. Android Lollipop.....	19
2.4. Limitazioni dell'offerta attuale .....	20
3. L'approccio BYOD-Cert .....	21
3.1. Gli attori.....	22
3.2. Casi d'uso.....	22
3.2.1. Registrazione di un nuovo client (First app execution) .....	22
3.2.2. Login .....	23
3.2.3. Richiesta di certificazione del dispositivo .....	25
3.2.4. Installazione di una nuova applicazione .....	27
3.2.5. Utente invalida l'attuale configurazione certificata .....	29
3.2.6. Disinstallazione di un'applicazione .....	29
3.2.7. Aggiornamento di un'applicazione .....	30
3.2.8. Revoca di un certificato.....	30

## Sommario

3.2.9.	Ingresso nella rete aziendale .....	30
3.2.10.	Permanenza del dispositivo in azienda .....	31
3.2.11.	Disinstallazione del Client .....	32
3.2.12.	Rimozione di un utente dall'azienda .....	32
3.2.13.	Richiesta di certificazione per una policy .....	32
3.3.	Certificati di conformità .....	32
4.	WPA2 e gestione delle policy .....	33
4.1.	Il protocollo WPA2 .....	33
4.2.	Il protocollo EAP .....	33
4.2.1.	EAP-TLS .....	34
4.2.2.	EAP-TTLS .....	34
4.2.3.	PEAP .....	34
4.3.	Il protocollo RADIUS .....	35
4.4.	Generazione di certificati X.509v3 .....	36
4.5.	Sperimentazione con un server FreeRADIUS .....	37
5.	Remote Attestation in Android .....	43
5.1.	Il TEE .....	43
5.2.	Remote attestation in Android .....	45
6.	L'applicazione client di BYOD-Cert .....	47
6.1.	Raccolta informazioni sulle applicazioni installate .....	47
6.2.	Gestione certificati su dispositivo Android .....	49
6.3.	Monitorare le (dis)installazioni .....	51
6.4.	Verifica dinamica .....	51
7.	Conclusioni e lavoro futuro .....	55
A.	Appendice tecnica .....	57
A.1.	WPA2 .....	57
A.1.1.	Il Four-Way Handshake .....	57
A.1.2.	Il Group Key Handshake .....	58
A.2.	Il protocollo EAP .....	58
A.2.1.	EAP-SIM .....	58
A.1.1.	EAP-AKA .....	58
A.1.2.	EAP-FAST .....	59
A.1.3.	EAP-PWD .....	59

## Sommario

B. Indice analitico.....	61
Indice delle figure .....	63
Bibliografia.....	65



# Ringraziamenti

In conclusione di questo mio percorso universitario, tengo particolarmente importante ringraziare tutti quelli che mi hanno permesso di giungere fino a qui e che mi hanno accompagnato in questi anni.

Ringrazio innanzitutto la mia famiglia: i miei genitori, mio fratello Andrea, i miei nonni e tutti i miei parenti che mi hanno dato la possibilità di giungere fino a qui e che mi hanno sostenuto con il loro affetto e il loro amore.

Ringrazio quindi i miei relatori, Giovanni Lagorio e Alessio Merlo, che, con pazienza, in questi mesi mi hanno aiutato a realizzare questo lavoro; Luca Verderame e tutte le persone che mi hanno dato materiale e consigli per questa tesi. Voglio anche ringraziare tutti i docenti che in questi anni mi hanno fornito le conoscenze necessarie per compiere questo lavoro e tutti quelli futuri.

Tengo particolarmente a ringraziare Simone Ponte, fidato, paziente e assennato compagno, e Francesco Valli: due sinceri amici sempre presenti che più di tutti in questi mesi mi sono stati vicini e che mi hanno dato la possibilità di condividere con loro innumerevoli bellissime esperienze.

Grazie anche a tutti i miei compagni di università e di studi di questi anni: Veronica, Alessio, Benedetta, Camilla, Damiano, Davide, Emanuele, Federico, Francesca, Francesco, Gianluca, Giovanni, Lara, Laura, Luca, Marco, Mattia, Niccolò, Nikolas, Paolo, Simone, ...

Ringrazio quindi tutti i miei compagni scout con cui negli anni ho percorso molta strada.

A tutti gli amici con cui ho avuto modo di condividere numerosi e bei momenti, anche se qui non citati personalmente, va il mio non meno sentito grazie.





# 1. Introduzione

BYOD, acronimo di *Bring Your Own Device*, è un paradigma adottato da un numero crescente di aziende che consente ai propri impiegati di utilizzare propri dispositivi mobili sul luogo di lavoro anche per accedere ad applicazioni e informazioni riservate dell'azienda.

Le soluzioni attualmente proposte sul mercato richiedono di verificare la conformità alle politiche aziendali ogni volta che il dispositivo si connette alla rete aziendale (policy). Tuttavia, capita sempre più spesso che i lavoratori debbano muoversi di azienda in azienda e quindi debbano nuovamente verificare il proprio dispositivo. Tra l'altro, se le aziende utilizzano soluzioni BYOD diverse, questo potrebbe richiedere un ripristino totale del dispositivo alle condizioni di fabbrica (*factory reset*).

In questa tesi proponiamo un approccio che permette di certificare un dispositivo Android (*unrooted*) rispetto a delle politiche BYOD, in modo tale che se più aziende condividono le medesime politiche (o sottoinsiemi di esse) il dispositivo non debba esser verificato ogni volta.

Questo lavoro si pone all'interno di un'attività di ricerca già in atto, che prevede un approccio client-server. Il server, tramite tecniche di analisi statica (sul *control-flow graph* estratto dal codice binario), può verificare se un certo insieme di applicazioni rispetta, o meno, una politica BYOD. Nel caso la rispetti, può emettere un certificato che il dispositivo può usare per provare la sua conformità quando si collega alla rete aziendale. In caso non si riesca a valutare completamente, tramite l'analisi statica, la conformità di certe applicazioni, queste possono essere instrumentate (lato client, come dettaglieremo nel capitolo 6) in modo da essere controllate dinamicamente.

Questa tesi si focalizza sul lato client del sistema: saranno considerati i requisiti hardware e software necessari per il dispositivo da certificare, le informazioni necessarie per certificare la conformità di un dispositivo a una certa politica e i meccanismi per permettere solo ai dispositivi correttamente certificati di connettersi alla rete aziendale.

Il capitolo 2 si focalizzerà sul concetto di BYOD, presentando le quattro principali categorie di soluzioni attualmente offerte sul mercato e analizzando in particolare Samsung KNOX e le recenti innovazioni messe a disposizione da Android Lollipop.

Nel capitolo 3 verrà presentata la soluzione proposta in questa tesi, BYOD-Cert, illustrandone i diversi casi d'uso e il contenuto dei certificati di conformità.

Poiché le aziende fanno spesso ricorso al protocollo WPA2-Enterprise per consentire l'accesso ai propri dipendenti alla rete aziendale, poiché tale protocollo consente al client di conoscere il server e viceversa, in questo lavoro si è scelto di estendere tale approccio. In particolare, i certificati utilizzati permettono di verificare non solo l'identità del client, ma anche la sua conformità alle policy richieste dall'azienda.

## Introduzione

Nel capitolo 4 sarà dapprima illustrato il protocollo WPA2 e il suo funzionamento, quindi, verrà mostrato come generare certificati X.509v3 con estensioni custom, ovvero che contengano l'informazione delle policy per cui il dispositivo è conforme. Infine, verrà descritta una configurazione del server FreeRADIUS che permette di raggiungere i nostri obiettivi.

Nel capitolo 5 ci si soffermerà su alcune caratteristiche hardware e software che il dispositivo mobile deve avere affinché possa esser garantita l'autenticità dell'applicazione client durante la comunicazione col server BYOD.

Le caratteristiche dell'applicazione client sono descritte nel capitolo 6 dove, in particolare, verrà spiegato come recuperare la configurazione corrente del dispositivo, come gestire lato client i certificati di conformità e come intercettare l'installazione/rimozione di applicazioni. Infine, considereremo come aggiungere dei controlli dinamici, tramite strumentazione, per le applicazioni di cui non può essere completamente determinata staticamente la conformità rispetto alle policy volute.

Nel capitolo 7, saranno tratte le conclusioni, discussi i limiti del prototipo realizzato e delineati i possibili sviluppi futuri.

Sono presenti, infine, un'appendice tecnica in cui vengono dettagliati ulteriormente i protocolli WPA2 e EAP, e un indice analitico.

## 2. Il paradigma BYOD

Con il termine BYOD ci si riferisce alla politica, adottata da un numero crescente di aziende, di permettere ai propri impiegati di utilizzare i propri dispositivi mobili (ad esempio laptop, tablet, smartphone, ...) sul luogo di lavoro anche per accedere ad applicazioni e informazioni riservate dell'azienda [1].

Il termine BYOD è stato introdotto da Ballagas e altri per la prima volta nel 2005 [2], ma venne utilizzato per la prima volta in ambito aziendale nel 2009 da parte di Intel, che riconobbe che un sempre maggior numero di propri impiegati utilizzava i propri dispositivi personali per accedere alla rete aziendale. Tuttavia, solamente nel 2011 il termine si fece strada anche fra i fornitori di sistemi informatici e rivenditori di sistemi software quali Unisys, VMWare e Citrix Systems [1].

Spesso, al fine di meglio gestire i dispositivi degli impiegati e di evitare che la sicurezza della rete aziendale non sia compromessa dall'utilizzo degli stessi, le aziende stabiliscono delle regole per l'uso di tali dispositivi che non sempre incontrano il favore degli impiegati in quanto talvolta appaiono eccessivamente restrittive.

Queste, ad esempio, possono prevedere che il dispositivo debba essere configurato con una password, che non possano esser installate certe applicazioni, che tutti i dati sul dispositivo siano criptati, che possa esser utilizzata solo la mail aziendale, etc... [3].

In sezione 2.1, saranno presentate alcune delle principali soluzioni attualmente offerte in tale ambito sul mercato. Le sezioni 2.2 e 2.3 si focalizzeranno su Samsung KNOX e Android Lollipop, mostrando in cosa esse differiscano da quella presentata in questo lavoro, dettagliata in sezione 2.4.

### 2.1. Panoramica dell'offerta attuale

Attualmente esistono numerose soluzioni in ambito BYOD, fornite da diverse compagnie, che si possono suddividere in quattro gruppi [4]:

- *Mobile Device Management (MDM)* [5]: è la tipologia di soluzione maggiormente adottata. Permette di monitorare e gestire in sicurezza i dispositivi degli impiegati (sia quelli personali, sia quelli forniti dall'azienda) che possono accedere ai dati aziendali da ogni posto. Le funzionalità dell'MDM spesso includono la distribuzione virtuale delle applicazioni e configurazioni specifiche per tutti i tipi di dispositivi mobili.
- *Dual Persona* [6]: questo tipo di soluzione fornisce le stesse sicurezze e gestibilità del MDM, avendo tuttavia il minimo impatto su privacy e usabilità dell'impiegato; infatti, le soluzioni Dual Persona separano i dati e le applicazioni personali dell'utente da quelle aziendali attraverso la creazione di uno spazio di lavoro dedicato a queste ultime. I dati personali dell'utente non sono così acceduti dal sistema aziendale; inoltre, l'impiegato può passare facilmente da un ambiente all'altro premendo semplicemente un pulsante.

- *Remote Workspace* [7]: questa soluzione elimina la necessità di mettere in sicurezza il dispositivo dell'impiegato, poiché i dati aziendali non vengono salvati sul dispositivo, bensì su uno spazio di lavoro virtuale che risiede su di una piattaforma sicura accessibile dai dispositivi mobili dell'utente che è quindi libero di scegliere in completa autonomia i dispositivi da utilizzare.
- *Virtualization Platform* [8]: con questo tipo di soluzioni viene installato, sul dispositivo dell'impiegato, un ambiente virtuale di lavoro sicuro che può essere monitorato dall'azienda e che non va a interferire coi dati personali dell'utente.

Nelle sezioni successive si cercheranno di illustrare le caratteristiche principali di alcune soluzioni BYOD per ognuno dei quattro gruppi.

### 2.1.1. Soluzioni MDM

**Profile Manager** [9] di Apple è un prodotto che supporta solo i dispositivi con sistema operativo Apple iOS o Mac OS X. Consente a un server remoto di controllare i dispositivi registrati e di comunicare loro dei profili di configurazione. Da remoto è possibile bloccare o formattare i dispositivi e anche installare delle applicazioni. I dispositivi client utilizzano un *passcode* che può essere eventualmente reimpostato dal server remoto in caso l'utente lo dimentichi.

**Endpoint Manager for Mobile Devices** [10] di IBM supporta i dispositivi con sistema operativo Apple, Windows, Android e Symbian e consente di monitorare in tempo reale tutti i dispositivi registrati (fino a 250000 per server). Da remoto è possibile bloccare o formattare i dispositivi, installare delle applicazioni e visualizzare l'elenco delle applicazioni installate. I dispositivi client utilizzano un *passcode* che può essere eventualmente reimpostato dal server remoto in caso l'utente lo dimentichi.

**BYODroid** [11] è ideato per i dispositivi Android ed è schematizzato in Figura 1. Esso è costituito da due componenti: *BYODroid Market* e *BYODroid Installer*. La prima permette all'utente di installare applicazioni sul dispositivo che rispettino le policy aziendali; la seconda, invece, sostituisce l'installer nativo di Android e analizza lo stato del dispositivo comunicandolo al *market*. Il market si preoccupa di mantenere il dispositivo conforme alle policy aziendali e permette quindi al dispositivo di connettersi alla rete aziendale.

## Il paradigma BYOD

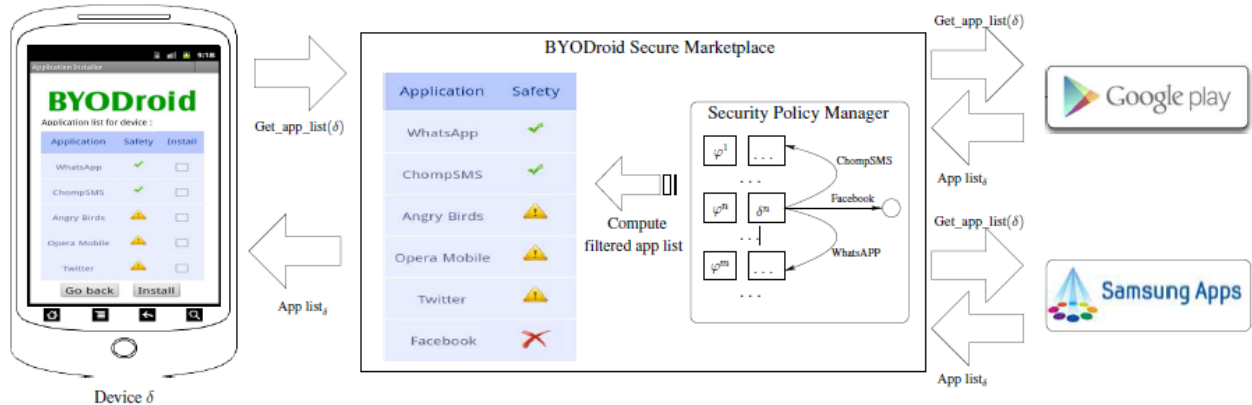


Figura 1: BYODroid

### 2.1.2. Soluzioni Dual Persona

**Samsung KNOX** [12] di Samsung supporta i dispositivi con sistema operativo Android. Esso crea una partizione sicura all'interno del dispositivo client, in cui possono esser salvati i file aziendali, all'esterno della quale vengono mantenuti i dati e le applicazioni personali dell'utente. Attraverso l'inserimento di un PIN, l'utente può entrare nella zona sicura del suo dispositivo e installare solamente applicazioni fidate, distribuite attraverso un apposito market. Per assicurare la sicurezza dei dati della partizione aziendale, Samsung si appoggia a diverse caratteristiche hardware del dispositivo, ognuna delle quali mirata a risolvere alcuni problemi di sicurezza. Tali caratteristiche saranno meglio approfondite nella sezione 2.2.

**Divide** [13] di Enterproid supporta i dispositivi con sistema operativo iOS e Android. A differenza di Samsung KNOX, Divide non mantiene i dati aziendali sul dispositivo, bensì su di un file system cloud. Da remoto è possibile bloccare o formattare i dispositivi e anche installare delle applicazioni. I dispositivi client utilizzano un *passcode* che può essere eventualmente reimpostato dal server remoto in caso l'utente lo dimentichi.

### 2.1.3. Soluzioni Remote Workplace

**Receiver** [14] di Citrix Systems supporta i dispositivi con sistema operativo Windows, Android, Apple e Blackberry. Questo software permette all'utente di accedere ai dati aziendali, da qualunque dispositivo e da qualunque luogo, accedendo a un desktop virtuale già configurato senza salvare alcun dato sul dispositivo stesso.

**CloudOn** [15] di CloudOn è una soluzione ibrida fra Dual Persona e Remote Workplace. Supporta i dispositivi iOS e Android. Consente all'utente di modificare, con Microsoft Office, i documenti salvati su Dropbox, Box o GDrive. I file sono recuperati quando necessario, salvati nella locazione originale e quindi immediatamente cancellati dal dispositivo.

### 2.1.4. Soluzioni Virtualization Platform

**Horizon Mobile** [16] di VMware supporta i dispositivi Android e iOS. Sul dispositivo registrato viene creata una macchina virtuale ad hoc per permettere all'utente di lavorare e al contempo di poter accedere ai propri dati ed alle proprie applicazioni.

## 2.2. Samsung KNOX

Samsung KNOX è una soluzione per il BYOD di tipo Dual Persona sviluppata da Samsung già introdotta nella sezione 2.1.2.

Per garantire la sicurezza del sistema, KNOX si basa su diverse funzionalità (anche hardware), riportate in Figura 2, del dispositivo su cui viene installato [17].

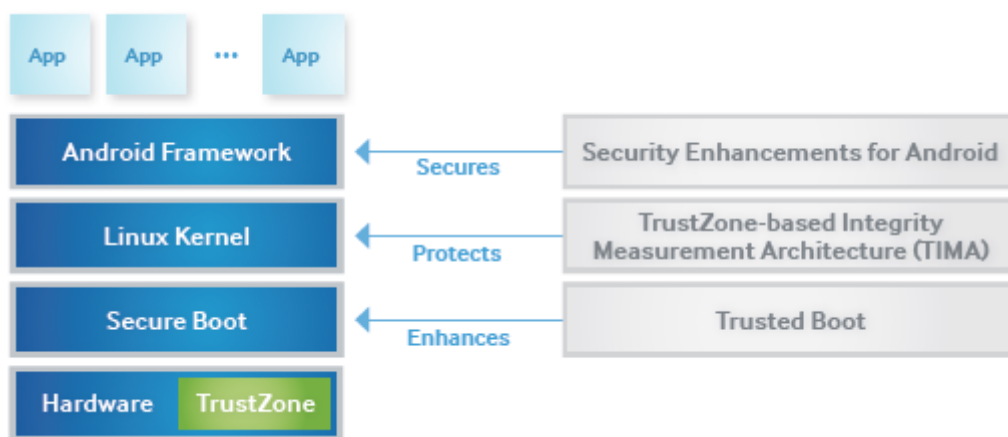


Figura 2: Architettura di Samsung KNOX

### 2.2.1. TIMA

La componente principale, senza cui il sistema non potrebbe garantire la sicurezza, è il TIMA (ARM TrustZone-based Integrity Measurement Architecture): un settore antimanomissione del processore ARM che verifica che il kernel del dispositivo non sia stato compromesso. Esso opera controlli periodici che mirano a verificare che il kernel non sia stato modificato, confrontandolo col kernel originale di fabbrica; inoltre, autentica ciascun modulo kernel quando viene caricato dinamicamente.

A partire da KNOX 2.0, il TIMA Real-time Kernel Protection (RKP) intercetta eventi critici che avvengono all'interno del sistema operativo e li ispeziona all'interno della TrustZone al fine di verificare la non manomissione del sistema operativo stesso. Se RKP stabilisce che un evento possa andare a impattare sull'integrità del sistema operativo, o lo interrompe o manda al MDM (il server dell'azienda cui deve rispondere il dispositivo) una comunicazione di sospetta manomissione. In Figura 3 è riportato lo schema di funzionamento del TIMA.

