UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA IN INFORMATICA

ANALISI, PROGETTAZIONE E REALIZZAZIONE DI UN EMULATORE DI TRUSTED COMPUTING PLATFORM

Relatore: Prof. Danilo BRUSCHI
Correlatore: Ing. Mattia MONGA

Tesi di Laurea di:
Lorenzo CAVALLARO
Matricola 529863

Anno Accademico 2003–04

Indice

A Simona,
il mio grande, unico ed eterno amore

e ai miei genitori,
per avermi donato la vita

Ringraziamenti

Sinceramente non avrei mai pensato di arrivare[1] a questo primo traguardo, quello della laurea. Quando mi sono iscritto a questo corso di laurea[2] non ero molto interessato a ciò che invece, ora, penso possa diventare parte della mia vita.

Ci sono molte persone che ho incontrato durante questi anni trascorsi in università; persone che vorrei ringraziare per avermi accompagnato in questo cammino e per avermelo fatto pesare, spesso, molto meno (palla di fuoco!!!). Ma andiamo per ordine (nella mia mente, non ti importanza, chiaro?! :-))

Desidero ringraziare il mio relatore, il Prof. Danilo Bruschi, sia per avermi dato l'opportunità di lavorare in piena libertà a questa tesi, ma sopratutto per aver saputo creare, insieme alla Prof.ssa Rosti, un certo rapporto che mi rendo conto non è fortuna di tutti gli studenti avere. Allo stesso modo ringrazio il mio correlatore, l'Ing. Mattia Monga, che ha saputo/voluto dimostrarsi non solo un fantastico mentore, ma anche un amico.

Un grazie di cuore ad Andrea Lanzi, fedele compagno di tesi e di avventure "lavorative". Grazie anche a tutto il LaSeR (che nome l4m3r scritto così, non vi pare?) e in particolare a Lorenzo, Igor N. F., gianz, Julian, Alberto, Emanuele e a tutti gli altri che non ho nominato (chiedo scusa!): non ce l'avrei fatta senza di voi.

Ringrazio gli ORGOGLIONI[3] per non avermi mai fatto sentire solo e con i quali

[1] qualcuno di voi dirà *finalmente*.

[2] laurea quinquennale in informatica, nel lontano a.a. 1995/1996, se ricordo bene.

[3] . . . e indirettamente Marco della Noce, che ha ispirato, goliardicamente tutti noi.

mi sono divertito (e continuerò a farlo). Conserverò con amore i magici momenti immortalati dalla telecamera!

Grazie alle vecchie leve: Andrea il “Tama”, Dario Cif., Lele il “Nonnu” , Andrea l’“Omone”, Davide il “Dino”, Ricky, Igor C., ndetof, Roby, Beppe G., Gloria, Beppe A., Bacci, Dario Cro., Diego, Enzo M., TooMuch, Paolo “sponge”, Aldo, (Dema), Vale, Ale, Betta, Popi, Zimon, Fede, ma anche alle “nuove”: Carlitos, Thomaz, Linda, Orlando, Tinybyte. Chiedo venia se ho dimenticato qualcuno (vedere sotto per un bell’alibi! :))

Grazie ad Alberto “ALoR” Ornaghi, per avermi insegnato molto e grazie anche alle crew, *Antifork* e *s0ft project* della “scena” italiana: ho imparato tanto da voi e sono orgoglioso di conoscervi. Spero di potervi frequentare più spesso, ora . . . :)

Non potrei certo dimenticare i miei amici di *Campagnola Emilia*, “capitanati” da Jorè, con i quali mi sono divertito e che mi hanno fatto sentire come a casa.

Un ringraziamento particolare va a mio zio Roberto e a mia zia Anna, che mi hanno permesso, in un certo senso, di cambiare vita. Grazie di cuore.

Grazie ai miei cugini, Paolo, Anna, Giovanni e zii annessi :) (zia Marina, zia Tina e zio Giulio) perché, anche se non ci frequentiamo spesso, mi sento sempre a mio agio con voi . . .

Ringrazio il mio padrino (no, non di “cosa nostra” :)) Gianni, la sua famiglia, la Milena, il Lino (e famiglia), la Piera e il Giuliano per avermi sempre aiutato nel momento del bisogno.

Grazie Simona, amore mio, perché mi sopporti in continuazione, perché mi dai sempre conforto e credi in me, e perché non mi fai mai mancare niente, sorpratutto il tuo amore. Sei la mia felicità.

Un grazie particolare ed un ricordo affettuoso va ai miei nonni.