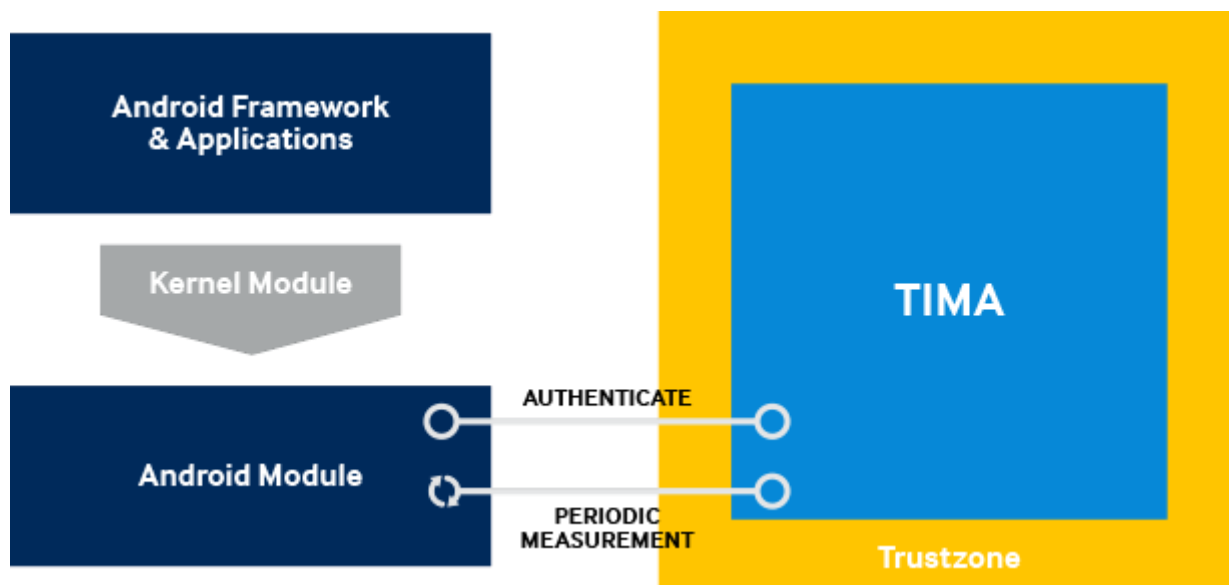


Figura 3: Funzionamento del TIMA in KNOX

Il TIMA Client Certificate Management (CCM), invece, permette la gestione (salvataggio, accesso, cifratura, decifratura, firma, verifica, ...) di certificati digitali in modo simile a una smart card. I certificati e le chiavi associate sono cifrati con una chiave unica per ciascun dispositivo che può decifrata solo attraverso l'esecuzione di un codice eseguito nella TrustZone.

### 2.2.2. Secure Boot

Il *Secure Boot* (alle volte chiamato *Verified Boot*) è la prima linea di difesa contro gli attacchi a un dispositivo. È un meccanismo di sicurezza che impedisce che boot loader o kernel non autorizzati vengano caricati durante l'avvio del dispositivo. Gli autorizzati sono quelli firmati da fonti conosciute e fidate.

Per far ciò, com'è possibile vedere in Figura 4, KNOX usa dei controlli sistematici. A livello hardware, il Primary Boot Loader utilizza un certificato PKI per verificare l'integrità del Secondary Boot Loader 1 che, analogamente, verifica l'integrità del Secondary Boot Loader 2. Similmente quest'ultimo verifica l'integrità dell'Android Boot Loader che, infine, caricherà solamente un kernel Samsung autorizzato, ovvero con un certificato Samsung come suo Root-of-Trust. Durante questo processo, l'ARM TrustZone salva i fingerprint (detti "misurazioni") di tutti i boot loader e del kernel che verranno usati dal trusted boot, descritto nella sezione 2.2.3.

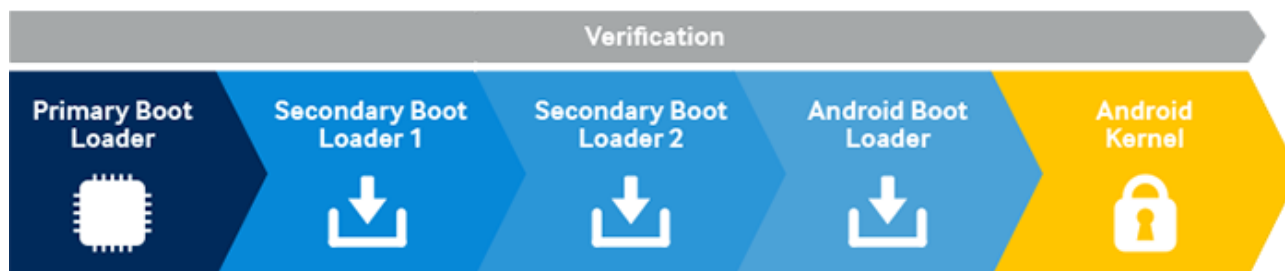


Figura 4: Secure boot in KNOX

### **2.2.3.Trusted Boot**

Il *Trusted Boot* (alle volte chiamato *Authenticated Boot*) prosegue il lavoro del secure boot in modo da garantire maggiormente l'integrità del kernel. A run-time, continuano a essere verificate le misurazioni che erano state salvate nella TrustZone durante l'avvio del dispositivo. Le chiavi crittografiche usate da KNOX sono mantenute nel TIMA Keystore, all'interno della TrustZone, e vengono rilasciate esclusivamente quando la TrustZone, attraverso il processo di trusted boot, conferma che le misurazioni del sistema operativo siano le stesse di quelle salvate all'avvio del dispositivo.

Il TIMA-Keystore mette inoltre a disposizione delle applicazioni servizi per generare e mantenere chiavi crittografiche che potranno esser utilizzate solo se il trusted boot conferma l'autenticità del sistema.

### **2.2.4.Security Enhancements (SE)**

Unix e Linux permettono agli utenti di ottenere privilegi per leggere, scrivere ed eseguire file, in un processo definito DAC (Discretionary Access Control). Un utente può quindi (volontariamente o involontariamente) divulgare delle informazioni e permettere l'accesso non autorizzato a dati, applicazioni e risorse, fra cui anche password o dati sensibili.

KNOX protegge il sistema operativo attraverso SE per Android, che si basa sulla tecnologia SELinux, introdotta dalla NSA nel 2010. Esso permette di definire, attraverso un sistema di policy, quali utenti e quali applicazioni possono accedere a particolari file e risorse del sistema operativo forzando il MAC (Mandatory Access Control). Queste policy sono definite centralmente dall'amministratore aziendale e non possono essere sovrascritte dagli utenti; inoltre, anche il superuser del dispositivo è soggetto a MAC. A partire da KNOX 2.0, viene introdotto SEAMS (SE for Android Management Service) che fornisce un accesso controllato al sistema di policy SELinux. Tale servizio è usato sia internamente da KNOX 2.0, sia da terze parti, in particolare, dalle aziende.

SE per Android, inoltre, mette in sicurezza il sistema operativo separandolo in due domini di sicurezza distinti, in ciascuno dei quali alle applicazioni sono accordati i permessi minimi per funzionare. In questo modo, gli eventuali danni vengono limitati ad un'unica area, lasciando inalterata l'altra.

### **2.2.5.Attestation**

Attraverso l'attestation viene verificato che il kernel corrente sul dispositivo sia autorizzato, confrontandone le misurazioni con quelle fatte all'avvio. L'attestation si basa sulla coppia di chiavi pubblica e privata che viene assegnata in fabbrica in modo unico a ciascun dispositivo, assieme al certificato firmato da Samsung per la chiave pubblica. L'attestation server invia un challenge casuale al dispositivo per verificarne l'integrità. Il challenge è inviato assieme ad un nonce (KNOX usa un numero casuale di 32 bit) per evitare replay-attack. La TIMA risponde alla challenge



inviando al server un blob firmato con la propria chiave privata (univoca per il dispositivo) contenente i seguenti dati [18]:

- Le misurazioni dei boot loader e del kernel
- Lo status del SE per Android
- Il device ID unico (chip ID)
- L'IMEI del dispositivo
- Il valore del bit di violazione della garanzia
- Il nonce

Poiché fra quando il dispositivo genera l'attestation e quando questi ne conferma la validità il dispositivo potrebbe venir compromesso, l'attestation ha una validità limitata.

A differenza del Trusted Boot, l'attestation può esser richiesta in modo asincrono dal MDM.

### 2.2.6. On-Device Encryption (ODE)

KNOX abilita l'ODE, il cui funzionamento è riportato in Figura 5, di default. Usa un algoritmo di cifratura AES a 256 bit per cifrare i dati sull'intero dispositivo, compresi lo storage interno e l'SD esterna. La chiave usata per la cifratura è protetta da una password dell'utente.

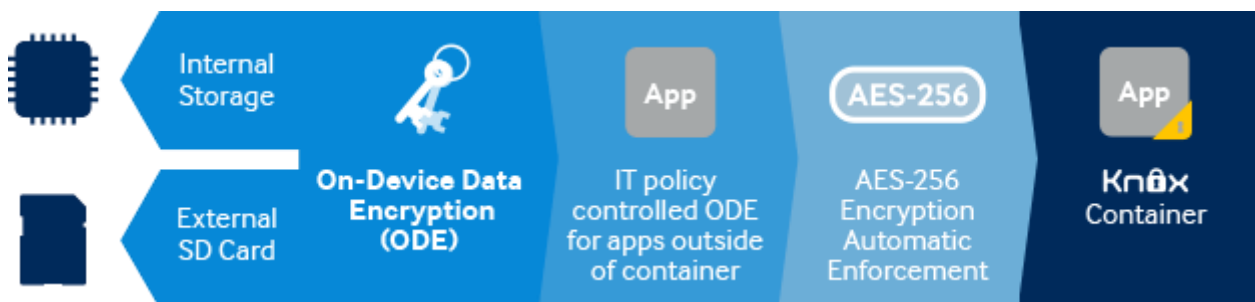


Figura 5: ODE di KNOX

### 2.2.7. MDM

Samsung KNOX può essere gestito attraverso un sistema MDM. In questo modo, in remoto, il responsabile del reparto IT dell'azienda può fare un *wipe* dell'intero dispositivo nel caso in cui risultasse compromesso. In Figura 6 sono riportate le funzionalità per l'MDM messe a disposizione da KNOX.

ENTERPRISE NEED	MANAGEMENT POLICIES			
Configuration	Wi-Fi	Security	Email accounts	
	Bluetooth	Password	Browser	
Controls & Limits	Kiosk mode	Application permissions	Lock screen	Camera
	Roaming	Phone restrictions	APN settings	
Enterprise Integration	VPN	Exchange account permissions	Browser	SSO
	Geofencing	CAC	Firewall	
Monitoring	Call logs	Message logs	Device inventory	Integrity monitoring
Prevent Data Leak	Email forwarding	Container management	Integrity management	
SRG-specific Policies	CAC	FIPS VPN	TBD	


 Samsung KNOX API (includes SAFE API)

Figura 6: Funzioni per MDM in KNOX

## 2.2.8. Applicazioni e informazioni protette

Il container di Samsung KNOX è un ambiente sicuro Android all'interno del dispositivo, completo della sua *homescreen*, *launcher*, applicazioni e *widget*. Le applicazioni e i dati al suo interno non possono interagire con quelli al suo esterno. Come riportato in Figura 7, le applicazioni all'interno del container di KNOX devono essere contenute all'interno di un ulteriore strato di sicurezza che viene fornito da KNOX stesso attraverso un servizio che estrae dall'APK il certificato dello sviluppatore e aggiunge dei file per garantire la sicurezza delle operazioni all'interno del container di KNOX. Una volta fatto ciò, prima di essere installata, l'applicazione è inviata al servizio Quality Assurance (QA) di Samsung per testarne la compatibilità col dispositivo, le funzioni di base, e identificare eventuali comportamenti malevoli e pericolosi.

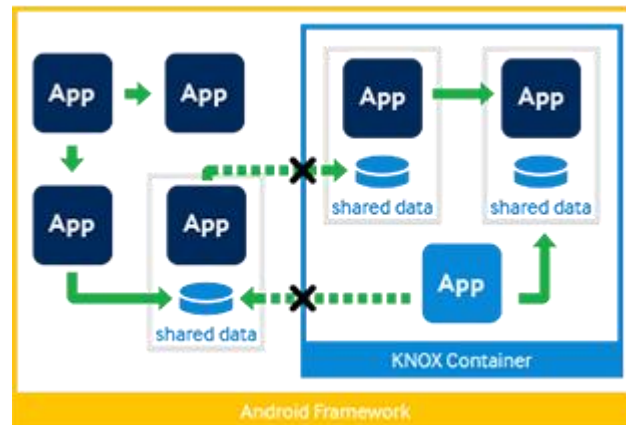


Figura 7: Container di KNOX

## 2.3. Android Lollipop

A partire da Android 5.0, noto anche come Android L o Android Lollipop [19], alcune caratteristiche software di Samsung KNOX vengono integrate all'interno del sistema operativo, come è possibile vedere in Figura 8. La separazione degli ambienti è ottenuta sfruttando la già presente architettura multi-utente di Android. I dati vengono mantenuti al sicuro grazie alla crittografia del disco che avviene a partire dal primo avvio [20], al *Verified Boot* e all'adozione obbligatoria da parte delle applicazioni del sistema SELinux per l'accesso alle risorse.

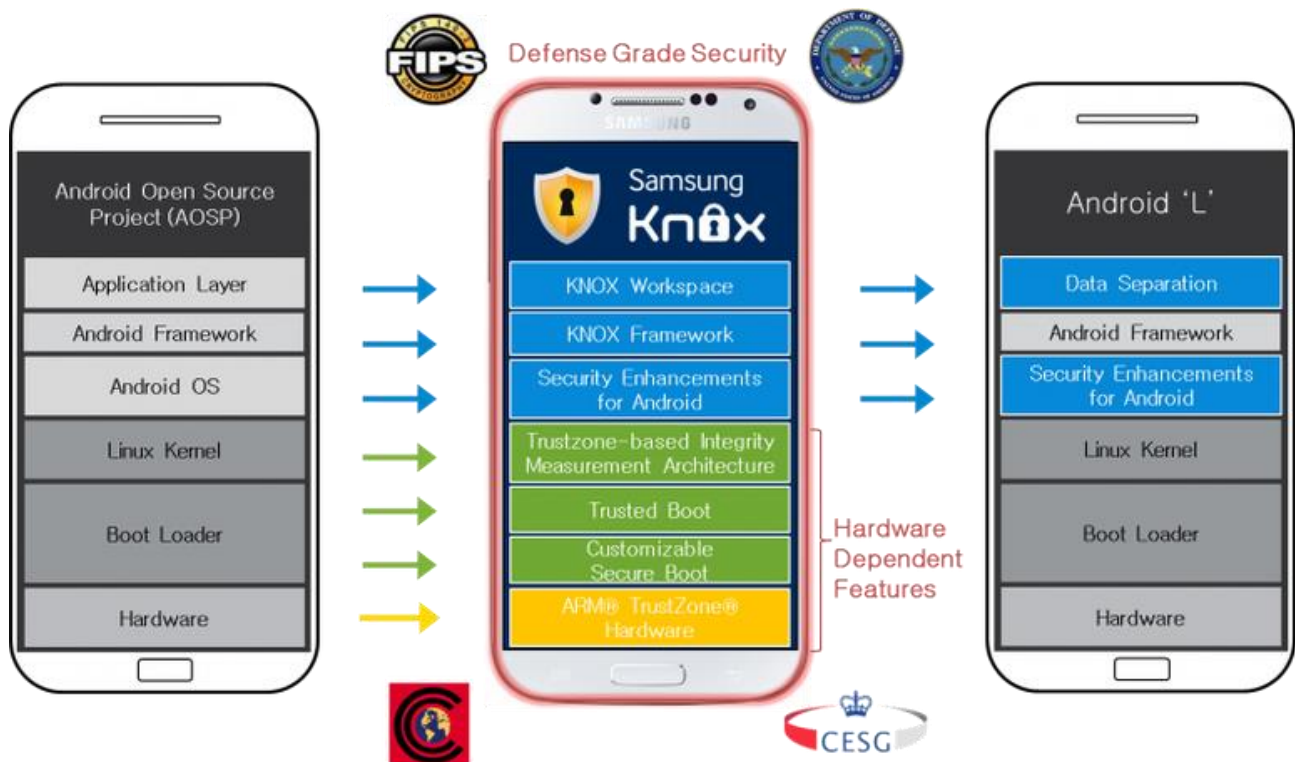


Figura 8: Le parti di Samsung KNOX che vengono integrate in Android L

## 2.4. Limitazioni dell'offerta attuale

Tutte le soluzioni descritte finora non si adattano facilmente a scenari in cui l'utente possa muoversi di azienda in azienda e quindi in cui il suo dispositivo debba esser conforme a differenti policy.

In particolare, aziende diverse potrebbero voler usare le stesse policy, per cui è conveniente certificare i dispositivi rispetto ad esse, non rispetto a delle aziende, in modo che un dispositivo non debba esser ri-verificato se si sposta in una nuova azienda la cui policy è compatibile con una di quelle per cui il dispositivo, nella configurazione attuale, è stato certificato.

In questo lavoro studieremo proprio scenari di questo tipo estendendo BYODroid, presentato in sezione 2.1.1. Ci limiteremo, quindi, all'utilizzo di dispositivi con sistema operativo Android e assumeremo che:

- non siano necessari i permessi di root
- i dispositivi abbiano attive alcune componenti hardware di sicurezza come il TEE (Trusted Execution Environment), trattato nel capitolo 5, ipotesi senza la quale la soluzione illustrata non può funzionare correttamente; tuttavia nei moderni dispositivi sono quasi sempre presenti [21].

### 3. L'approccio BYOD-Cert

In questa tesi presentiamo un approccio che permette solo a dispositivi "fidati" l'accesso a reti aziendali. In realtà, non sono i dispositivi a essere "fidati", in senso assoluto, ma le loro configurazioni possono, o meno, rispettare le policy di sicurezza di una certa azienda.

Per l'autenticazione alla rete aziendale, e la verifica dei certificati, si è scelto di utilizzare il protocollo WPA2-Enterprise, spesso già adottato nelle aziende e che fa uso di certificati X.509 per l'autenticazione. Tale protocollo sarà approfondito nel capitolo 4.

Quando un utente vuole connettersi alla rete aziendale con un proprio dispositivo mobile, egli dovrà aver installata sul dispositivo l'applicazione client di BYOD-Cert e quindi autenticarsi alla rete aziendale assicurando di soddisfare le policy richieste. Per dimostrare il soddisfacimento di una certa policy, è sufficiente che il dispositivo posseda un certificato valido per quella policy rilasciato dal server BYOD-Cert. Affinché il server possa rilasciare tale certificato, esso necessita delle informazioni che gli vengono comunicate dall'applicazione client. In particolare, è necessario l'elenco delle applicazioni installate; tale argomento verrà approfondito nella sezione 6.1.

La comunicazione client-server è di tipo asincrono e per essa si è scelto di appoggiarsi al GCM (Google Cloud Messaging, trattato in sezione 3.2.2). La comunicazione deve avvenire, in particolare, ogni volta che l'utente voglia installare una nuova applicazione, in modo da poter avere un nuovo certificato di conformità per le policy cui è interessato. Tuttavia, non sempre l'installazione può continuare a garantire la conformità del dispositivo a determinate policy. Pertanto, il server potrebbe dover richiedere l'installazione (o la non installazione) di determinate applicazioni e, quindi, il client deve essere in grado di seguire eventuali istruzioni d'installazione di un'applicazione che il server gli invia.

I certificati rilasciati dal server sono in formato X.509v3 e possono essere revocati in ogni momento. Il client deve essere quindi in grado di gestirli; in particolare deve essere in grado di effettuare su di essi operazioni di aggiunta, rimozione ed elenco. In sezione 4.5 verrà approfondito come realizzare un certificato X.509v3 con attributi custom che identificano la policy, mentre nella sezione 6.2 verrà trattata la gestione dei certificati lato client.

Tuttavia, per garantire l'autenticità (e di conseguenza l'affidabilità) dell'applicazione client, è necessario che il dispositivo metta a disposizione delle tecniche per la remote attestation, approfondite nel capitolo 5.

In sezione 3.1 verranno analizzati gli attori, in sezione 3.2 i casi d'uso e in sezione 3.3 il contenuto dei certificati di conformità.

### 3.1. Gli attori

Nell'utilizzo di BYOD-Cert sono coinvolti i seguenti attori:

- *Utente*: l'utente finale che utilizza il dispositivo, di cui è probabilmente il proprietario;
- *Client*: l'applicazione BYOD-Cert installata sul dispositivo dell'utente che mantiene i certificati di conformità del dispositivo a delle policy aziendali e che consente all'utente di installare nuove applicazioni sul proprio dispositivo (eventualmente instrumentandole). Si assume che esso sia fidato, potendone controllare l'integrità mediante remote attestation;
- *BYOD-Server*: il server BYOD che verifica che determinate configurazioni di un dispositivo siano conformi ad un certo insieme di policy. Fornisce istruzioni al client per l'instrumentazione di applicazioni e rilascia e revoca certificati di conformità;
- *MDM Server*<sup>1</sup>: il server RADIUS (aziendale) per l'accesso alla rete aziendale.

### 3.2. Casi d'uso

Tutte le comunicazioni avvengono su canali sicuri, per esempio HTTPS, e utilizzano *nonce* generati dal server onde evitare reply-attack.

Le sezioni seguenti dettagliano i vari casi d'uso del sistema.

#### 3.2.1. Registrazione di un nuovo client (First app execution)

Questo scenario, illustrato in Figura 9, avviene quando il Client è eseguito per la prima volta sul dispositivo dell'utente, in seguito alla sua installazione. In tal caso è necessario registrare il dispositivo utente sul BYOD-Server in modo da poter ottenere tutte le informazioni atte a verificarne la corretta configurazione.

- *Utente*: avvia (per la prima volta) l'applicazione Client sul proprio dispositivo.
- *Client*: recupera (dal dispositivo o dall'utente) e invia al BYOD-Server:
  - Un'e-mail Google dell'utente;
  - Il device ID<sup>2</sup>
  - La versione del Client
- *BYOD-Server*: Verifica i dati inviati dal Client verificando che il numero di versione non sia precedente a una versione eventualmente precedente installata sempre su quel dispositivo per quell'utente<sup>3</sup>.
- *BYOD-Server*: In caso le informazioni non siano valide, invia un codice di errore al Client.
- *BYOD-Server*: Se le informazioni inviate dal Client sono corrette, le salva e invia al Client un codice di risposta positivo.

---

<sup>1</sup> In realtà il Client non comunica direttamente con il MDM Server, ma con un Access Point che inoltra la comunicazione al MDM Server. Tuttavia, per semplicità, nel seguito verrà omesso tale attore.

<sup>2</sup> Il device ID [20], noto meglio come `SETTINGS.SECURE.ANDROID_ID`, è un numero a 64 bit generato e memorizzato la prima volta che il dispositivo viene avviato. E' stato introdotto in Android 2.2. A partire da Android 4.2 è stata introdotta la funzionalità di multi-utente e tale numero è quindi associato a ciascun utente [21].

<sup>3</sup> Ciò per evitare che l'utente effettui un downgrade di versione atto a sfruttare eventuali vulnerabilità già corrette nella versione corrente.

- *Client*: Se riceve codice di risposta positivo procede con il login.

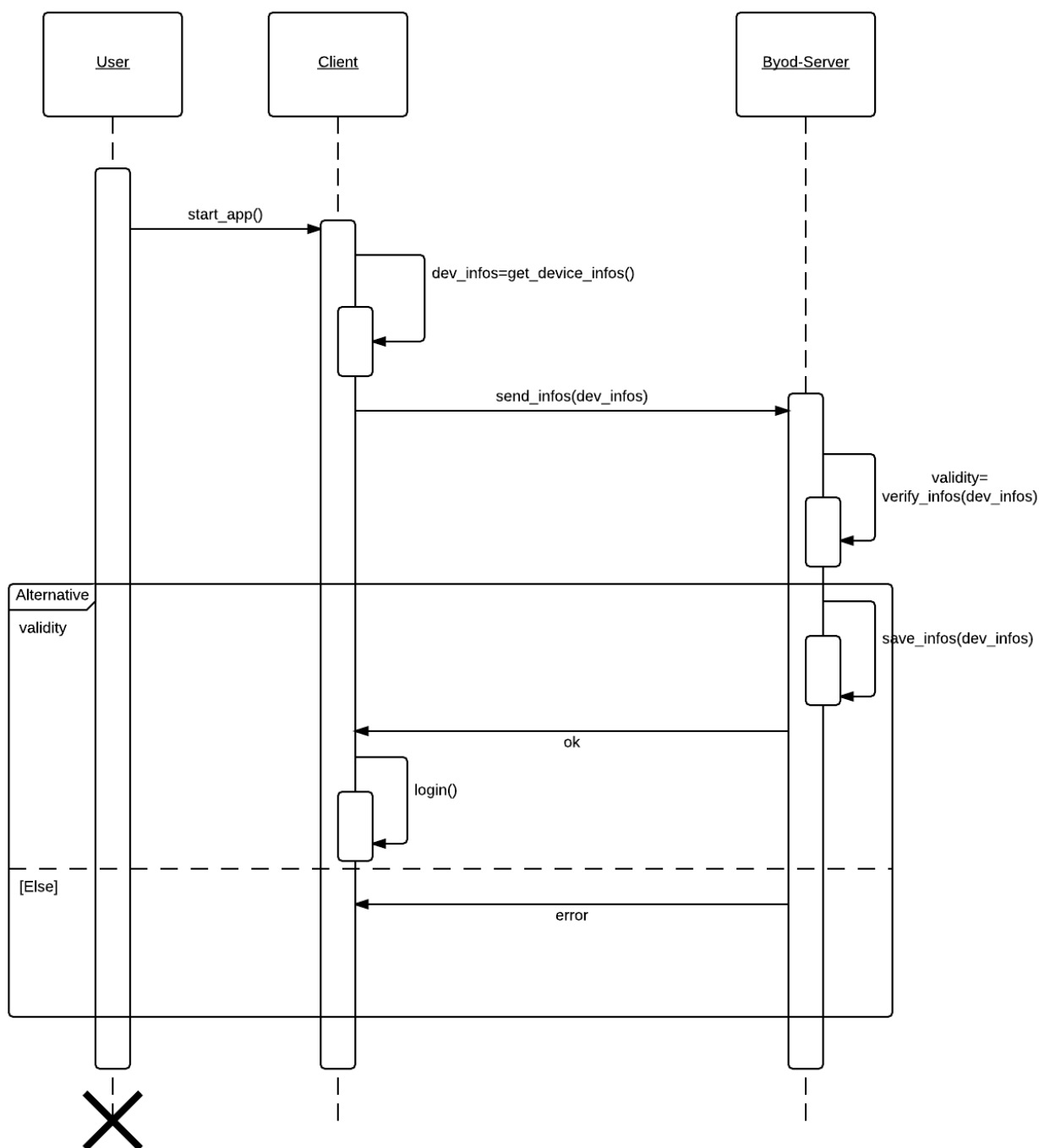


Figura 9: Sequence Diagram "Registrazione nuovo client"

### 3.2.2.Login

Il seguente scenario, illustrato in Figura 10, corrisponde alla procedura di accesso all'applicazione Client da parte dell'utente dopo essersi già registrato al servizio BYOD. In seguito a tale operazione, in caso di successo, egli potrà accedere a tutte le funzionalità del Client, compresa l'installazione di nuove applicazioni.

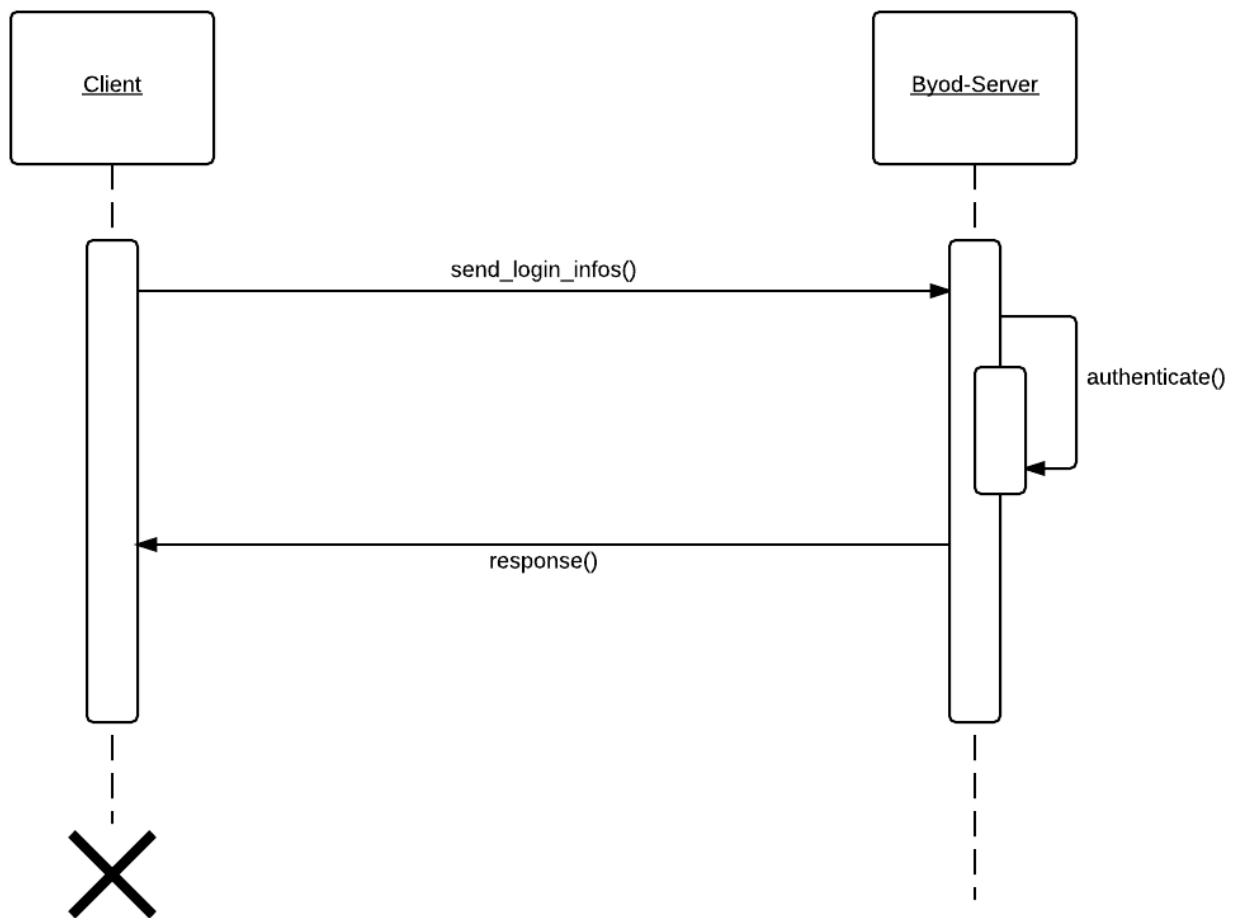


Figura 10: Sequence Diagram "Login"

Onde permettere l'installazione e la disinstallazione remota delle applicazioni sul dispositivo, al login viene trasmesso ogni volta al server il corrente codice GCM. Questo servizio, la cui architettura è riportata in Figura 11 e che è messo a disposizione gratuitamente da Google, consente a un server registrato di inviare messaggi a una determinata istanza di una specifica applicazione registrata e viceversa [22]. Il numero di GCM di un'istanza può cambiare in seguito ad aggiornamenti dell'applicazione stessa [23]. Inoltre, ogni messaggio GCM ha un tempo di vita limitato ed esiste un numero massimo di messaggi che possono rimanere in attesa di esser consegnati, raggiunto il quale tutti i messaggi pendenti vengono rimossi dalla coda; pertanto non è garantito che un messaggio GCM giunga sempre a destinazione.

- *Client*: manda E-mail, Device ID, e GCM (se cambiato) al BYOD-Server.
- *BYOD-Server*: verifica che le credenziali corrispondano a quelle di un utente registrato e in caso sia stato inviato salva il GCM. Invia un codice di risposta al Client.
- *Client*: Se il codice di risposta è positivo procede.



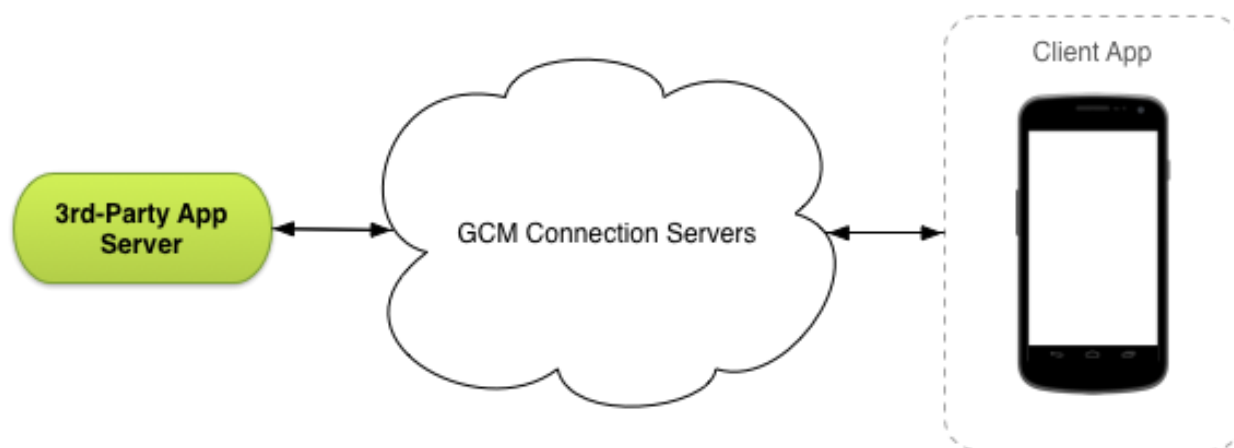


Figura 11: Architettura di GCM

### 3.2.3. Richiesta di certificazione del dispositivo

In questo scenario, illustrato in Figura 12, il Client richiede al BYOD-Server di certificare l'attuale configurazione del dispositivo per una determinata policy. Tale scenario è tipicamente eseguito in seguito alla richiesta di un certificato valido da parte del MDM Server al Client.

- *Client*: recupera l'attuale configurazione del dispositivo, ovvero le seguenti informazioni sulle applicazioni attualmente installate:
  - App Name
  - Package Name
  - Version Code
  - Permessi
  - Signatures
- *Client*: firma digitalmente le informazioni di cui sopra, assieme all'identificativo della policy per cui richiede la certificazione, e le invia al BYOD-Server.
- *BYOD-Server*: riceve i dati dal client. Se la configurazione differisce, avendo più applicazioni, da quella eventualmente già salvata in precedenza (per quella stessa policy), revoca gli eventuali certificati già salvati.
- *BYOD-Server*: Verifica che la configurazione sia conforme alle policy per cui è richiesta la certificazione.
  - Se è conforme, si salva l'attuale configurazione, assegnando dei nomi univoci alle tuple d'informazioni di ciascun'applicazione, e genera un certificato che si salva e invia al Client.
  - Altrimenti, se la configurazione può divenire conforme in seguito all'installazione di una o più applicazioni, invia al Client le informazioni per instrumentare tali applicazioni.
  - Altrimenti, invia un codice di risposta indicante l'impossibilità di certificare la configurazione del dispositivo per le policy indicate.
- *Client*: Riceve la risposta dal BYOD-Server.
  - Se riceve un certificato: lo installa in modo sicuro sul dispositivo.

- Se riceve delle istruzioni d'istrumentazione: richiede all'utente se vuole procedere con l'istrumentazione e, in caso affermativo, la esegue. In seguito a tale operazione, l'applicazione (istrumentata) può essere installata e si può ripartire dal passo iniziale.
- In tutti gli altri casi comunica all'utente che non può ricevere il certificato.

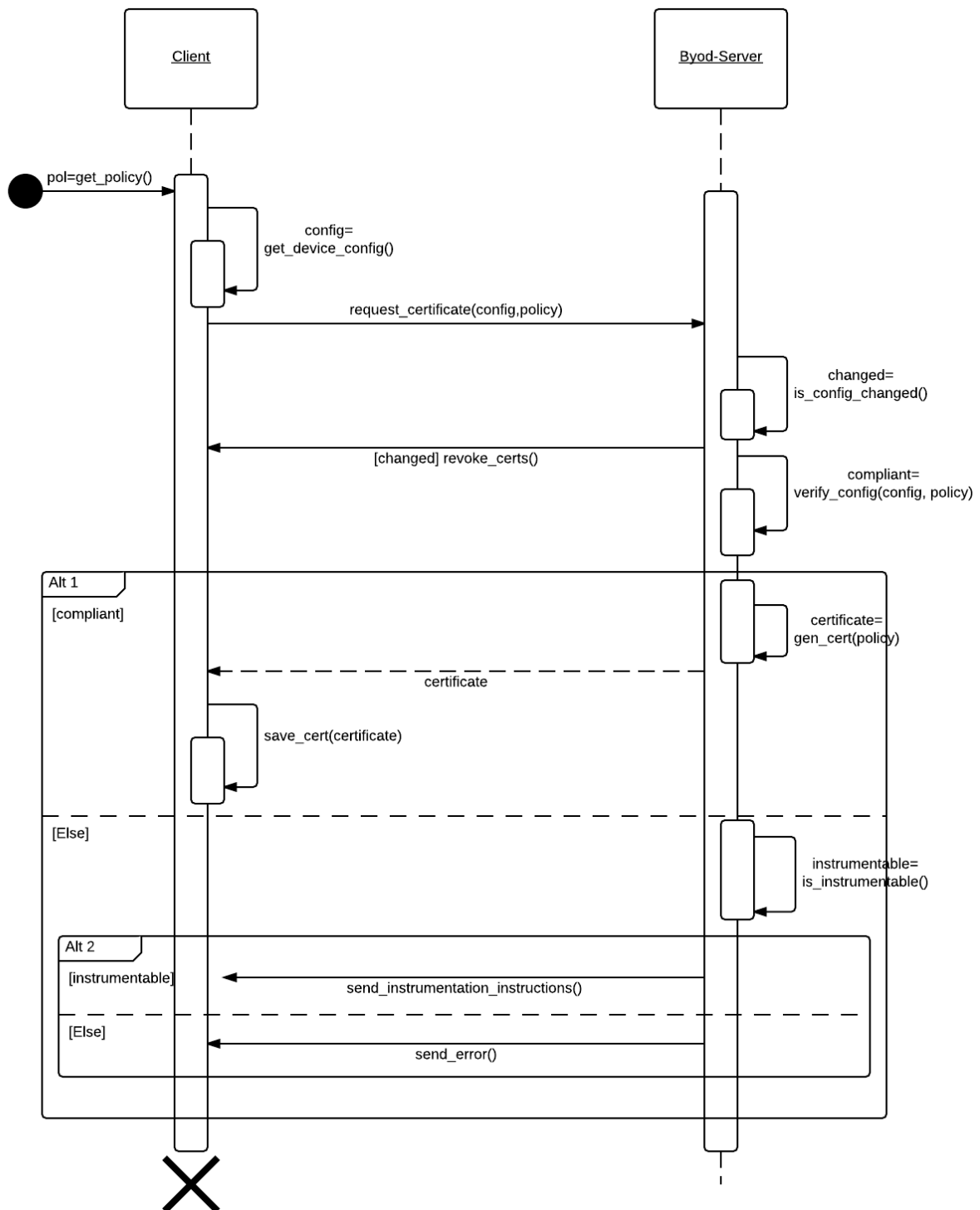


Figura 12: Sequence Diagram "Richiesta di certificazione"

### 3.2.4. Installazione di una nuova applicazione

Quando l'utente vuole installare una nuova applicazione sul proprio dispositivo tramite il Client, viene contattato il BYOD-Server per verificare che la nuova configurazione possa continuare ad essere conforme alle politiche attualmente in vigore per il dispositivo. Questo scenario è illustrato in Figura 13.

- *Utente*: sceglie l'applicazione da installare sul Client.
- *Client*: comunica al BYOD-Server le seguenti informazioni:
  - identificativo dell'applicazione che si vuole installare
  - Informazioni di login sul BYOD-Server
  - Policy per cui il dispositivo deve esser certificato
- *BYOD-Server*: riceve i dati dal Client e verifica che la nuova configurazione con la nuova applicazione sia conforme alle policy.
  - Se la configurazione è valida: salva la nuova configurazione del Client sul server, e genera un certificato per la nuova configurazione che si salva e invia al Client.
  - Se la configurazione non è conforme ma è possibile renderla tale tramite sua strumentazione: invia al Client le istruzioni per poter instrumentare l'applicazione in modo che la configurazione diventi valida.
  - Se la configurazione non è conforme: invia al Client un opportuno codice di errore.
- *Client*:
  - Se riceve le istruzioni per l'instrumentazione dell'applicazione: chiede all'utente e in caso positivo la installa. Richiede quindi al BYOD-Server un certificato per l'attuale nuova configurazione.
  - Se riceve un nuovo certificato: lo salva.
  - Se riceve un codice di errore: lo mostra all'utente e gli chiede come intende procedere.

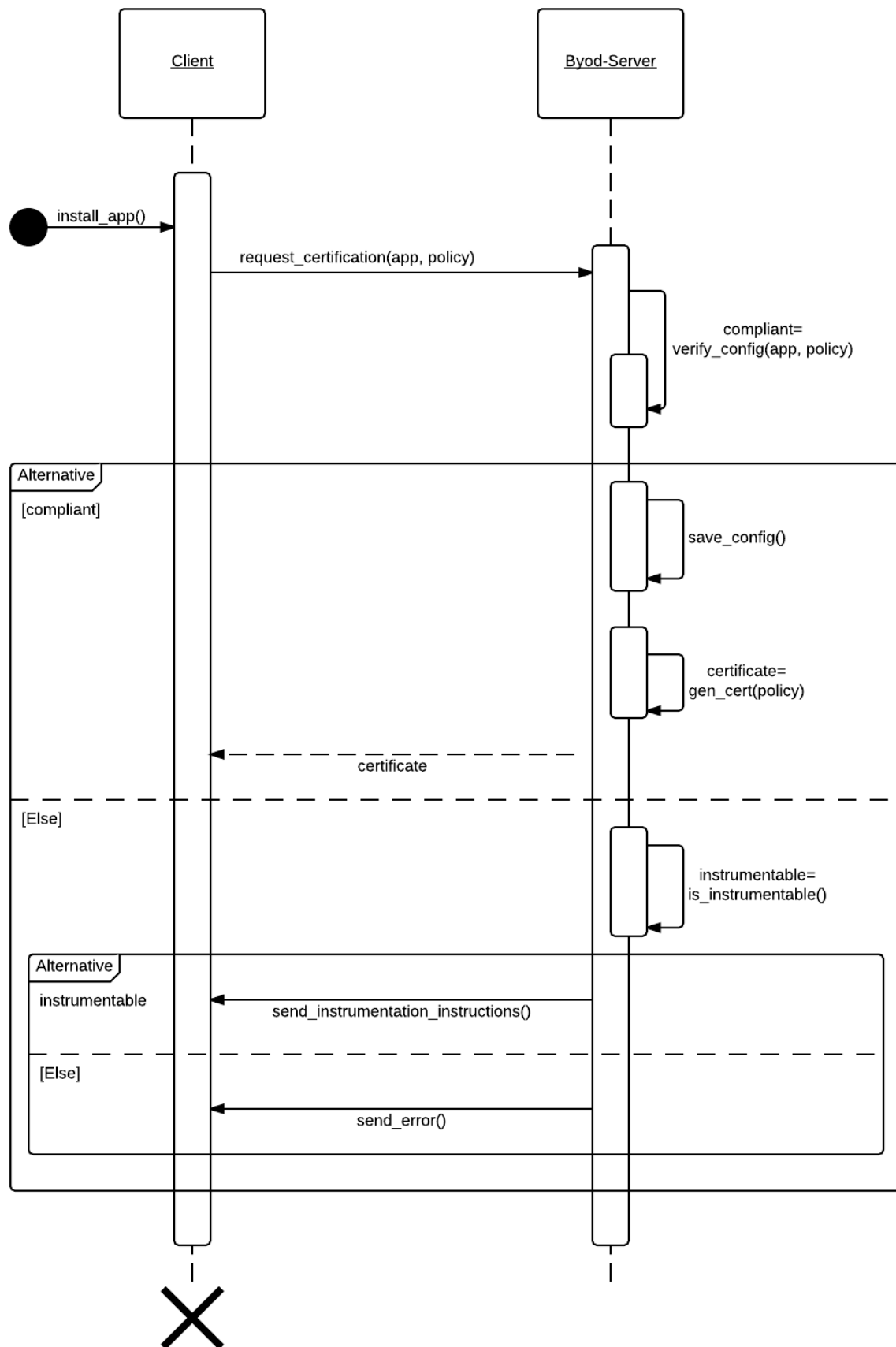


Figura 13: Sequence Diagram "Installazione nuova applicazione"

### 3.2.5. Utente invalida l'attuale configurazione certificata

Un utente potrebbe decidere di installare (senza strumentarla) un'applicazione che non garantisca il rispetto delle policy aziendali cui finora era sottostato.