Prima di congedarmi, vorrei chiedere scusa a tutte le persone che non ho citato esplicitamente; sono le 1:30 AM e domani, se tutto va bene, devo andare a portare la tesi in copisteria . . . può andar bene come alibi?! :)

Infine, vorrei ringraziare il Signore, per essermi sempre stato vicino e per avermi donato tutto ciò che sono e i miei genitori che, anche se non lo sanno o non lo pensano, *forse* qualcosa di buono l’hanno fatto insieme. Me.

Riassunto

Il lavoro di tesi analizza le cosidette *piattaforme di calcolo fidate* (Trusted Computing Platform, TP). La comprensione di questa tecnologia ha portato alla progettazione e realizzazione di un emulatore software per tale architettura che ha permesso di analizzarne la sicurezza individuando un attacco, precedentemente sconosciuto.

Le TP sono piattaforme di calcolo che introducono una nuova proprietà, quella di fiducia, che si traduce nella possibilità di verificare che i componenti hardware e software della TP si comportino secondo le loro specifiche progettuali.

Al fine di garantire questa proprietà, l'architettura introduce nuovi componenti hardware *assiomaticamente* fidati, detti *root of trust*, che rappresentano le fondamenta sulle quali costruire induttivamente tutto il processo di fiducia. In questo modo è possibile stabilire, ad esempio, se un processo di boot, un sistema operativo o un insieme di funzionalità di sicurezza della piattaforma sono stati propriamente "installati" e operano nel modo che ci si aspetta.

Questa nuova tecnologia dovrebbe trovare impiego in diversi settori dell'informatica quali autonomic computing, grid computing, on demand computing, e-commerce, in cui è necessaria la verifica della proprietà di fiducia, introdotta dalle Trusted Platform, che deve instaurarsi tra la piattaforma e gli utilizzatori della stessa.

Allo scopo di migliorare la conoscenza dell'architettura[4] proposta dal *Trusted Com-*

[4] architettura *aperta*; molti ricercatori possono apportare il loro know how, contribuire attivamente e capire a fondo la specifica.

puting Group[5], è stato progettato e realizzato un emulatore software, sviluppato su piattaforma ia32, sistema operativo GNU/Linux.

La comprensione approfondita dei protocolli alla base del funzionamento della piattaforma, ha permesso, grazie anche all'uso di tecniche di *model checking*, di identificare un attacco di tipo *reply* che permette di sovvertire le caratteristiche legate alla sicurezza di tali protocolli.

[5] *conosciuto precedentemente come Trusted Computing Platform Alliance, TCPA.*

Introduzione

Lo scopo di questo lavoro di tesi è stato quello di studiare ed analizzare una nuova tecnologia di sicurezza, le cosidette *piattaforme di calcolo fidate* (Trusted Computing Platform, TP), descritte secondo la specifica definita dal Trusted Computing Group[6] (TCG).

Le minacce contro la sicurezza delle informazioni sono reali e crescenti e, a tutt'oggi, le infrastrutture di calcolo mancano di un metodo di difesa univoco ed economico.

Le TP nascono quindi, con l'obiettivo di *coadiuvare* i meccanismi di sicurezza esistenti, fornendo una proprietà aggiuntiva, quella di *fiducia*, al fine di poter garantire per il corretto funzionamento della piattaforma di calcolo. Infatti, uno dei problemi più grandi della sicurezza dei sistemi di calcolo, rimane quello di stabilire in modo sicuro e inopinabile, ad esempio, che il software in esecuzione su una piattaforma sia inalterato, genuino e si comporti nel modo per il quale è stato progettato, ovvero, in altre parole, determinare l'integrità, tramite l'uso di funzioni di hash crittografiche, dell'ambiente software di una piattaforma.

Grazie all'introduzione di componenti hardware, le *root of trust* considerate assiomaticamente fidate[7], le TP definiscono dei meccanismi attraverso i quali è possibile identificare in modo sicuro e univoco la piattaforma coinvolta nella comunicazione (concetto di identità di una TP), determinare l'integrità del suo ambiente software

[6] conosciuto precedentemente come Trusted Computing Platform Alliance, TCPA.

[7] genuine, perché attestate da terze parti fidate per mezzo di certificati digitali, solitamente.

(concetto di misurazione di metriche d'integrità) e fornire un portale crittografico per la protezione di dati sensibili (concetto di protected storage).

Il concetto di fiducia, qui introdotto, è riferito alla capacità, per una TP, di garantire per la correttezza e l'inalterabilità dei dati prodotti da essa. Così, sebbene rimanga compito degli utilizzatori determinare se un dato ambiente software possa rappresentare, o meno, un sufficiente livello di sicurezza, le TP garantiscono che le misurazioni d'integrità effettuate sono inalterate e si riferiscono proprio a quell'ambiente software, di una determinata piattaforma.

Il lavoro di tesi si articola in tre parti principali. Nella prima parte si introducono le piattaforme di calcolo fidate, evidenziando però, prima lo scenario nel quale si trovano ad operare (capitolo 1), e poi descrivendo le necessità, le caratteristiche architetturali (capitolo 2) e le innovazioni funzionali (capitolo 3) che esse introducono, al fine di coadiuvare i meccanismi di sicurezza esistenti.

La seconda parte si focalizza sulla progettazione e implementazione di un emulatore software dell'architettura TP proposta da TCG. Il capitolo 4 definisce le motivazioni che hanno spinto alla progettazione di un'emulatore e la sua architettura generale. Il capitolo 5 individua i componenti principali e le interazioni esistenti tra essi, mentre l'ultimo capitolo di questa parte, descrive la progettazione di uno dei componenti più importanti dell'intera architettura, la root of trust TPM.

Infine, la terza parte, dopo una breve introduzione ai protocolli di autorizzazione usati dall'architettura al fine di permettere di effettuare operazioni "sensibili", affronta l'analisi di sicurezza di uno questi, l'OIAP, definito dalla specifica TCG, provando la sua "resistenza" (o meno) ad attacchi di tipo *reply* e *man in the middle*.

Parte I

Introduzione alle Trusted Computing Platform

Capitolo 1

Il problema della sicurezza informatica

Le piattaforme di calcolo sono diventate ormai parte centrale e fondamentale per la crescita del business e del commercio elettronico. Privati, imprese e pubbliche amministrazione utilizzano sempre di più i vari strumenti informatici, spesso spinti proprio da questi "nuovi" modelli commerciali. La necessità di offrire protezione ai dati sensibili[1] *e alle piattaforme coinvolte in queste transazioni è aumentata notevolmente in questi ultimi anni, tanto da spingere i ricercatori a progettare nuove architetture dotate di proprietà sempre più importanti, quali la fiducia nella piattaforma di calcolo stessa; fiducia che la piattaforma agisca nel modo per il quale è stata progettata.*

Il grado di fiducia nelle soluzioni di sicurezza basate solo sull'impiego di software, dipende dalla loro corretta installazione ed esecuzione; anche il software più "robusto" e altamente controllato non può attestare la sua stessa integrità.

Nel capitolo che segue, si evidenziano le minacce che minano la sicurezza dei sistemi informatici e le contromisure attualmente impiegate per contrastarle; queste non sono sufficienti a garantire la sicurezza di una piattaforma perché non esistono meccanismi in grado di garantirne il corretto funzionamento, capaci di fornire una proprietà di fiducia *nella piattaforma di calcolo.*

[1] numero di carta di credito, passphrase per effettuare bonifici bancari, etc.

1.1 Minacce alla sicurezza

Le minacce contro la sicurezza delle informazioni sono reali e crescenti e, a tutt'oggi, le infrastrutture di calcolo mancano di un metodo di difesa univoco ed economico.

È possibile suddividere le minacce alla sicurezza in grandi, generiche categorie[2]:

- Buffer Overflow[3],
- Virus, Worm e Trojan Horse[4],
- Manomissione del software e pirateria,
- Furto di dati, software o hardware,
- Accessi non autorizzati,
- Ripudio nell'aver effettuato un'operazione o transazione,
- Denial of Service

Le misure di sicurezza che possono essere adottate per prevenire o ridurre al minimo queste minacce comprendono:

- Autenticazione,
- Controllo degli accessi,
- Confidenzialità dei dati,
- Integrità dei dati,
- Non ripudio

[2]la trattazione di queste minacce non verrà affrontata in questo lavoro. Si rimanda alla copiosa letteratura esistente; una buona introduzione può essere fornita da [27] e [21].

[3]vulnerabilità attraverso la quale si cerca di sovvertire il normale flusso di esecuzione di un processo. Basati sullo stack e sullo heap/bss.

[4]spesso sfruttano vulnerabilità di buffer overflow.

Tuttavia queste misure di sicurezza non riescono a fornire protezione totale; vedremo in seguito come le Trusted Platform possono affiancare le attuali misure di sicurezza per rafforzarne l'efficacia, introducendo la proprietà, finora mancante, di *fiducia*. Fiducia che la piattaforma di calcolo si comporti nel modo per il quale è stata progettata.

1.2 Soluzioni attuali e loro limitazioni

Le infrastrutture di sicurezza esistenti che cercano di contrastare le minacce alla sicurezza si possono suddividere in grandi categorie[5] ognuna delle quali presenta però delle limitazioni; in genere l'impiego di più soluzioni possono mitigare il rischio che eventuali attacchi possano avere esito positivo, per l'attaccante, e negativo, per l'attaccato.