- *Utente*: installa un'applicazione illegale
- *Client*: comunica al BYOD-Server la "violazione" e invalida i certificati
- *BYOD-Server*: revoca dei certificati del client

### 3.2.6. Disinstallazione di un'applicazione

In seguito alla disinstallazione di un'applicazione da parte dell'utente non è necessario richiedere una nuova certificazione per la configurazione del dispositivo. Lo scenario è illustrato in Figura 14.

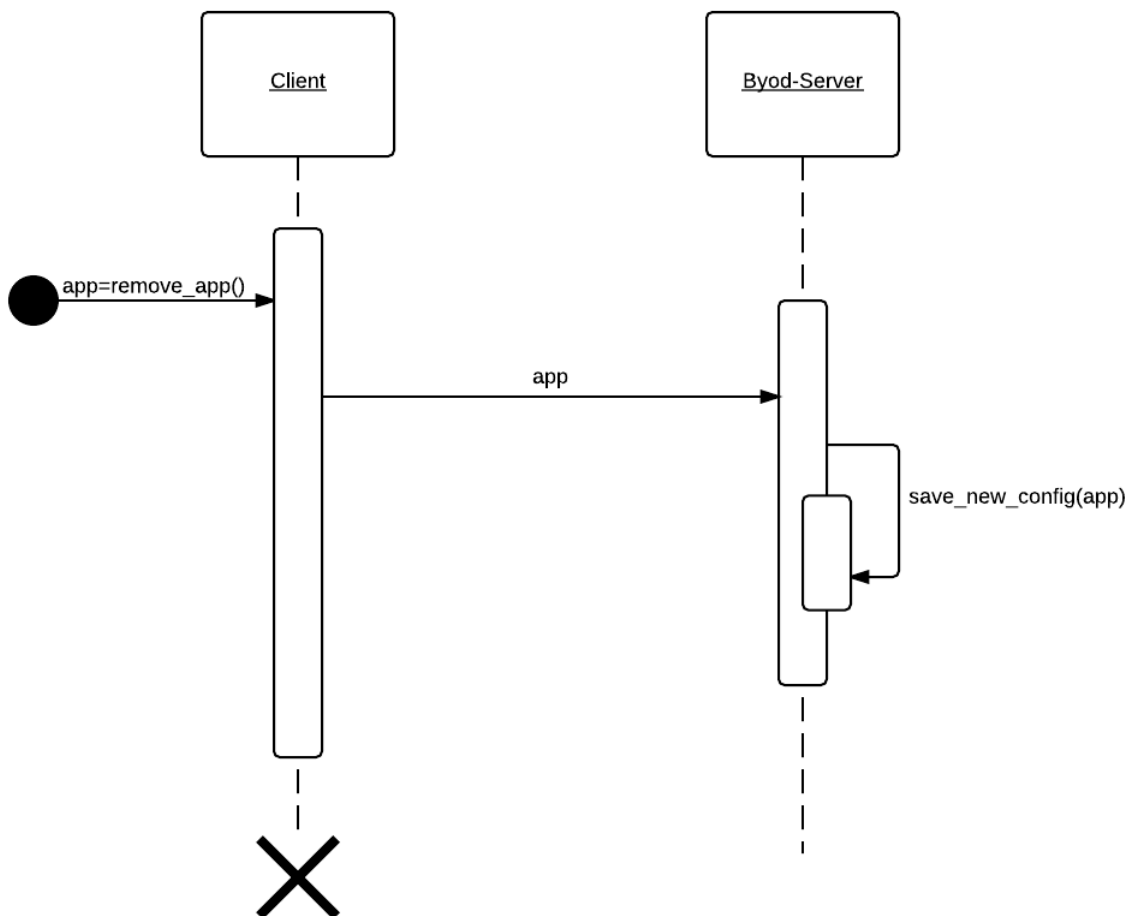


Figura 14: Sequence Diagram "Disinstallazione applicazione"

- *Utente*: disinstalla un'applicazione dal dispositivo.
- *Client*: comunica al BYOD-Server l'applicazione che si è disinstallata.
- *BYOD-Server*: salva la nuova configurazione del dispositivo.

### 3.2.7. Aggiornamento di un'applicazione

Un utente con il proprio dispositivo in una configurazione certificata, può voler aggiornare un'applicazione installata.

È consigliabile che l'utente disabiliti l'aggiornamento automatico delle applicazioni sul proprio dispositivo, delegando l'applicazione client a effettuare l'aggiornamento delle applicazioni. Se il client avesse anche un market, simile a quello già presente in BYODroid, potrebbe consentire all'utente l'aggiornamento delle applicazioni che, anche in seguito all'aggiornamento, siano conformi alle policy per cui il dispositivo è attualmente certificato.

Se, tuttavia, l'utente decidesse di aggiornare un'applicazione senza utilizzare l'applicazione client, questi dovrà invalidare i certificati già presenti sul dispositivo e richiedere al BYOD-Server una nuova certificazione come in sezione 3.2.4.

- *Utente*: aggiorna un'applicazione del dispositivo.
- *Client*: invalida i certificati e comunica al server l'applicazione aggiornata.
- *BYOD-Server*: procede come nel caso d'installazione di una nuova applicazione.

### 3.2.8. Revoca di un certificato

In diverse occasioni il BYOD-Server può aver necessità di revocare un certificato che aveva precedentemente rilasciato. Il meccanismo di revoca di certificati è realizzato sfruttando i meccanismi di revoca già presenti per i certificati X.509. Per velocizzare l'invalidazione da parte del client:

- *BYOD-Server*: comunica al client il certificato da revocare tramite una notifica GCM.
- *Client*: cancella il certificato dal dispositivo.

### 3.2.9. Ingresso nella rete aziendale

Questo scenario, illustrato in Figura 15, si verifica quando il dispositivo dell'utente si connette alla rete aziendale. Dopo che l'MDM Server ha verificato le credenziali dell'utente, procede con la verifica del dispositivo al cui Client richiede un certificato valido per le policy aziendali. In assenza di tale certificazione il dispositivo non può essere ammesso.

- *MDM Server*: chiede al client una certificazione valida per una policy P.
- *Client*: verifica di possedere un certificato valido per P.
  - Se posseduto: invia all'azienda tale certificato.
  - Se non posseduto: richiede al BYOD-Server una certificazione per P.
- *MDM Server*: verifica che il certificato sia valido per la policy P
  - Se sì: autentica il client
  - Altrimenti: non lo autentica

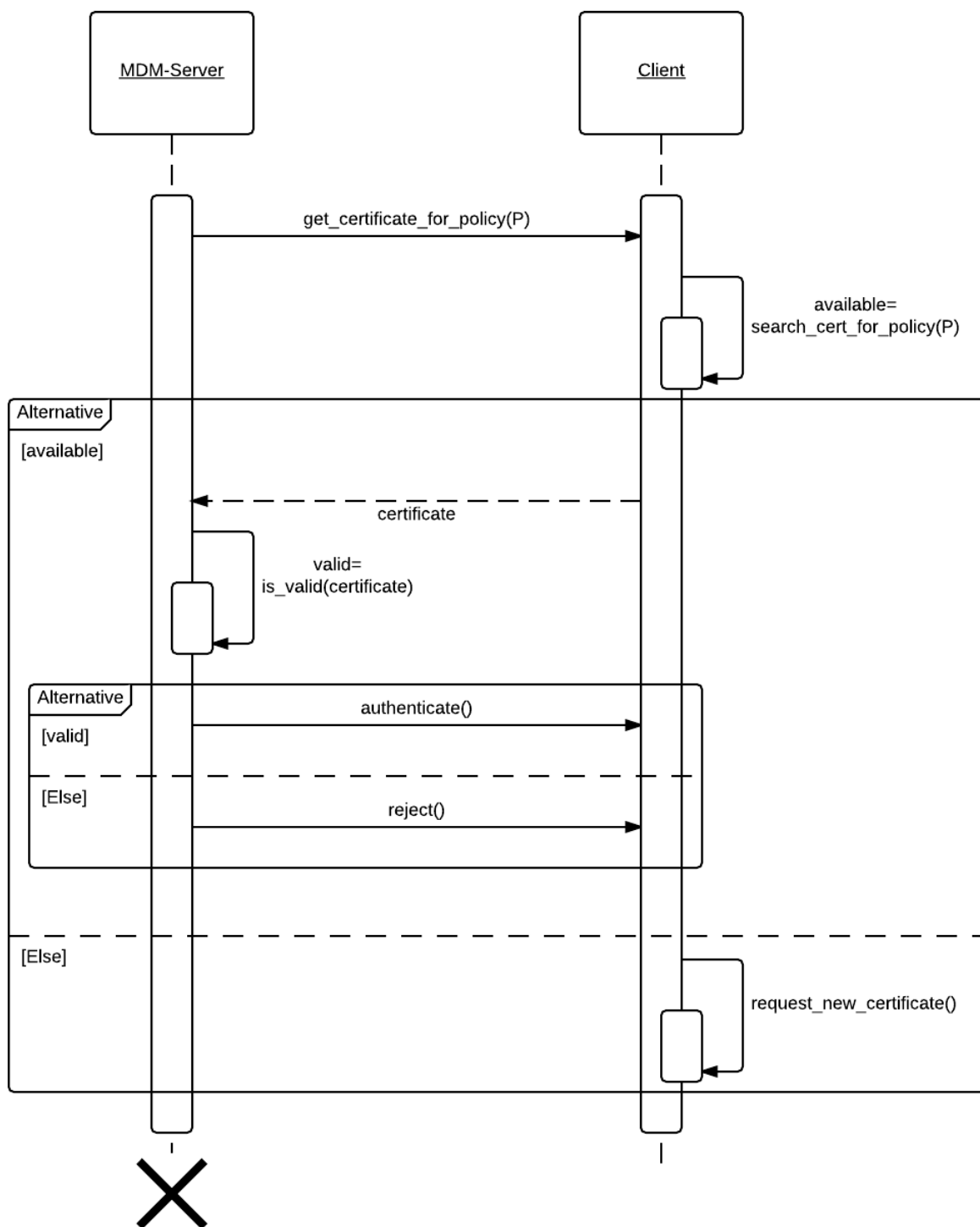


Figura 15: Sequence Diagram "Ingresso in azienda"

### 3.2.10. Permanenza del dispositivo in azienda

Durante tutta la permanenza del dispositivo Client all'interno della rete aziendale, esso è sottoposto periodicamente a dei challenge da parte del MDM-Server al fine di verificare che l'utente non abbia disinstallato il Client, potendo così aggirare le policy aziendali. In caso il Client non risponda più di tre volte di fila a un challenge, o non risponda correttamente, il dispositivo verrà scollegato dalla rete aziendale e dovrà quindi effettuare nuovamente l'ingresso.

### **3.2.11. Disinstallazione del Client**

Per una soluzione completamente sicura, il Client deve essere installato all'interno del TEE. In caso non fosse possibile, una soluzione parziale (non completamente sicura) per verificare quando il Client viene disinstallato dal dispositivo, fa ricorso a una seconda applicazione Client che controlla la prima e viceversa. Anch'essa sarà sempre in ascolto di *Intent* di disinstallazione generati dal sistema operativo. In caso ne sia intercettato uno relativo all'avvenuta disinstallazione dell'altra applicazione, i certificati vengono eliminati e il dispositivo viene immediatamente sconnesso dalle reti.

### **3.2.12. Rimozione di un utente dall'azienda**

In ogni momento, l'azienda potrebbe voler impedire, magari in seguito al fatto che l'utente non sia più affiliato all'azienda, ogni futura connessione alla rete aziendale da parte di un utente.

Questo caso non differisce dal normale uso di WPA2-Enterprise e sarà quindi sufficiente revocare le sue credenziali di accesso, revocando tutti i certificati rilasciati a quel dispositivo.

### **3.2.13. Richiesta di certificazione per una policy**

Tipicamente in seguito alla variazione di una policy aziendale, il BYOD-Server potrebbe voler richiedere a un Client di certificarsi nuovamente per una specifica policy.

In questo scenario, il BYOD-Server invaliderà tutti i certificati corrispondenti alla vecchia versione della policy e invierà una notifica GCM al Client che effettuerà una nuova certificazione del dispositivo secondo lo scenario descritto in 3.2.3.

## **3.3. Certificati di conformità**

I certificati che vengono rilasciati dal BYOD-Server al Client contengono le seguenti informazioni:

- Identificativo della policy per la cui configurazione è conforme
- Data di emissione
- Data di scadenza/validità
- Identificativo del dispositivo



## 4. WPA2 e gestione delle policy

I tre principali standard crittografici per i meccanismi di sicurezza per reti wireless sono WEP (Wired Equivalent Privacy), WPA (Wi-Fi Protected Access) e WPA2 (Wi-Fi Protected Access 2).

Il primo, rilasciato nel 1999, è considerato insicuro a causa di errori concettuali ed è stato ritirato dal Wi-Fi Alliance nel 2004, con l'introduzione di WPA2 [24].

Il secondo, reso disponibile nel 2003, è un protocollo transitorio che è stato rilasciato per cercare di rimediare alle vulnerabilità scoperte in WEP e richiede lo stesso hardware necessario per l'utilizzazione di WEP. Tuttavia, nel 2008, anche per questo protocollo sono state scoperte alcune vulnerabilità [25].

WPA2, noto anche come IEEE 802.11i-2004 o IEEE 802.11i, è stato invece rilasciato nel 2004 [26]. Attualmente è ritenuto sicuro ma richiede un hardware diverso rispetto a quello necessario per l'utilizzo dei due precedenti.

In questo capitolo ci si focalizzerà sui protocolli WPA2 (sezione 4.1), EAP (sezione 4.2) e RADIUS (sezione 4.3), sulla generazione di certificati X.509v3 con estensioni custom (sezione 4.4) e, infine, in sezione 4.5, verrà mostrata una possibile configurazione di un server FreeRADIUS funzionante per questo lavoro.

### 4.1. Il protocollo WPA2

WPA2 è stato rilasciato in due versioni: PSK (Pre-Shared Keys) destinato all'uso domestico, ed Enterprise, destinato all'uso aziendale.

WPA2-PSK (o anche WPA2-Personal) permette l'autenticazione a un access point della rete attraverso l'utilizzo di una chiave a 256bit generata da una password o da una passphrase.

WPA2-Enterprise (o anche WPA-802.1X mode) richiede l'utilizzo di un server d'autenticazione RADIUS (trattato nella sezione 4.3). Non si fa uso di password o passphrase per l'autenticazione, bensì di certificati X.509 che devono esser installati sui dispositivi client. Questa modalità è considerata maggiormente sicura della precedente in quanto si evitano attacchi di tipo dizionario e garantisce al client l'autenticità dell'access point con cui comunica.

### 4.2. Il protocollo EAP

EAP, Extensible Authentication Protocol, è un framework di autenticazione spesso usato nelle reti wireless e punto-punto [27]. Esso definisce solamente il formato dei messaggi, lasciando ai protocolli che ne fanno uso il compito di definire il modo di incapsularli e trasmetterli.

EAP non fornisce quindi un unico meccanismo di autenticazione, bensì diverse funzioni e meccanismi di autenticazione noti come *metodi* EAP. Essi sono attualmente più di 40; nel seguito

descriveremo quelli maggiormente legati all'utilizzo del protocollo WPA2. Con Android KitKat, sono supportati solamente PEAP, EAP-TLS, EAP-TTLS e EAP-PWD.

### **4.2.1.EAP-TLS**

EAP-Transport Layer Security è uno standard aperto che usa il protocollo TLS e fino al 2005 era l'unico di protocollo EAP adottato dalla Wi-Fi Alliance nei protocolli WPA-Enterprise e WPA2-Enterprise [28].

Questo tipo di autenticazione è supportato da quasi tutte le implementazioni obbligatoriamente attraverso l'utilizzo di certificati X.509. In particolare, ogni client ha un proprio certificato che lo identifica.

Come specificato nel RFC 5280 [29], a partire dalla versione 3, è possibile definire delle proprie estensioni ai certificati X.509. Ognuna deve includere un OID ed una struttura ASN.1, e in ciascun certificato non deve comparire più di un'istanza di ciascuna.

Inoltre, è possibile, per ogni certificato X.509v3, specificare delle estensioni per Certificati di Policy. Un certificato di policy, come riportato nel RFC 3647 [30], "è un insieme di regole che indica l'applicabilità di un certificato a una particolare comunità e/o classe di applicazioni con requisiti di sicurezza comuni".

### **4.2.2.EAP-TTLS**

EAP-Tunneled Transport Layer Security è un'estensione del precedente. In questa versione è semplificato il processo di autenticazione del client presso il server in quanto diviene facoltativo l'utilizzo di un certificato su ciascun client.

Dopo che il server si è autenticato in maniera sicura presso il client attraverso il proprio certificato firmato da una CA, ed eventualmente il client si è autenticato presso il server, il primo può utilizzare la connessione sicura (tunnel) già stabilita per autenticare il client.

### **4.2.3.PEAP**

Protected Extensible Authentication Protocol è un protocollo che incapsula EAP con un tunnel TLS cifrato ed autenticato.

Altri metodi EAP sono descritti in Appendice A.2.

### 4.3. Il protocollo RADIUS

Remote Authentication Dial In User Service (RADIUS) è un protocollo di rete sviluppato nel 1991 dalla Livingstone Enterprises, Inc. che fornisce una gestione centralizzata di AAA (Authentication, Authorization, and Accounting) per gli utenti che si connettono ed usano un servizio di rete [31].

RADIUS implementa principalmente due flussi di processo: uno per autenticazione e autorizzazione, mostrato in Figura 16, l'altro adibito alla profilazione.

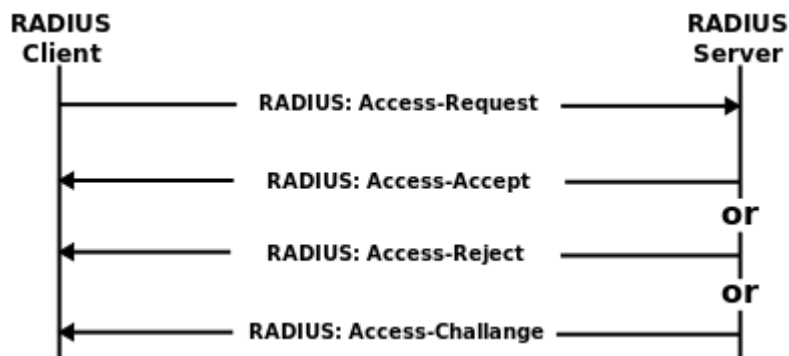


Figura 16: Flusso RADIUS di autenticazione e autorizzazione

Nel primo caso, il client invia al NAS (Network Access Server), per esempio un access point, una richiesta con le proprie credenziali (username e password o un certificato) per ottenere accesso a una risorsa di rete.

Il NAS invia quindi al server RADIUS una richiesta Access-Request, comunicando le credenziali del client ed eventuali ulteriori informazioni.

Il server RADIUS verifica l'autenticità delle credenziali del client attraverso differenti schemi di autenticazioni, fra cui EAP. Quindi invia un messaggio di risposta che può essere:

- *Access Reject*: al client è impedito l'accesso a tutte le risorse cui aveva richiesto accesso. Tipicamente ciò avviene quando le credenziali non sono valide.
- *Access Challenge*: sono richieste ulteriori informazioni dal client, come, per esempio, un'ulteriore password, un PIN, un token o altro.
- *Access Accept*: al client è consentito l'accesso alle risorse per cui aveva fatto richiesta.

Il secondo flusso, descritto in Figura 17, invece, è adibito alla profilazione.

Quando l'accesso alla rete è autorizzato al client dal NAS, questi invia un messaggio *Accounting Start* al server RADIUS contenente le informazioni che permettono di identificare il client da profilare. Altri messaggi *Interim Update* e *Accounting Stop* possono sempre esser inviati dal NAS al a indicare rispettivamente una richiesta di aggiornamento o l'uscita del client dalla rete.

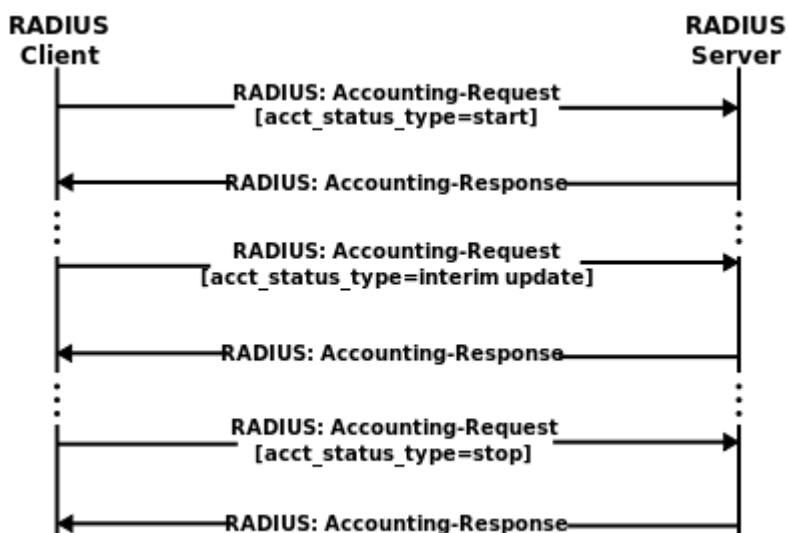


Figura 17: Flusso RADIUS di profilazione

#### 4.4. Generazione di certificati X.509v3

Per la generazione di certificati X.509v3, con estensione custom, è possibile, avendo già il certificato della CA, eseguire un comando di OpenSSL come il seguente:

```
CA -BATCH -KEYFILE CA.KEY -CERT CA.PEM -IN CLIENT.CSR -KEY PASSWORD_CA -OUT CLIENT.CRT -
EXTENSIONS XPCLIENT_EXT -EXTFILE XPEXTENSIONS
```

Con il parametro `-EXTENSIONS` si vanno a specificare quali estensioni aggiungere al certificato, mentre col parametro `-extfile` si specifica in quale file sono specificate le estensioni da aggiungere. Per definire una propria estensione, è sufficiente aggiungere due righe come nel seguente esempio:

```
1.2.3.412=CRITICAL,ASN1:UTF8STRING:MY_VALUE
1.2.3.412=ASN1:UTF8STRING:MY_VALUE
```

In tal modo è stata specificata un'estensione dal nome "1.2.3.412" che ha come valore la stringa UTF8 "MY\_VALUE", come mostra la Figura 18.

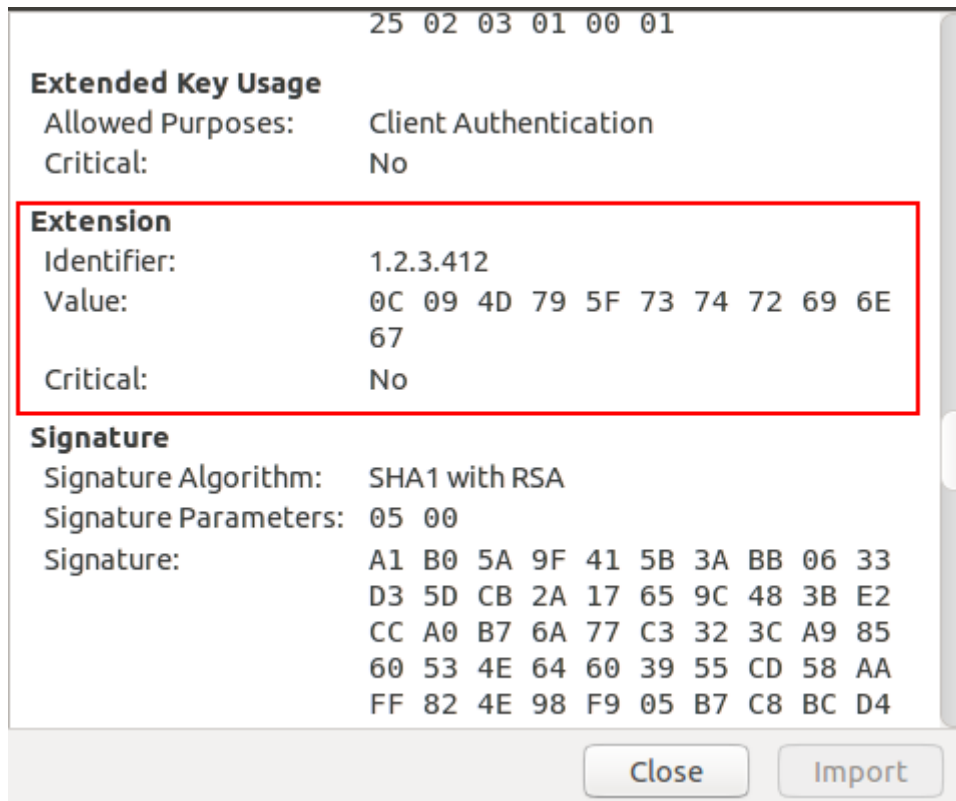


Figura 18: Certificato X.509v3 con custom extension

## 4.5. Sperimentazione con un server FreeRADIUS

Per testare il nostro approccio, abbiamo configurato un server FreeRADIUS con EAP-TLS su macchina Linux-Ubuntu [32]; nel seguito sono riportati i passi di configurazione. Per il corretto funzionamento del server, è stato opportuno installare una versione recente di OpenSSL (non affetta da bug heartbleed).

Una volta installato il server, tramite `APT-GET FREERADIUS` (versione 2), si sono disabilitati i proxy, andando a modificare nel modo seguente il file di configurazione `/etc/FreeRADIUS/radiusd.conf`:

```
PROXY_REQUESTS = NO
# $INCLUDE PROXY.CONF
```

Quindi, si è andato a modificare opportunamente il file `/etc/FreeRADIUS/clients.conf` per specificare il client cui si sarebbe dovuto interfacciare il server come nell'esempio seguente:

```
CLIENT YOUR_ROUTER_NAME {
    IPADDR = YOUR_ROUTER_IP_ADDRESS
    SECRET = RANDOM_SECRET
    REQUIRE_MESSAGE_AUTHENTICATOR = YES
}
```

Si noti che il client, per ciò che riguarda il server RADIUS, è l'access point.

Successivamente, è stato modificato il file `/etc/FreeRADIUS/eap.conf` per poter configurare il metodo eap-tls in maniera opportuna. In particolare, nella sezione `eap` sono stati modificati i seguenti parametri:

```
DEFAULT_EAP_TYPE = TLS
```

```
TLS {  
    CIPHER_LIST = "HIGH"  
    # MAKE_CERT_COMMAND = "${CERTDIR}/BOOTSTRAP"  
    VERIFY {  
        TMPDIR = /VAR/TMP/RADIUSD  
        SCRIPT  
    }  
}
```

SCRIPT è utilizzato per la verifica di certificati X.509v3 con estensione custom. Esso potrà richiamare il comando OpenSSL VERIFY nel seguente modo a verificare la validità del certificato fornito: `VERIFY -CAFILE ${..CA_PATH}/CA.PEM %[TLS-CLIENT-CERT-FILENAME]`. Quindi, in caso positivo, esso dovrà verificare la presenza e il corretto valore dell'estensione custom che deve esser presente nei certificati e che indica a quali policy il dispositivo è conforme. In caso uno dei due controlli fallisca, lo script dovrà restituire un valore diverso da 0, impedendo così l'autenticazione del client.

È possibile commentare gli altri metodi EAP in quanto non utilizzati.

Successivamente, è stata creata, come root, la cartella `/var/tmp/radiusd` coi seguenti comandi:

```
MKDIR /VAR/TMP/RADIUSD  
CHOWN FREERAD /VAR/TMP/RADIUSD  
CHGRP FREERAD /VAR/TMP/RADIUSD  
CHMOD 700 /VAR/TMP/RADIUSD
```

In seguito si sono disabilitati i server virtuali di default:

```
CD /ETC/FREERADIUS/SITES-ENABLED/  
RM *
```

Si è quindi configurato il nuovo server, modificando quello di default, ovvero il file `'default'` all'interno di `/etc/FreeRADIUS/sites-available`.

In particolare, nella sezione `'authorize'` sono stati commentati tutti i parametri a eccezione di `'preprocess'`, `'eap'`, `'expiration'` e `'logintime'`, mentre nella sezione `'authenticate'` è stato commentato tutto tranne `'eap'`.

In seguito è stato abilitato il nuovo server tramite i seguenti comandi:

```
CD /ETC/FREERADIUS/SITES-ENABLED/  
LN -S ../SITES-AVAILABLE/YOUR_AP_NAME_DEFAULT YOUR_AP_NAME_DEFAULT
```

Si è quindi fermato il servizio FreeRADIUS tramite il comando `SERVICE FREERADIUS STOP` e lo si è riavviato in modalità di debug tramite il comando `FREERADIUS -X`.

A questo punto si è proceduto con la rimozione dei certificati di default e con la creazione di quelli nuovi.

Per la rimozione di quelli di default si sono utilizzati i seguenti comandi:

```
CD /ETC/FREERADIUS/CERTS/  
RM *.PEM  
RM *.KEY
```

Quindi si sono copiati in una opportuna cartella i tool per creare quelli nuovi:

```
MKDIR /VAR/CERTS  
MKDIR /VAR/CERTS/FREERADIUS  
CHGRP SSL-CERT /VAR/CERTS/FREERADIUS
```

## WPA2 e gestione delle policy

```
CHMOD 710 /VAR/CERTS/FREERADIUS
CP /USR/SHARE/DOC/FREERADIUS/EXAMPLES/CERTS/* /VAR/CERTS/FREERADIUS/
CD /VAR/CERTS/FREERADIUS/
RM BOOTSTRAP
CHMOD 600 *
```

Con i seguenti comandi si sono poi cancellati tutti gli eventuali tentativi precedenti e inizializzati gli opportuni file:

```
MAKE DESTROYCERTS
MAKE INDEX.TXT
MAKE SERIAL
```

Si sono quindi opportunamente modificati i file di configurazione dei certificati, ovvero 'ca.cnf', 'server.cnf' e 'client.cnf'. In particolare è stata modificata in ognuno la 'output\_password' e, nel caso del client, pure 'emailAddress' e 'commonName' con opportuni valori. Con i seguenti comandi sono stati generati i certificati di CA e server:

```
MAKE CA.PEM
MAKE CA.DER
MAKE PRINTCA
MAKE SERVER.PEM
```

Prima di generare il certificato del client, è stato opportunamente modificato il file 'Makefile' in modo da generare certificati installabili su Android (ovvero in formato *p12*).

```
CLIENT.P12: CLIENT.CRT
    OPENSSL PKCS12 -EXPORT -IN CLIENT.CRT -INKEY CLIENT.KEY -OUT CLIENT.P12 -PASSIN
PASS:${PASSWORD_CLIENT} -PASSOUT PASS:${PASSWORD_CLIENT}
    CP CLIENT.P12 $(USER_NAME).P12

CLIENT.PEM: CLIENT.P12
    OPENSSL PKCS12 -IN CLIENT.P12 -OUT CLIENT.PEM -PASSIN PASS:${PASSWORD_CLIENT} -
PASSOUT PASS:${PASSWORD_CLIENT}
    CP CLIENT.PEM $(USER_NAME).PEM

CLIENT_ANDROID.P12: CLIENT.CRT
    OPENSSL PKCS12 -EXPORT -IN CLIENT.CRT -INKEY CLIENT.KEY -CERTFILE CA.PEM -NAME
"${USER_NAME}" -OUT CLIENT_ANDROID.P12 -PASSIN PASS:${PASSWORD_CLIENT} -PASSOUT
PASS:${PASSWORD_CLIENT}
    CP CLIENT_ANDROID.P12 $(USER_NAME)_ANDROID.P12
```

E quindi sono stati generati i certificati per il client:

```
MAKE CLIENT.PEM
MAKE CLIENT_ANDROID.P12
```

Infine, si sono indicati al server FreeRADIUS i certificati con i comandi nel seguito e si è modificato nuovamente il file 'eap.conf' modificando il campo 'private\_key\_password' con lo stesso valore inserito in '/var/certs/FreeRADIUS/server.cnf' in 'output\_password'.

```
CHMOD 600 *
CHMOD 640 CA.PEM
CHMOD 640 SERVER.PEM
CHMOD 640 SERVER.KEY
CHGRP SSL-CERT CA.PEM
CHGRP SSL-CERT SERVER.PEM
CHGRP SSL-CERT SERVER.KEY
```

## WPA2 e gestione delle policy

```
CD /ETC/FREERADIUS/CERTS/  
LN -S /VAR/CERTS/FREERADIUS/CA.PEM CA.PEM  
LN -S /VAR/CERTS/FREERADIUS/SERVER.PEM SERVER.PEM  
LN -S /VAR/CERTS/FREERADIUS/SERVER.KEY SERVER.KEY
```

Per questo esperimento è stato realizzato un piccolo programma in linguaggio C per la verifica dei certificati che opera come descritto precedentemente.

In Figura 19, è possibile vedere come il server FreeRADIUS si comporti in caso un client si provi ad autenticare con un certificato valido ma privo dell'estensione custom. Si può notare, evidenziato nel primo box rosso, che viene eseguita la verifica del certificato client ma, non essendo trovata l'estensione custom, lo script restituisce 1 e quindi l'autenticazione del client fallisce.

```
[eap] Request found, released from the list  
[eap] EAP/tls  
[eap] processing type tls  
[tls] Authenticate  
[tls] processing EAP-TLS  
[tls] eaptls_verify returned 7  
[tls] Done initial handshake  
[tls] <<< TLS 1.0 Handshake [length 0c66], Certificate  
[tls] chain-depth=1,  
[tls] error=0  
[tls] --> User-Name = user  
[tls] --> BUF-Name = Example Certificate Authority  
[tls] --> subject = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./emailAddress=admin@example.com/CN=Example Certificate Authority  
[tls] --> issuer = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./emailAddress=admin@example.com/CN=Example Certificate Authority  
[tls] --> verify return:1  
[tls] Verifying client certificate: /etc/freeradius/script %(TLS-Client-Cert-Filename) /etc/freeradius/certs/ca.pem  
[tls] expand: %(TLS-Client-Cert-Filename) -> /var/tmp/radiusd/freeradius.client.XXdSkWV3  
Exec-Program output: /var/tmp/radiusd/freeradius.client.XXdSkWV3: OK  
Exec-Program-Wait: plaintext: /var/tmp/radiusd/freeradius.client.XXdSkWV3: OK  
Exec-Program: returned: 1  
rlm_eap_tls: Certificate CN (user@example.com) fails external verification!  
[tls] chain-depth=0,  
[tls] error=0  
[tls] --> User-Name = user  
[tls] --> BUF-Name = user@example.com  
[tls] --> subject = /C=FR/ST=Radius/O=Example Inc./CN=user@example.com/emailAddress=user@example.com  
[tls] --> issuer = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./emailAddress=admin@example.com/CN=Example Certificate Authority  
[tls] --> verify return:0  
[tls] >>> TLS 1.0 Alert [length 0002], fatal certificate_unknown  
TLS Alert write: fatal: certificate_unknown  
TLS accept: error in SSLv3 read client certificate B  
rlm_eap: SSL error error:140890B2:SSL routines:SSL3_GET_CLIENT_CERTIFICATE:no certificate returned  
SSL: SSL_read failed in a system call (-1), TLS session fails.  
TLS receive handshake failed during operation  
[tls] eaptls_process returned 4  
[eap] Handler failed in EAP/tls  
[eap] Failed in EAP select  
++[eap] returns invalid  
Failed to authenticate the user.
```

Figura 19: Autenticazione FreeRADIUS con certificato senza estensione custom



## WPA2 e gestione delle policy

In Figura 20, invece, è possibile vedere come il server FreeRADIUS si comporti nel caso in cui invece il certificato fornito non sia valido. Nel primo box rosso, infatti, è possibile notare come già la verifica del certificato fallisca e che quindi lo script restituisca 2, impedendo quindi l'autenticazione.

```
+- entering group authenticate {...}
[eap] Request found, released from the list
[eap] EAP/tls
[eap] processing type tls
[tls] Authenticate
[tls] processing EAP-TLS
[tls] eap_tls_verify returned 7
[tls] Done initial handshake
[tls] <<< TLS 1.0 Handshake [length 0c60], Certificate
[tls] chain-depth=1,
[tls] error=0
[tls] --> User-Name = user
[tls] --> BUF-Name = Example Certificate Authority
[tls] --> subject = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./emailAddress=admin@example.com/CN=Example Certificate Authority
[tls] --> issuer = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./emailAddress=admin@example.com/CN=Example Certificate Authority
[tls] --> verify return:1
[tls] Verifying client certificate: /etc/freeradius/script %{TLS-Client-Cert-Filename} /etc/freeradius/certs/ca.pem
[tls] expand: %{TLS-Client-Cert-Filename} -> /var/tmp/radiusd/freeradius.client.XXc3S5gr
Exec-Program output: /var/tmp/radiusd/freeradius.client.XXc3S5gr: OK
Exec-Program-Wait: plaintext: /var/tmp/radiusd/freeradius.client.XXc3S5gr: OK
Exec-Program: returned: 2
rlm_eap_tls: Certificate CN (user@example.com) fails external verification!
[tls] chain-depth=0,
[tls] error=0
[tls] --> User-Name = user
[tls] --> BUF-Name = user@example.com
[tls] --> subject = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./CN=user@example.com/emailAddress=user@example.com
[tls] --> issuer = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./emailAddress=admin@example.com/CN=Example Certificate Authority
[tls] --> verify return:0
[tls] >>> TLS 1.0 Alert [length 0002], fatal certificate_unknown
TLS Alert write: fatal: certificate unknown
TLS accept: error in SSLv3 read client certificate B
rlm_eap: SSL error error:140890B2:SSL routines:SSL3_GET_CLIENT_CERTIFICATE:no certificate returned
SSL: SSL_read failed in a system call (-1), TLS session fails.
TLS receive handshake failed during operation
[tls] eap_tls_process returned 4
[eap] Handler failed in EAP/tls
[eap] Failed in EAP select
+-[eap] returns invalid
```

Figura 20: Autenticazione FreeRADIUS con certificato non valido

In Figura 21 e Figura 22, infine, è possibile osservare il comportamento del server FreeRADIUS nel caso in cui il certificato fornito per l'autenticazione sia valido e contenente l'estensione custom. In questo, com'è osservabile nel box rosso in Figura 21, caso lo script restituisce 0, comunicando quindi al server la correttezza del certificato che, a sua volta, consentirà il login (box rosso Figura 22).

```
[eap] Request found, released from the list
[eap] EAP/tls
[eap] processing type tls
[tls] Authenticate
[tls] processing EAP-TLS
[tls] eap_tls_verify returned 7
[tls] Done initial handshake
[tls] <<< TLS 1.0 Handshake [length 0c66], Certificate
[tls] chain-depth=1,
[tls] error=0
[tls] --> User-Name = user
[tls] --> BUF-Name = Example Certificate Authority
[tls] --> subject = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./emailAddress=admin@example.com/CN=Example Certificate Authority
[tls] --> issuer = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./emailAddress=admin@example.com/CN=Example Certificate Authority
[tls] --> verify return:1
[tls] Verifying client certificate: /etc/freeradius/script %{TLS-Client-Cert-Filename} /etc/freeradius/certs/ca.pem
[tls] expand: %{TLS-Client-Cert-Filename} -> /var/tmp/radiusd/freeradius.client.XXTS4Yqi
Exec-Program output: /var/tmp/radiusd/freeradius.client.XXTS4Yqi: OK
Exec-Program-Wait: plaintext: /var/tmp/radiusd/freeradius.client.XXTS4Yqi: OK
Exec-Program: returned: 0
[tls] Client certificate CN user@example.com passed external validation
[tls] chain-depth=0,
[tls] error=0
[tls] --> User-Name = user
[tls] --> BUF-Name = user@example.com
[tls] --> subject = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./CN=user@example.com/emailAddress=user@example.com
[tls] --> issuer = /C=FR/ST=Radius/L=Somewhere/O=Example Inc./emailAddress=admin@example.com/CN=Example Certificate Authority
[tls] --> verify return:1
[tls] TLS accept: SSLv3 read client certificate A
[tls] <<< TLS 1.0 Handshake [length 0046], ClientKeyExchange
[tls] TLS accept: SSLv3 read client key exchange A
[tls] <<< TLS 1.0 Handshake [length 0206], CertificateVerify
[tls] TLS accept: SSLv3 read certificate verify A
[tls] <<< TLS 1.0 ChangeCipherSpec [length 0001]
[tls] <<< TLS 1.0 Handshake [length 0010], Finished
[tls] TLS accept: SSLv3 read finished A
[tls] >>> TLS 1.0 ChangeCipherSpec [length 0001]
[tls] TLS accept: SSLv3 write change cipher spec A
[tls] >>> TLS 1.0 Handshake [length 0010], Finished
```

Figura 21: Autenticazione FreeRADIUS con certificato valido (1)

## WPA2 e gestione delle policy

```
NAS-Port-Type = Wireless-802.11
Connect-Info = "CONNECT 11Mbps 802.11b"
EAP-Message = 0x020800060d00
State = 0xf82fb3ddff27be4614cdfc841bc4518a
Message-Authenticator = 0xf871c7a487e8a650bdfb961dfd5051b
# Executing section authorize from file /etc/freeradius/sites-enabled/asus_router
+- entering group authorize {...}
++[preprocess] returns ok
[eap] EAP packet type response id 8 length 6
[eap] No EAP Start, assuming it's an on-going EAP conversation
++[eap] returns updated
++[expiration] returns noop
++[logintime] returns noop
Found Auth-Type = EAP
# Executing group from file /etc/freeradius/sites-enabled/asus_router
+- entering group authenticate {...}
[eap] Request found, released from the list
[eap] EAP/tls
[eap] processing type tls
[tls] Authenticate
[tls] processing EAP-TLS
[tls] Received TLS ACK
[tls] ACK handshake is finished
[tls] eaptls_verify returned 3
[tls] eaptls_process returned 3
[tls] Adding user data to cached session
[eap] Freeing handler
++[eap] returns ok
Login OK: [user/<via Auth-Type = EAP>] (from client localhost port 0 cli 02-00-00-00-00-01)
# Executing section post-auth from file /etc/freeradius/sites-enabled/asus_router
+- entering group post-auth {...}
++[exec] returns noop
Sending Access-Accept of id 17 to 127.0.0.1 port 45460
MS-MPPE-Recv-Key = 0xfa779ecc824c89ad06ac27862dcc2b9deeb6893d21e28ac9ce79b96b8fb3a956
MS-MPPE-Send-Key = 0x07227c5ac30bc3a58285f7c02fc895cea71135020f268146d3959eb2d86faa6c
EAP-Message = 0x03080004
Message-Authenticator = 0x00000000000000000000000000000000
User-Name = "user"
Finished request 33.
```

Figura 22: Autenticazione FreeRADIUS con certificato valido (2)

## 5. Remote Attestation in Android

In questo capitolo si andranno ad analizzare alcune possibili soluzioni per garantire la *remote attestation* su dispositivi mobili e, in particolare, in dispositivi mobili con sistema operativo Android. Per ottenere tale obiettivo sono necessarie alcune caratteristiche hardware, ovvero un Trusted Execution Environment, che possano garantire alcune proprietà di sicurezza e che sono trattate in sezione 5.1. In sezione 5.2 verrà poi presentata una possibile implementazione in un sistema operativo Android.

### 5.1. Il TEE

Un Trusted Execution Environment è un ambiente di esecuzione sicura che è isolato dal “normale” ambiente di esecuzione dove vengono eseguiti il sistema operativo del dispositivo e le applicazioni, come riportato in Figura 23 [33].

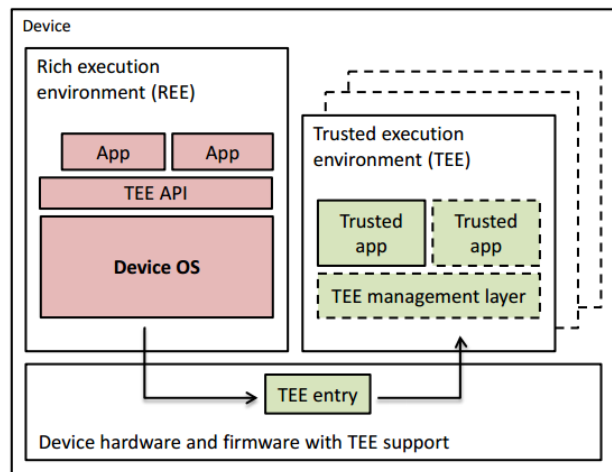


Figura 23: Architettura di un dispositivo mobile con TEE

Nonostante attualmente quasi ogni dispositivo abbia un TEE [21], il loro uso è limitato dagli sviluppatori.

Un TEE, schematizzato in Figura 24, fornisce diverse caratteristiche di sicurezza: esecuzione isolata, salvataggio sicuro, identificazione del dispositivo e autenticazione del dispositivo e integrità della piattaforma [34].

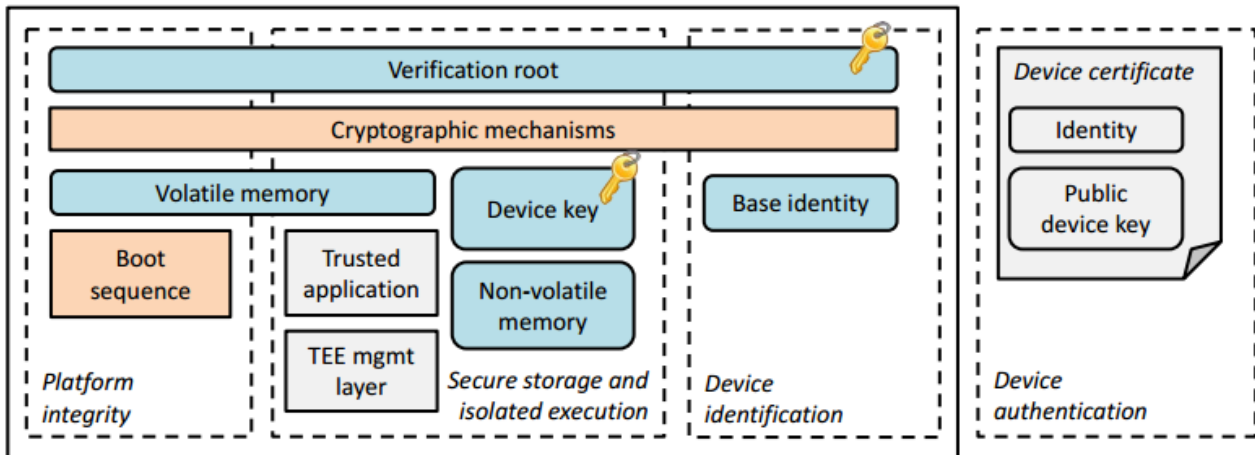


Figura 24: Schema di TEE

L'**esecuzione isolata** [35] permette di eseguire un modulo software in completo isolamento rispetto al resto del codice. Fornisce, inoltre, segretezza e integrità del codice di quel modulo a run-time. Onde evitare che in seguito alla compromissione del sistema operativo anche l'esecuzione isolata venga compromessa, l'ambiente in cui il codice è eseguito in isolamento è indipendente dal sistema operativo e risiede in hardware sottostante o parallelo (come un coprocessore separato).

Il **salvataggio sicuro** [35] fornisce segretezza, integrità e freschezza per i dati di un modulo software a riposo (principalmente quando il dispositivo è spento). Ciò è particolarmente importante nel caso di salvataggio di credenziali come password o chiavi asimmetriche. Ciò può essere realizzato mediante l'utilizzo di una locazione di salvataggio fisicamente protetta, con un controllo degli accessi indipendente dal sistema operativo, definita *RTS: Root of Trust for Storage*. Il RTS può essere utilizzato per caricare un meccanismo più complesso: il *sealed storage*. Quest'ultima utilizza una chiave protetta dal RTS per cifrare i dati e per proteggere l'autenticità loro e di eventuali meta-dati correlati fra cui quelli per le politiche di controllo dell'accesso. I dati così cifrati ("sealed") possono essere salvati anche in locazioni non protette.

La **remote attestation** [35] consente di garantire proprietà di sicurezza come l'identificazione e l'autenticazione del dispositivo. Essa permette a entità remote di verificare che un particolare messaggio sia stato originato da un particolare modulo software. Meccanismi di questo tipo sono tipicamente realizzati con l'ausilio di una chiave privata accessibile esclusivamente da un TCB (Trusted Computing Base) e salvata in una locazione di salvataggio sicura. Un certificato rilasciato da un ente affidabile – tipicamente il costruttore del dispositivo – certifica la corrispondenza fra la chiave pubblica e il dispositivo. La chiave privata è utilizzata per generare attestazioni firmate sullo stato del sistema.

Il **Provisioning sicuro** [35] è un meccanismo che permette di inviare dati a uno specifico modulo software che è eseguito su di uno specifico dispositivo, proteggendo al contempo la segretezza e l'integrità dei dati stessi. Il tipo scenario d'uso è la migrazione o sincronizzazione dei dati utente fra i suoi dispositivi. Tale funzionalità è tipicamente realizzata attraverso la remote attestation.

Il **trusted path** [35] protegge l'autenticità e, opzionalmente, la segretezza e la disponibilità, della comunicazione fra un modulo software e una periferica (come una tastiera o uno schermo touch).

L'**integrità della piattaforma** [34] è assicurata tramite meccanismi di *Secure Boot* e *Authenticated Boot* (o *Trusted Boot*) già trattati nella sezione 2.2.

### 5.2. Remote attestation in Android

Attraverso la remote attestation è possibile determinare se un dispositivo sia in uno stato fidato (*"trust is the expectation that a device will behave in an expected manner for a specific purpose"* [36]) e quindi sia possibile affidargli dati sensibili [37]. Per raggiungere questo scopo è necessario avere a disposizione delle particolari componenti hardware: infatti, non esistono ancora soluzioni esclusivamente software che permettano di raggiungere con certezza questo scopo [38].

A differenza di quanto fatto da Symbian, Android non distingue fra applicazioni fidate e non fidate: esse sono tutte uguali e non è quindi possibile trattarle in modi diversi. La presenza di una VM (che è un eseguibile) su cui sono eseguite le applicazioni, impedisce di poter applicare le tecniche tradizionali di remote attestation. Esse, infatti, comunicherebbero all'attestatore le misurazioni della VM con quelle delle applicazioni eseguite su di essa.

In [37] viene presentata una possibile architettura adottabile su di un dispositivo Android per la remote attestation che è illustrata in Figura 25. Essa si appoggia al PCR (Platform Configuration Registers): una zona di memoria sicura dove vengono scritti gli hash delle misurazioni del sistema.

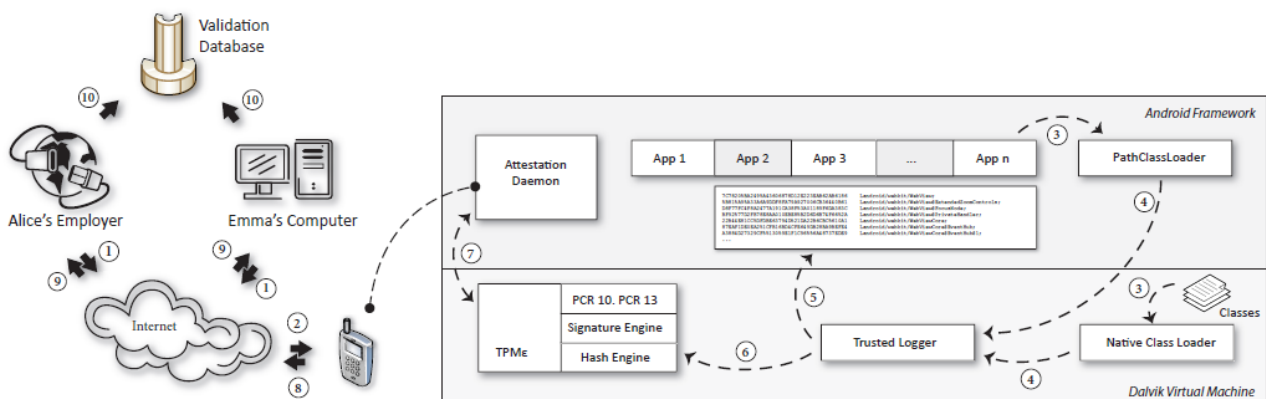


Figura 25: Architettura della soluzione per la remote attestation in Android

La richiesta di attestazione è inviata all'*Attestation Daemon* sul dispositivo, su cui deve essere presente almeno un meccanismo di attestation che continuamente effettua e salva su un log fidato le misurazioni del sistema (applicazioni native e/o in VM). Il demone risponde alla richiesta inviando le misurazioni riportate nel log e quelle salvate dal PCR, che potranno essere utilizzate dall'attestatore per conoscere l'affidabilità del sistema confrontandole con quelle attese.

Per i nostri scopi è di particolare interesse analizzare il meccanismo di attestation a livello applicativo. Quando viene avviata una nuova applicazione, Android cerca le classi di cui necessita e invoca il *PATHCLASSLOADER* per recuperarle dall'APK. Nella soluzione proposta, all'invocazione del

metodo `FINDCLASS()` di `PATHCLASSLOADER` vengono anche effettuate delle misurazioni di integrità sull'intero APK atte a verificarne l'autenticità. In particolare, ne viene calcolato l'hash.

Tuttavia, questo livello di granularità non è sufficiente a fornire un'attestation completa perché non vengono misurate le classi di sistema. Senza una loro misurazione, pertanto, non è garantito che il dispositivo rispetti il meccanismo dei permessi di Android.

Un meccanismo di attestation a livello di tutte le classi, invece, permette di colmare questa lacuna. Nella soluzione proposta in [37], quindi, vengono effettuate opportune operazioni di misura all'interno della funzione di sistema responsabile del caricamento di tutte le classi, comprese quelle di sistema. Nel caso della Dalvik VM (presente fino ad Android KitKat), questa era la `DVMFINDCLASSNOINIT()`. In tal modo è possibile esser sicuri che tutte le classi caricate nella VM vengano misurate prima di esser eseguite.

Le misurazioni da effettuarsi riguardano tutte le informazioni della classe che ne possano influenzare il comportamento a run-time. Esse vengono estratte dalla struttura dati messa a disposizione del sistema in seguito al caricamento della classe stessa. Nel caso della Dalvik VM possono essere suddivise in tre categorie:

- *Meta-informazioni*: pur non influenzando direttamente il comportamento della classe, ne consentono l'identificazione. Fra esse dovrebbero esserci: il *descrittore*, ovvero il nome completo della classe, il corrispondente file dex, il *class loader*, etc.
- *Entità passive*: porzioni statiche della classe che, seppur non eseguibili, potrebbero influenzarne il comportamento. Fra esse: *campi statici*, *nomi di campi*, *nomi di metodo* etc.
- *Codice eseguibile*: la parte più importante della misura, include tutte le istruzioni nel corpo della classe.

## 6. L'applicazione client di BYOD-Cert

Questo capitolo illustra le varie caratteristiche del client che è stato realizzato per questa tesi.

In sezione 6.1 vengono illustrati i passi necessari per recuperare la configurazione, ovvero le applicazioni installate e i loro permessi, del dispositivo Android su cui è eseguita l'applicazione client. La sezione 6.2 mostra come gestire i certificati X.509 sul dispositivo Android. La sezione 6.3 spiega come intercettare l'installazione di un'applicazione in Android mentre in sezione 6.4 è spiegato come monitorare dinamicamente applicazioni arbitrarie tramite *instrumentation* e vengono elencati alcuni strumenti utilizzabili a tale scopo.

### 6.1. Raccolta informazioni sulle applicazioni installate

Per ottenere la lista delle applicazioni attualmente installate sul dispositivo Android, non è necessario richiedere particolari permessi, ma è sufficiente utilizzare la classe di sistema `PackageManager` [39]. Per ottenere un'istanza di tale classe è sufficiente invocare il metodo `getPackageManager()`.

Invocando il metodo `getInstalledPackages()` sull'istanza di `PackageManager`, si ottiene la lista delle applicazioni installate come istanze di `PackageInfo` [40]. Tuttavia, solo alcune delle informazioni da esso messe a disposizione sono effettivamente disponibili in base ai flag passati come parametro alla funzione `getInstalledPackages()`. In particolare, è necessario impostare il flag `PackageManager.GET_PERMISSIONS` per ottenere la lista dei permessi richiesti dall'applicazione.

Maggiore è il numero di flag impostati (e quindi d'informazioni richieste), maggiore è il tempo impiegato dall'applicazione per recuperare l'elenco delle applicazioni.

Le informazioni che invece sono sempre presenti, a prescindere dai flag impostati, e che vengono utilizzate dal client di BYOD-Cert sono il nome dell'applicazione, il nome del package e il numero della versione dell'applicazione.

In Figura 26 è possibile vedere uno screenshot di un'applicazione realizzata per questo lavoro che recupera le informazioni descritte in questa sezione.

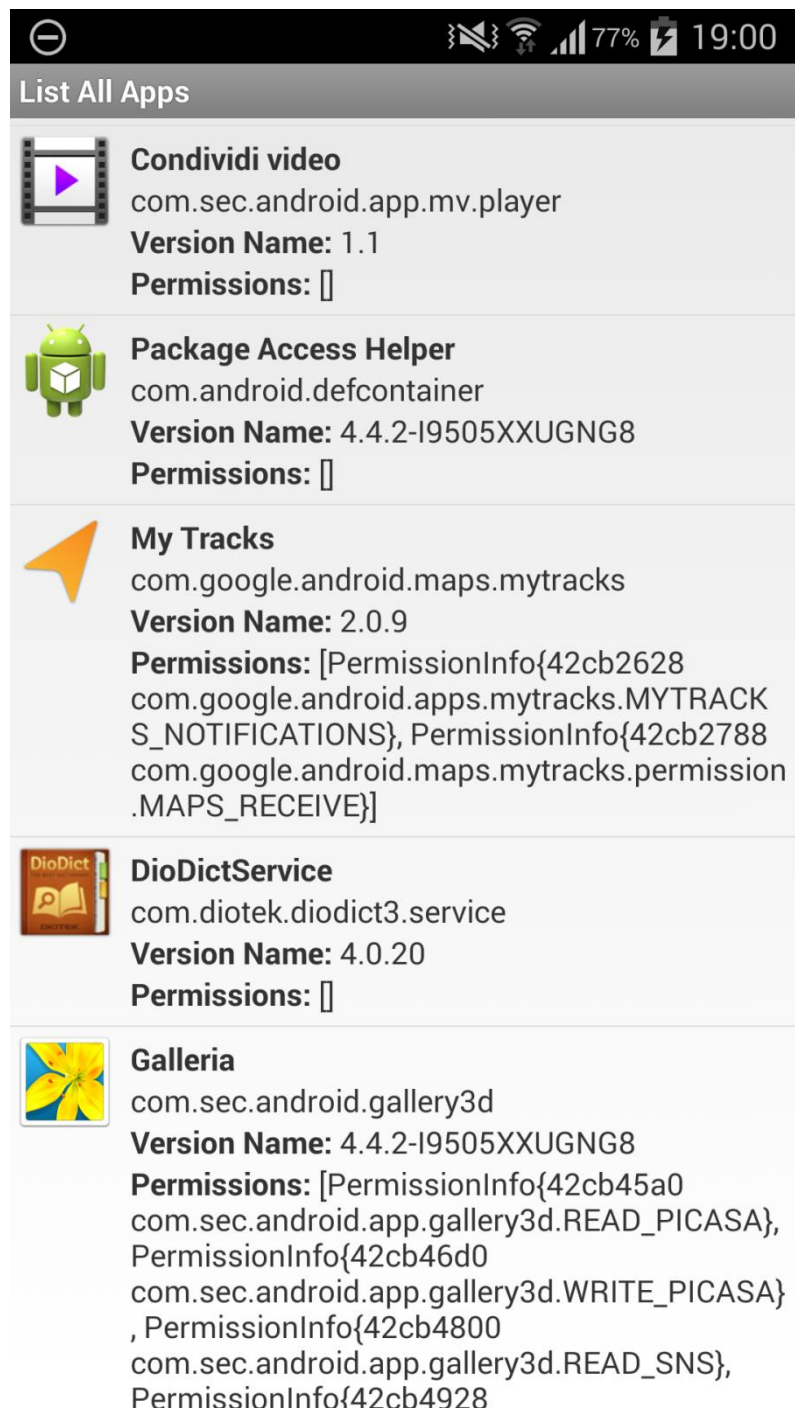


Figura 26: Screenshot applicazione che recupera la configurazione del dispositivo



## 6.2. Gestione certificati su dispositivo Android

Android supporta solamente certificati X.509 in formato DER salvati in file di estensione .crt o .cer e certificati X.509 salvati in file con chiave PKCS#12 con estensione .p12 o .pfx [41].

A partire da Android 4.0 (Ice Cream Sandwich, ICS) viene messa a disposizione degli sviluppatori l'API KEYCHAIN che consente la gestione dei certificati [42].

La classe KEYCHAIN mette a disposizione un metodo factory – `CREATEINSTALLINTENT()` – che restituisce un intent di sistema che effettua il parsing e installa chiavi e certificati. Per installare un file PKCS#12, è necessario leggerlo da un array binario, salvarlo negli extra dell'intent con chiave `EXTRA_PKCS12` e quindi far partire l'activity associata come nell'esempio:

```
Intent intent = KeyChain.createInstallIntent();  
byte[] p12 = readFile("keystore-test.pfx");  
intent.putExtra(KeyChain.EXTRA_PKCS12, p12);  
startActivity(intent);
```

In seguito a tale chiamata, verrà chiesto all'utente di immettere la password necessaria a estrarre password e certificato. In caso di successo, verrà chiesto all'utente di assegnare un nome al certificato, come nello screenshot di Figura 27.

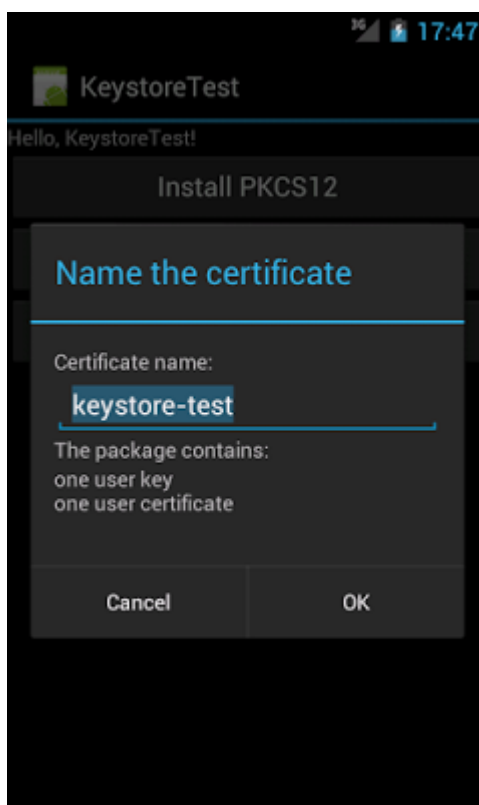


Figura 27: Richiesta di inserimento di un nome per il certificato estratto

Per utilizzare una chiave privata, salvata nello storage delle credenziali di sistema, è necessario chiamare il metodo `KEYCHAIN.CHOOSEPRIVATEALIAS()` specificando gli opportuni parametri, ovvero, nell'ordine, il contesto corrente, la callback da invocare, i formati di chiavi accettabili (RSA, DSA), le entità rilascianti certificati accettabili, l'host e la porta del server richiedente il certificato e l'alias

da preselezionare. In questo modo Android mostrerà all'utente una schermata come quella riportata in Figura 28.

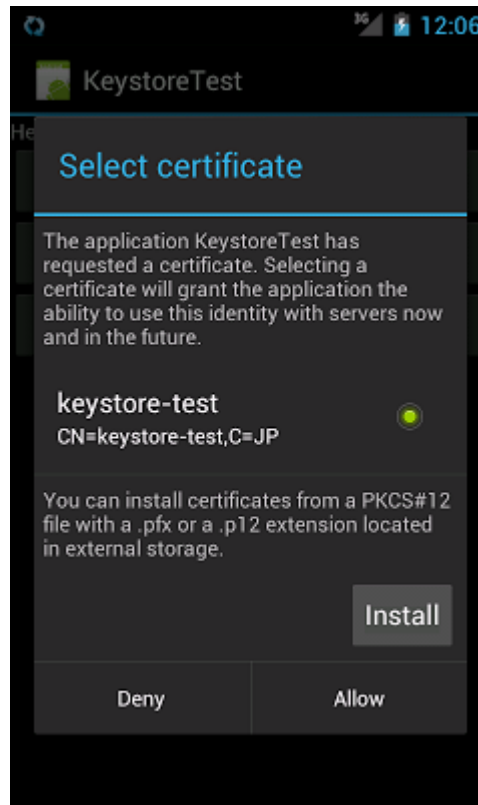


Figura 28: Selezione del certificato

Per verificare che la chiave sia utilizzabile, è sufficiente recuperare la chiave privata col metodo `KEYCHAIN.GETPRIVATEKEY()`, la chain di certificati tramite `KEYCHAIN.GETCERTIFICATECHAIN()` e quindi provare a generare una firma.

Installare un certificato di una CA, è molto simile a quanto visto prima. Analogamente a prima, va letto da un array binario, ma passato come extra all'intent con tag `EXTRA_CERTIFICATE`.

Sebbene sia possibile cancellare i singoli certificati CA, non è possibile (se non rimuovendoli tutti) rimuovere singoli certificati utente.

Per ottenere la lista dei certificati correntemente installati sul dispositivo, si fa uso della API `KEYSTORE` [43]. Il seguente frammento di codice mostra come recuperare le informazioni dei certificati installati:

```
KeyStore ks = KeyStore.getInstance("AndroidCAStore");
ks.load(null, null);
Enumeration aliases = ks.aliases();
while (aliases.hasMoreElements()) {
    String alias = aliases.nextElement();
    X09Certificate cert = (X09Certificate) ks.getCertificate(alias);
    Log.d(TAG, "Subject DN: " + cert.getSubjectDN().getName());
    Log.d(TAG, "Issuer DN: " + cert.getIssuerDN().getName());
}
```

Analizzandone l'output è possibile notare che gli alias dei certificati può iniziare o con 'user:' nel caso di certificati utenti o con 'system:' nel caso di certificati di sistema pre-installati. Per validare un certificato è possibile utilizzare i metodi della classe `TRUSTMANAGERFACTORY` che, tuttavia, non verifica che un certificato sia stato revocato.

La classe di sistema CRL mette a disposizione il metodo `ISREVOKED()` che consente di verificare se un certificato sia stato revocato o meno [44]. Tuttavia, le API attuali non permettono la cancellazione dei singoli certificati utente installati su di un dispositivo Android. È pertanto compito del BYOD-Server revocare i certificati tramite i meccanismi di revoca messi a disposizione dal sistema di certificazione X.509 in tutti gli scenari dettagliati nel corso della sezione 3.2 in cui si prevedeva la cancellazione dei certificati da parte del client.

### 6.3. Monitorare le (dis)installazioni

Affinché la nostra applicazione client possa sapere quando un'altra applicazione viene installata sul dispositivo, è necessario che faccia uso di un `BROADCASTRECEIVER` [45]. Essi sono componenti di Android che permettono di registrarsi a eventi generati dal sistema o dalle applicazioni. Quando un evento viene scatenato, tutti i receiver registrati vengono attivati dal sistema.

Un receiver può essere registrato staticamente, specificandolo nel manifesto dell'applicazione tramite il tag XML `<RECEIVER>`:

```
<receiver android:name="MyReceiver" ></receiver>
```

Per specificare a quali eventi registrarsi è necessario specificare all'interno del tag un `<INTENT-FILTER>`, in cui vengono elencati con una serie di tag `<ACTION>` i vari eventi. In particolare, per la nostra applicazione client, come mostra l'estratto di codice seguente, occorre specificare l'action `ANDROID.INTENT.ACTION.PACKAGE_ADDED` [46]. Analogamente, specificando l'action `ANDROID.INTENT.ACTION.PACKAGE_REMOVED` ci si può registrare per la disinstallazione di applicazioni. L'ulteriore tag `<DATA>`, con valore "package", specifica che si è interessati ai package installati.

```
<receiver android:name="MyReceiver" >
  <intent-filter>
    <action android:name="android.intent.action.PACKAGE_ADDED" />
    <data android:scheme="package" />
  </intent-filter>
</receiver>
```

All'interno del codice Java sarà poi necessario definire una classe di nome `MYRECEIVER` che estenda la classe `BROADCASTRECEIVER` e quindi ridefinire il metodo `ONRECEIVE` nel modo opportuno.

### 6.4. Verifica dinamica

Come visto in sezione 3.2, diversi casi d'uso prevedono la verifica dinamica dell'applicazione tramite dei Reference Monitor (RM) affinché la configurazione del dispositivo su cui essa viene installata possa esser conforme a una certa politica. Ciò in quanto da un'analisi statica del codice dell'applicazione operata dal BYOD-Server non è possibile determinare in maniera assoluta se essa abbia comportamenti non conformi alla policy attesa. Monitorando l'applicazione è quindi possibile in alcuni casi poter eseguire correttamente l'applicazione pur mantenendo il dispositivo

conforme.

I Reference Monitor a livello software, ovvero quelli utilizzati in questo progetto, possono essere a tre differenti livelli: kernel, runtime environment e applicazione. In Figura 29 sono confrontati. Nel primo caso, i controlli sono implementati nelle routine a basso livello e nelle librerie base del sistema operativo. Nel secondo, i controlli sono inseriti nell'ambiente in cui sono eseguite le applicazioni; nel caso di Android nella Virtual Machine Java che esegue il codice (Dalvik o ART a seconda delle versioni di Android). Nell'ultimo caso, invece, i controlli vengono inseriti nel codice dell'applicazione. Se i controlli vengono inseriti successivamente alla compilazione dell'applicazione stessa, si parla di Inlined Reference Monitor (IRM). Per questo progetto vengono considerati questi ultimi.

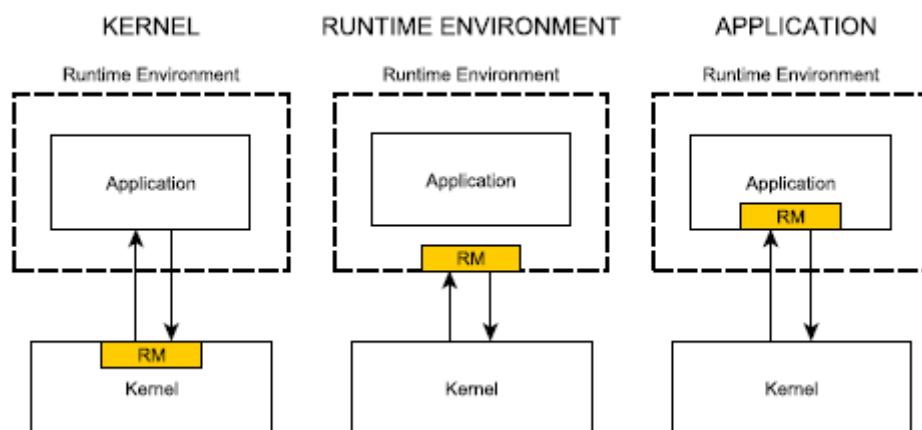


Figura 29: Tipologie di RM software

Sulla Dalvik Virtual Machine, l'instrumentazione di un'applicazione può avvenire secondo due differenti approcci.

Il primo, detto *approccio diretto* e schematizzato in Figura 30, prevede l'inserimento delle istruzioni del RM (in punti detti "PEP": *Policy Enforcement Point*) direttamente prima e/o dopo l'istruzione *invoke*.

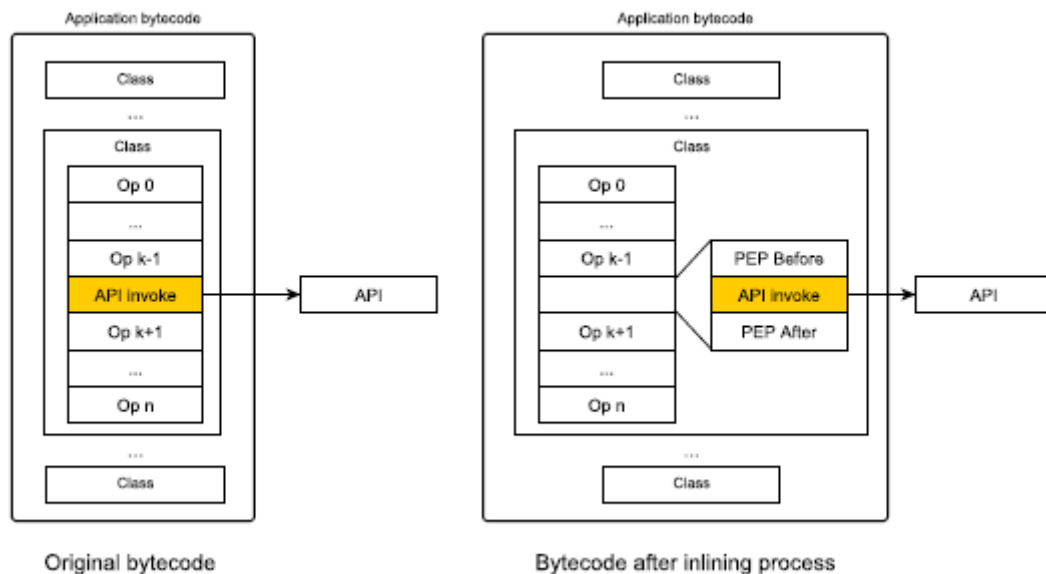


Figura 30: Approccio diretto per instrumentare il bytecode su Dalvik VM

Il secondo approccio, basato su metodi wrapper e schematizzato in Figura 31, prevede la creazione di una libreria esterna di sicurezza che raggruppi le principali operazioni necessarie per l'IRM all'interno di metodi detti *wrapper*.

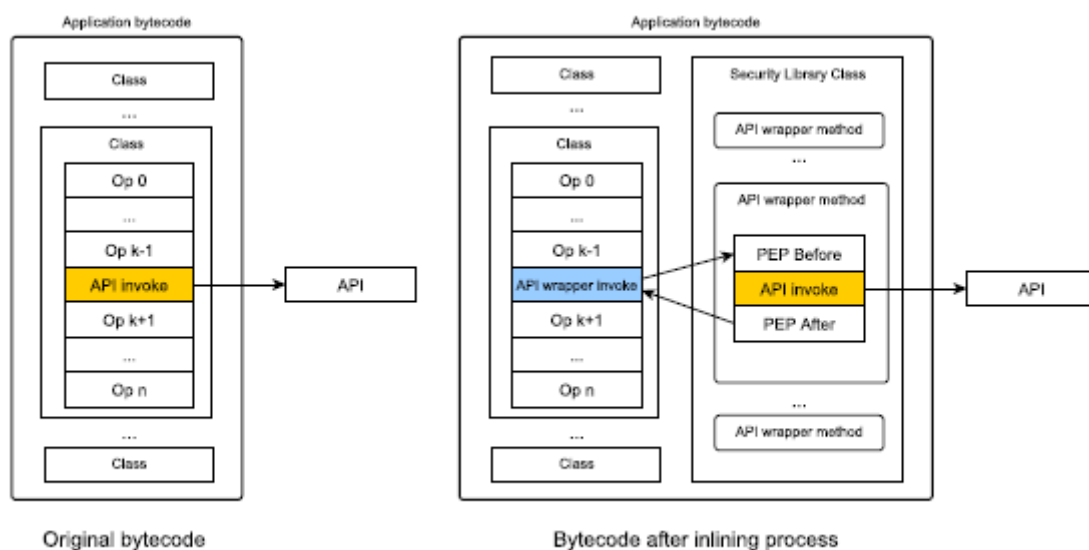


Figura 31: Approccio per instrumentare il bytecode utilizzando i metodi wrapper su Dalvik VM

Per questioni legali, l'instrumentazione non può avvenire lato server in quanto se il server instrumentasse e ri-firmasse delle applicazioni, esso distribuirebbe illegalmente delle applicazioni "non originali". Nel nostro caso, invece, il server invia al client le istruzioni per l'instrumentazione, quindi l'applicazione modificata rimane vincolata all'installazione specifica.

Ci sono diversi tool che permettono d'instrumentare applicazioni Android, in particolare citiamo:

**Redexer** [47] è un framework per l'instrumentazione di bytecode per la Dalvik VM di Android. È composto da un insieme di funzionalità in OCaml che sono in grado di decodificare di un file DEX in

una struttura dati interna alla memoria, di inferire i parametri con cui l'applicazione utilizza certi permessi, di modificare e codificare quella struttura dati per ottenere un nuovo file DEX.

Tuttavia, in quanto OCaml non è supportato nativamente da Android, per l'utilizzo di tale tool sono richiesti programmi esterni per l'esecuzione del codice compilato.

**Soot** [48] è un framework per l'ottimizzazione, l'analisi, l'instrumentazione di bytecode Java. Questo tool, tuttavia, richiede elevati requisiti hardware.

**dexlib2** [49] è una libreria del progetto open source smali/baksmali che mette a disposizione diverse funzionalità per eseguire la conversione delle istruzioni bytecode Dalvik in oggetti manipolabili e viceversa. È compatibile con Android in quanto sviluppata in Java e non richiede eccessivi requisiti hardware. È quindi il tool preferito per questo compito fra quelli proposti in [50].

La parte client d'instrumentazione è stata già trattata in una precedente tesi [50]; ulteriori dettagli sull'instrumentazione e i RM in Android si possono trovare in [51] e [52].

## 7. Conclusioni e lavoro futuro

Questo lavoro, in ambito BYOD, ha affrontato il problema di facilitare la mobilità dei dispositivi Android (*unrooted*) fra aziende diverse. In particolare, permettendo a un dispositivo mobile di esser certificato rispetto a delle politiche BYOD generiche, senza legarsi a una e una sola specifica azienda, come avviene invece per le soluzioni attualmente disponibili sul mercato.

Questo lavoro si è concentrato sulla parte client della soluzione, andando a definire gli scenari operativi e i requisiti hardware e software necessari per garantire il corretto funzionamento. Inoltre, come *proof of concept*, è stato implementato un prototipo che supporta la maggior parte delle funzionalità. La parte server è in fase di sviluppo da parte del gruppo di ricerca presso il Computer Security Lab del DIBRIS [53].

Per l'ingresso del dispositivo alla rete aziendale si è riutilizzato il protocollo WPA2-Enterprise, ma con dei certificati estesi che non solo identificano il client, ma anche le policy. Questo protocollo è spesso già usato all'interno delle rete aziendali, quindi la soluzione proposta potrebbe essere adottata con bassissimo impatto all'interno dei sistemi attuali.

Sviluppi futuri includono:

- implementare il meccanismo di challenge periodici che il MDM-Server dovrebbe inviare al dispositivo, durante la sua permanenza in azienda, atti a verificare il mantenimento di una configurazione conforme alle policy richieste. Questo meccanismo è stato discusso nella tesi ma non implementato nel prototipo.
- migliorare la gestione dei certificati sul dispositivo client, nel momento in cui le API diventassero disponibili. Attualmente, infatti, le API di Android non consentono la rimozione dei singoli certificati utente, rendendo così più difficoltoso non considerare più i certificati per cui il dispositivo non è più conforme.
- Sviluppo di un meta-market che offra all'utente solo le applicazioni, e gli aggiornamenti alle applicazioni già installate, conformi alle policy desiderate (originali o opportunamente instrumentate). Come discusso nella tesi, questo potrebbe esser un problema più legale che tecnico perché richiederebbe la distribuzione di codice non originale (in quanto firmato con una chiave necessariamente diversa da quella dell'autore originale) .





# A. Appendice tecnica

## A.1. WPA2

WPA2, a differenza di WEP e WPA, che usano uno stream cipher RC4, utilizza un block cipher AES.

802.11i utilizza al suo interno diverse componenti: un sistema 802.1x per l'autenticazione (tipicamente il protocollo EAP o un server di autenticazione), il protocollo RSN (Robust Security Network) per tener traccia delle associazioni e i protocolli CCMP (Counter-Mode/CBC Mac Protocol) e, opzionalmente, TKIP (Temporal Key Integrity Protocol) per garantire confidenzialità e integrità dei dati e certezza del mittente.

Il processo di autenticazione avviene tramite un four-way handshake.

### A.1.1. Il Four-Way Handshake

Il meccanismo del four-way handshake prevede che l'access point (AP) si debba autenticare presso il client (STA) e che le chiavi necessarie a cifrare il traffico debbano ancora esser costruite. Attraverso il protocollo EAP (analizzato nella sezione 4.2) o una configurazione WPA2-PSK entrambe le parti condividono la chiave segreta PMK (Pairwise Master Key) che dura per l'intera sessione. Il four-way handshake è inoltre utilizzato per generare un'altra chiave: PTK (Pairwise Transient Key). Essa è generata dalla concatenazione di PMK, ANonce (Access-point Nonce), SNonce (STA Nonce), indirizzo MAC dell'AP e indirizzo MAC del STA e il tutto è utilizzato come parametro di una funzione pseudo-randomica. La PTK, al termine del protocollo, è scomposta in varie sotto-chiavi, che verranno utilizzate nella comunicazione. In particolare da essa sono ricavate la chiave di 16 byte TK (Temporal Key), preposta alla cifratura del traffico unicast fra AP e STA, e la chiave di 16 byte KEK (EAPOL-Key Encryption Key), utilizzata dall'AP per cifrare dati aggiuntivi inviati al client (come, ad esempio, la GTK). Un'ulteriore chiave che viene prodotta durante il four-way handshake è la GTK (Group Temporal Key) che viene usata per decifrare il traffico broadcast e multicast.

In Figura 32 è riportato lo schema di funzionamento del protocollo four-way handshake.

- L'AP invia un proprio nonce (ANonce) al STA.  
Tramite esso, il STA può generare la PTK.
- Il STA invia il proprio nonce (SNonce) all'AP assieme a un MIC (Message Integrity Code) e a un MAIC (Message Authentication and Integrity Code).  
L'AP è quindi in grado di generare la PTK.
- L'AP invia al STA la GTK, una sequenza numerica e un altro MIC. Tale sequenza sarà utilizzata nella successiva comunicazione multicast o broadcast (ovvero, indica il successivo pacchetto cifrato).
- Il STA invia un pacchetto di conferma (ACK) all'AP.

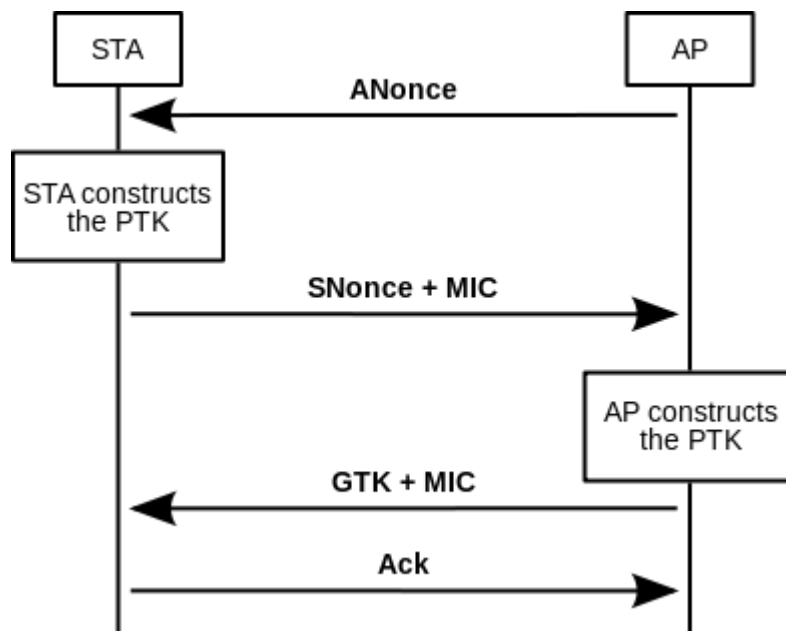


Figura 32: Four-Way handshake in WPA2

### A.1.2. Il Group Key Handshake

La chiave GTK deve essere aggiornata o in seguito alla sua scadenza o in seguito all'abbandono della rete da parte di un dispositivo in modo da evitare che esso continui a ricevere comunicazioni broadcast o multicast.

802.11i definisce il protocollo Group Key Handshake per l'aggiornamento della GTK.

- L'AP invia a ciascun STA della rete la nuova GTK, cifrata con la KEK di ciascun STA, assieme a un MIC.
- Ogni STA manda un ACK all'AP.

## A.2. Il protocollo EAP

### A.2.1. EAP-SIM

EAP per GSM Subscriber Identity Module è utilizzato per la distribuzioni di chiavi di autenticazione e sessione attraverso l'uso di una SIM. EAP-SIM, infatti, utilizza un algoritmo di autenticazione SIM fra il client e il server, fornendo così una mutua autenticazione fra client e rete, evitando la necessità di dover utilizzare password prestabilite.

### A.1.1. EAP-AKA

EAP-AKA è un meccanismo per l'autenticazione attraverso l'uso di una USIM (UMTS-SIM).

### **A.1.2. EAP-FAST**

EAP-Flexible Authentication via Secure Tunnel è un protocollo proposto da CISCO. In esso l'uso di certificati da parte del server è facoltativo. EAP-Fast utilizza un PAC (Protected Access Credential) per stabilire un tunnel TLS in cui le credenziali del client vengono verificate.

### **A.1.3. EAP-PWD**

EAP-PWD è un metodo EAP che risolve il problema di autenticazione basato su scambio di chiavi utilizzando password per l'autenticazione anche deboli per ricavare un segreto condiviso e crittograficamente forte [54].



## B. Indice analitico

- AAA (Authentication, Authorization, and Accounting); 35
- Android Lollipop; 9; 11; 19
- ANonce (Access-point Nonce); 57
- AP (Access Point); 57; 58
- Attestation Daemon; 45
- Authenticated Boot; 16; 45
- Boot loader; 15
- BYOD (Bring Your Own Device); 9; 10; 11; 12; 14; 55
- CA (Certification Authority); 34; 36; 39; 50
- CCM(Client Certificate Management ); 15
- CCMP (Counter-Mode/CBC Mac Protocol); 57
- DAC (Discretionary Access Control); 16
- Dalvik; 46; 52; 53; 54
- dexlib2; 54
- Dual Persona; 11; 13; 14
- EAP (Extensible Authentication Protocol); 10; 33; 34; 35; 38; 57; 58
- EAP-AKA (EAP for Universal Mobile Telecommunications System); 58
- EAP-Fast (EAP Flexible Authentication via Secure Tunneling); 59
- EAP-PWD (EAP with Password); 34; 59
- EAP-SIM (EAP for GSM Subscriber Identity Module); 58
- EAP-TLS (EAP-Transport Layer Security); 34; 37
- EAP-TTLS (EAP-Tunneled Transport Layer Security); 34
- esecuzione isolata; 43; 44
- four-way handshake; 57
- FreeRADIUS; 10; 33; 37; 40; 41
- GCM (Google Cloud Messaging); 21; 24; 30; 32
- GTK (Group Temporal Key); 57; 58
- integrità della piattaforma; 43; 45
- Intent; 32; 49
- IRM (Inlined Reference Monitor); 52; 53
- KEK(EAPOL-Key Encryption Key); 57; 58
- KNOX; 9; 11; 13; 14; 15; 16; 17; 18; 19
- MAC (Mandatory Access Control); 16
- MAIC (Message Authentication and Integrity Code); 57
- MDM (Mobile Device Management); 11; 14; 17
- MIC (Message Integrity Code); 57; 58
- misurazioni; 15; 16; 17; 45; 46
- NAS (Network Access Server); 35
- ODE (On-Device Encryption); 17
- openssl; 36; 37; 38; 39
- PAC (Protected Access Credential); 59
- PCR (Platform Configuration Registers); 45
- PEAP (Protected Extensible Authentication Protocol); 34
- PEP (Policy Enforcement Point); 52
- PMK (Pairwise Master Key); 57
- Provisioning sicuro; 44
- PSK (Pre-Shared Keys); 33
- PTK (Pairwise Transient Key); 57
- QA (Quality Assurance); 18
- RADIUS (Remote Authentication Dial In User Service); 22; 33; 35
- Redexer; 53
- remote attestation; 21; 22; 43; 44; 45
- Remote Workspace; 12
- RKP (Real-time Kernel Protection ); 14
- RM (Reference Monitor); 51; 52; 54
- RSN (Robust Security Network); 57
- RTS (Root of Trust for Storage); 44
- salvataggio sicuro; 43; 44
- SE (Security Enhancements); 16; 17
- SEAMS (SE for Android Management Service); 16
- Secure Boot; 15; 45
- SELinux (Security Enhancements for Linux); 16; 19
- SNonce (STA Nonce); 57
- Soot; 54
- TCB (Trusted Computing Base); 44
- TEE (Trusted Execution Environment); 20; 32; 43
- TIMA (TrustZone-based Integrity Measurement Architecture ); 14; 15; 16
- TK (Temporal Key); 57
- TKIP (Temporal Key Integrity Protocol); 57
- Trusted Boot; 16; 17; 45
- trusted path; 45
- TrustZone; 14; 15; 16

## Indice analitico

USIM (UMTS-SIM); 58

Verified Boot; 15; 19

Virtualization Platform; 12

WEP (Wired Equivalent Privacy); 33; 57

WPA (Wi-Fi Protected Access); 33; 57

WPA2 (Wi-Fi Protected Access 2); 10; 21; 33;  
34; 55; 57; 58

X.509v3; 10; 21; 30; 33; 34; 36; 38; 47; 49; 51

# Indice delle figure

Figura 1: BYODroid .....	13
Figura 2: Architettura di Samsung KNOX .....	14
Figura 3: Funzionamento del TIMA in KNOX .....	15
Figura 4: Secure boot in KNOX.....	15
Figura 5: ODE di KNOX.....	17
Figura 6: Funzioni per MDM in KNOX .....	18
Figura 7: Container di KNOX .....	19
Figura 8: Le parti di Samsung KNOX che vengono integrate in Android L .....	19
Figura 9: Sequence Diagram "Registrazione nuovo client" .....	23
Figura 10: Sequence Diagram "Login" .....	24
Figura 11: Architettura di GCM.....	25
Figura 12: Sequence Diagram "Richiesta di certificazione" .....	26
Figura 13: Sequence Diagram "Installazione nuova applicazione" .....	28
Figura 14: Sequence Diagram "Disinstallazione applicazione" .....	29
Figura 15: Sequence Diagram "Ingresso in azienda" .....	31
Figura 16: Flusso RADIUS di autenticazione e autorizzazione.....	35
Figura 17: Flusso RADIUS di profilazione .....	36
Figura 18: Certificato X.509v3 con custom extension .....	37
Figura 19: Autenticazione FreeRADIUS con certificato senza estensione custom.....	40
Figura 20: Autenticazione FreeRADIUS con certificato non valido .....	41
Figura 21: Autenticazione FreeRADIUS con certificato valido (1) .....	41
Figura 22: Autenticazione FreeRADIUS con certificato valido (2) .....	42
Figura 23: Architettura di un dispositivo mobile con TEE .....	43
Figura 24: Schema di TEE.....	44
Figura 25: Architettura della soluzione per la remote attestation in Android .....	45
Figura 26: Screenshot applicazione che recupera la configurazione del dispositivo .....	48
Figura 27: Richiesta di inserimento di un nome per il certificato estratto .....	49
Figura 28: Selezione del certificato.....	50
Figura 29: Tipologie di RM software .....	52
Figura 30: Approccio diretto per instrumentare il bytecode su Dalvik VM .....	53
Figura 31: Approccio per instrumentare il bytecode utilizzando i metodi wrapper su Dalvik VM ...	53
Figura 32: Four-Way handshake in WPA2 .....	58





# Bibliografia

- [1] Wikipedia, «Bring your own device,» Wikipedia, 22 10 2014. [Online]. Available: [http://en.wikipedia.org/wiki/Bring\\_your\\_own\\_device](http://en.wikipedia.org/wiki/Bring_your_own_device). [Consultato il giorno 10 11 2014].
- [2] R. Ballagas, M. Rohs, J. G. Sheridan e J. Borchers, «BYOD: Bring Your Own Device,» 2005. [Online]. Available: <http://www.vs.inf.ethz.ch/publ/papers/rohs-byod-2004.pdf>. [Consultato il giorno 15 11 2014].
- [3] V. Beal, «BYOD - bring your own device,» Webopedia, [Online]. Available: <http://www.webopedia.com/TERM/B/BYOD.html>. [Consultato il giorno 10 11 2014].
- [4] BYOD Prism, «BYOD Solutions,» [Online]. Available: <http://www.byod-solutions-comparison.com/byod-solutions>. [Consultato il giorno 11 11 2014].
- [5] BYOD Prism, «Mobile Device Management (MDM) Solutions,» [Online]. Available: <http://www.byod-solutions-comparison.com/mobile-device-management>. [Consultato il giorno 11 11 2014].
- [6] BYOD Prism, «Dual Persona Solutions,» [Online]. Available: <http://www.byod-solutions-comparison.com/dual-persona>. [Consultato il giorno 11 11 2014].
- [7] BYOD Prism, «Remote Workspace Solutions,» [Online]. Available: <http://www.byod-solutions-comparison.com/Remote-workspace-solutions>. [Consultato il giorno 11 11 2014].
- [8] BYOD Prism, «Virtualization Platform Solutions,» [Online]. Available: <http://www.byod-solutions-comparison.com/virtualization-platform>. [Consultato il giorno 11 11 2014].
- [9] BYOD Prism, «Profile Manager,» BYOD Solutions Comparison, [Online]. Available: <http://www.byod-solutions-comparison.com/mobile-device-management-mdm-solutions/profile-manager>. [Consultato il giorno 15 11 2014].
- [10] BYOD Prism, «Endpoint Manager for Mobile Devices,» BYOD Solution Comparison, [Online]. Available: <http://www.byod-solutions-comparison.com/mobile-device-management-mdm-solutions/endpoint-manager-mobile-devices>. [Consultato il giorno 15 11 2014].
- [11] A. Armando, G. Costa, L. Verderame e A. Merlo, *Bring Your Own Device, Securely*, 2013.
- [12] BYOD Prism, «Samsung Knox,» BYOD Solutions Comparison, [Online]. Available: <http://www.byod-solutions-comparison.com/dual-persona-solutions/samsung-knox>.

## Bibliografia

[Consultato il giorno 15 11 2014].

- [13] BYOD Prism, «Divide,» BYOD Solutions Comparison, [Online]. Available: <http://www.byod-solutions-comparison.com/dual-persona/divide%E2%84%A2>. [Consultato il giorno 15 11 2014].
- [14] BYOD Prism, «Receiver,» BYOD Solutions Comparison, [Online]. Available: <http://www.byod-solutions-comparison.com/virtual-workspace/receiver%E2%84%A2>. [Consultato il giorno 15 11 2014].
- [15] BYOD Prism, «CloudOn,» BYOD Solutions Comparison, [Online]. Available: <http://www.byod-solutions-comparison.com/virtual-workspace/cloudon%E2%84%A2>. [Consultato il giorno 15 11 2014].
- [16] BYOD Prism, «Horizon Mobile,» BYOD Solutions Comparison, [Online]. Available: <http://www.byod-solutions-comparison.com/virtualization-platform/horizon-mobile%E2%84%A2>. [Consultato il giorno 15 11 2014].
- [17] Samsung, «KNOX Workspace - Technical Details,» Samsung KNOX, 2014. [Online]. Available: <https://www.samsungknox.com/en/products/knox-workspace/technical>. [Consultato il giorno 15 11 2014].
- [18] Samsung, «Remote Verification of Device Integrity Using KNOX Attestation APIs and Cloud Platform,» 3 11 2013. [Online]. Available: <http://samsungdevus.com/sites/default/files/RemoteVerificationofDeviceIntegrityUsingKNOX-Chung.pdf>. [Consultato il giorno 20 11 2014].
- [19] S. Rosenblatt, «Android L builds on Samsung's Knox fortifications,» Cnet, 22 07 2014. [Online]. Available: <http://www.cnet.com/news/android-l-builds-on-samsungs-knox-fortifications/>. [Consultato il giorno 11 11 2014].
- [20] Google, Inc, «A sweet Lollipop, with a kevlar wrapping: New security features in Android 5.0.,» 28 10 2014. [Online]. Available: <http://officialandroid.blogspot.it/2014/10/a-sweet-lollipop-with-kevlar-wrapping.html>. [Consultato il giorno 19 11 2014].
- [21] J.-E. Ekberg, K. Kostinen e N. Asokan, «The Untapped Potential of Trusted Execution Environments on Mobile Devices,» *IEEE Security & Privacy magazine*, n. 12:4, pp. 29-37, 2014.
- [22] Google, Inc, «Overview,» [Online]. Available: <https://developer.android.com/google/gcm/gcm.html>. [Consultato il giorno 20 11 2014].
- [23] Google, Inc, «GCM Advanced Topics,» [Online]. Available: <https://developer.android.com/google/gcm/adv.html>. [Consultato il giorno 20 11 2014].

## Bibliografia

- [24] Wikipedia, «Wired Equivalency Privacy,» Wikipedia, 01 08 2014. [Online]. Available: [http://en.wikipedia.org/wiki/Wired\\_Equivalent\\_Privacy](http://en.wikipedia.org/wiki/Wired_Equivalent_Privacy). [Consultato il giorno 26 01 2015].
- [25] Wikipedia, «Wi-Fi Protected Access,» Wikipedia, 07 01 2015. [Online]. Available: [http://en.wikipedia.org/wiki/Wi-Fi\\_Protected\\_Access](http://en.wikipedia.org/wiki/Wi-Fi_Protected_Access). [Consultato il giorno 26 01 2015].
- [26] Wikipedia, «IEEE 802.11i-2004,» Wikipedia, 30 09 2014. [Online]. Available: [http://en.wikipedia.org/wiki/IEEE\\_802.11i-2004](http://en.wikipedia.org/wiki/IEEE_802.11i-2004). [Consultato il giorno 01 12 2014].
- [27] Wikipedia, «Extensible Authentication Protocol,» Wikipedia, 21 11 2014. [Online]. Available: [http://en.wikipedia.org/wiki/Extensible\\_Authentication\\_Protocol](http://en.wikipedia.org/wiki/Extensible_Authentication_Protocol). [Consultato il giorno 02 12 2014].
- [28] Wikipedia, «EAP extensions under WPA and WPA2 Enterprise,» Wikipedia, 02 12 2014. [Online]. Available: [http://en.wikipedia.org/wiki/Wi-Fi\\_Protected\\_Access#EAP\\_extensions\\_under\\_WPA\\_and\\_WPA2\\_Enterprise](http://en.wikipedia.org/wiki/Wi-Fi_Protected_Access#EAP_extensions_under_WPA_and_WPA2_Enterprise). [Consultato il giorno 02 12 2014].
- [29] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk, NIST, Microfost, T. C. Dublin e Entrust, «Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,» IETF, 05 2008. [Online]. Available: <http://tools.ietf.org/html/rfc5280>. [Consultato il giorno 02 12 2014].
- [30] S. Chokhani, Orion Security Solutions, Inc., W. Ford, VeriSign, Inc., R. Sabett, Cooley Godward LLP, C. Merrill, McCarter & English, LLP, S. Wu e Infoliance, Inc., «Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework,» IETF, 11 2003. [Online]. Available: <http://tools.ietf.org/html/rfc3647>. [Consultato il giorno 07 12 2014].
- [31] Wikipedia, «RADIUS,» Wikipedia, 30 11 2014. [Online]. Available: <http://en.wikipedia.org/wiki/RADIUS>. [Consultato il giorno 02 12 2014].
- [32] T. Hruska, «Setting up WPA2-Enterprise + AES with Ubuntu 12.04.2 + FreeRADIUS with EAP-TLS only (The Definitive Guide),» Cubicspot, 07 04 2013. [Online]. Available: <http://cubicspot.blogspot.it/2013/04/setting-up-wpa2-enterprise-aes-with.html>. [Consultato il giorno 26 01 2015].
- [33] Sigsac, «Trusted Execution Environments on Mobile Devices,» 2013. [Online]. Available: <http://www.sigsac.org/ccs/CCS2013/tutorials/index.html#tee>. [Consultato il giorno 18 12 2014].
- [34] J.-E. Ekberg, K. Kostiainen e N. Asokan, «Trusted Execution Environments on Mobile Devices,» 2013. [Online]. Available: <http://www.cs.helsinki.fi/group/secures/CCS-tutorial/tutorial-slides.pdf>. [Consultato il giorno 18 12 2014].

## Bibliografia

- [35] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome e J. M. McCune, «Trustworthy Execution on Mobile Devices: What security properties can my mobile platform give me?,» 2012. [Online]. Available: <https://sparrow.ece.cmu.edu/group/pub/paper-hyperphone-TRUST-2012.pdf>. [Consultato il giorno 08 01 2015].
- [36] Trusted Computing Group, «TCG Specification Architecture Overview v1.2, page 11-12. Technical report,» 2004.
- [37] M. Nauman, S. Khan, X. Zhang e J.-P. Seifert, «Beyond Kernel-level Integrity Measurement: Enabling Remote Attestation for the Android Platform,» [Online]. Available: <http://profsandhu.com/zhang/pub/trust10-android.pdf>. [Consultato il giorno 26 11 2014].
- [38] C. Castelluccia, A. Francillon, D. Perito e C. Soriente, «On the Difficulty of Software-Based Attestation of Embedded Devices,» 09 13 2009. [Online]. Available: [http://www.inrialpes.fr/planete/people/francill/Papers/CCS09\\_Software\\_based\\_attestation.pdf](http://www.inrialpes.fr/planete/people/francill/Papers/CCS09_Software_based_attestation.pdf). [Consultato il giorno 26 11 2014].
- [39] Android, Inc, «PackageManager,» Android Developers, 09 01 2015. [Online]. Available: <http://developer.android.com/reference/android/content/pm/PackageManager.html>. [Consultato il giorno 12 01 2015].
- [40] Android, Inc, «PackageInfo,» Android Developers, 09 01 2015. [Online]. Available: <http://developer.android.com/reference/android/content/pm/PackageInfo.html>. [Consultato il giorno 12 01 2015].
- [41] Google, Inc, «Work with certificates,» [Online]. Available: <https://support.google.com/nexus/answer/2844832?hl=en>. [Consultato il giorno 27 01 2015].
- [42] N. Elenkov, «Using the ICS KeyChain API,» Android Explorations, 29 11 2011. [Online]. Available: <http://nelenkov.blogspot.it/2011/11/using-ics-keychain-api.html>. [Consultato il giorno 27 01 2015].
- [43] N. Elenkov, «ICS Trust Store Implementation,» Android Explorations, 02 12 2011. [Online]. Available: <http://nelenkov.blogspot.it/2011/12/ics-trust-store-implementation.html>. [Consultato il giorno 27 01 2015].
- [44] Google, Inc, «CRL,» Android Developers, 21 01 2015. [Online]. Available: <http://developer.android.com/reference/java/security/cert/CRL.html>. [Consultato il giorno 27 01 2015].
- [45] Android, Inc,, «BroadcastReceiver,» 19 02 2015. [Online]. Available: <http://developer.android.com/reference/android/content/BroadcastReceiver.html>. [Consultato il giorno 25 02 2015].

## Bibliografia

- [46] Android, Inc, «Intent,» 19 02 2015. [Online]. Available: [http://developer.android.com/reference/android/content/Intent.html#ACTION\\_PACKAGE\\_ADDED](http://developer.android.com/reference/android/content/Intent.html#ACTION_PACKAGE_ADDED). [Consultato il giorno 25 02 2015].
- [47] PLUM, «Redexer,» 19 10 2012. [Online]. Available: <http://www.cs.umd.edu/projects/PL/redexer/>. [Consultato il giorno 27 01 2015].
- [48] Sable Research Group, «Soot,» [Online]. Available: <http://sable.github.io/soot/>. [Consultato il giorno 18 02 2015].
- [49] J. F. a. B. Gruver, *smali - an assembler/disassembler for*, 2014.
- [50] D. Fiori e F. Brizzolara, *Inlined Reference Monitor per piattaforme Android*, Genova, 2014.
- [51] Reddy, Nikhilesh e e. al., «Application-centric security policies on unmodified Android,» *UCLA Computer Science Department, Tech. Rep 110017*, 2011.
- [52] B. e. a. Davis, «I-arm-droid: A rewriting framework for in-app reference monitors for android applications.,» *Mobile Security Technologies*, 2012.
- [53] DIBRIS, «Computer Security Lab,» [Online]. Available: [www.csec.it](http://www.csec.it). [Consultato il giorno 25 02 2015].
- [54] D. Harkins, Aruba Networks, G. Zorn e Network Zen, «Extensible Authentication Protocol (EAP) Authentication Using Only a Password,» IETF, 08 2010. [Online]. Available: <https://tools.ietf.org/html/rfc5931>. [Consultato il giorno 19 01 2014].
- [55] Google, Inc, «Identifying App Installations,» 30 03 2011. [Online]. Available: <http://android-developers.blogspot.in/2011/03/identifying-app-installations.html>. [Consultato il giorno 20 11 2014].
- [56] N. Elenkov, *Android Security Internals*, San Francisco: no starch press, 2015.
- [57] D. Simon, B. Aboba, R. Hurst e M. Corporation, «The EAP-TLS Authentication Protocol,» IETF, 04 2008. [Online]. Available: <http://tools.ietf.org/html/rfc5216>. [Consultato il giorno 02 12 2012].
- [58] C. Rigney, S. Willens, Livingstone, A. Rubens, Merit, W. Simpson e Daydreamer, «Remote Authentication Dial In User Service (RADIUS),» IETF, 06 2000. [Online]. Available: <http://tools.ietf.org/html/rfc2865>. [Consultato il giorno 05 12 2014].