Firewall

Forniscono protezioni alle reti di computer delimitandone i confini. Indipendentemente dalla tipologia di firewall adottato, *packet filter* o *application gateway*, il loro impiego può sicuramente chiudere molti punti di accesso verso la rete protetta, limitando la possibilità di successo di un attacco.

Tuttavia essi rappresentano spesso "colli di bottiglia" nei confronti dell'utilizzo della banda disponibile e "single point of failure" nella maggior parte dei casi d'impiego, senza contare, altresì, tutti i casi di errata configurazione. Inoltre, la crescente complessità delle applicazioni di rete esistenti rende la configurazione e la progettazione di sistemi firewall complessa e non alla portata di tutti, sopratutto delle aziende che non possono permettersi un IT security a "tempo pieno".

[5] rimandi alla letteratura esistente per una maggior trattazione dell'argomento; una buona introduzione può essere fornita da [20] e [28].

Security software

Un numero sempre più crescente di programmi che forniscono funzionalità di sicurezza è disponibile al giorno d'oggi. Questi programmi possono essere eseguiti all'interno di un co-processore crittografico o sulla CPU principale della piattaforma di calcolo fornendo, ad esempio, funzionalità di cifratura a sistemi firewall.

Software che vengono eseguiti sul processore principale assumono, però, di venire eseguiti in un ambiente fidato, così che la massima sicurezza è fornita solo se il programma è installato e viene eseguito correttamente.

Tuttavia il software, generico, è vulnerabile a diversi tipi di attacchi, come quelli proveniente da virus o buffer overflow, ad esempio e il Security Software non rappresenta certo un'eccezione.

Co-processore crittografico

Rappresenta una combinazione di hardware/firmware che fornisce funzioni di sicurezza spesso più veloci di quelle che possono essere fornite dal processore "general-purpose" della piattaforma.

Forniscono inoltre un ambiente protetto per i dati sensibili ed includono meccanismi che possono rilevare tentativi effettuati per guadagnare accesso a questi dati.

Uno dei difetti maggiori legato a questa tecnologia, che preclude il fatto di un'installazione su larga scala, è l'elevato costo. Hardware di questo tipo può costare anche centinaia di USD ed incide decisamente sul costo totale di un PC.

Altre tecnologie

Altre tecnologie possono essere utilizzate per aumentare il livello di sicurezza[6] delle piattaforme di calcolo; modifiche di sicurezza ai kernel dei sistemi operativi più famosi

[6] PaX [4] e NGSEC StackDefender [5] ad esempio, per contrastare vulnerabilità di tipo *buffer overflow*.

hanno reso più arduo lo sfruttare particolari vulnearbilità. Progettazione ed implementazione di protocolli di sicurezza come il Transport Layer Security (TLS) o Internet Protocol Security Protocol (IPSec), insieme ad altri come Secure Multi-Purpose Internet Mail Extensions (S-MIME), Internet Key Encryption (IKE) e reti private virtuali (Virtual Private Network, VPN) forniscono diverse funzionalità di sicurezza, come autenticazione, controllo degli accessi e confidenza nelle informazioni attraverso l'uso della crittografia. Tutte queste tecniche che coinvolgono l'esecuzione di software sulla CPU della piattaforma di calcolo necessitano però di operare in un ambiente *Trusted*, fidato, dove la piattaforma di calcolo opera sempre nel modo per cui è stata progettata e che ci si aspetta, ambiente che finora è stato assunto come ipotesi, ma che non è stato mai, o difficilmente, garantito.

Capitolo 2

Trusted Computing Platform

Le problematiche accennate nel capitolo precedente, insieme alle minacce alla sicurezza e all'impiego delle tecnologie esistenti, accoppiate con le opportunità emergenti dell'e-business che richiedono livelli maggiori di confidenza e fiducia, hanno guidato il Trusted Computing Group[1] *[13],* TCG*, a progettare una specifica aperta per piattaforme di calcolo che crea una base di* trust[2] *per i processi software.*

Il capitolo fornisce un'introduzione alle Trusted Computing Platform definite dalla specifica TCG [10], [11] e [12]. Vengono introdotte le root of trust*, componenti hardware necessari per garantire la proprietà di fiducia nella piattaforma, proprietà che si traduce, come spesso accade nel mondo informatico, in un concetto di* fiducia sociale *che vede coinvolte molte entità al fine di garantire il corretto funzionamento della piattaforma di calcolo. Infine, dopo aver introdotto brevemente le necessità che hanno spinto i ricercatori a delineare le architetture trusted e i benefici che esse portano, viene illustrata una panoramica architetturale delle Trusted Platform, definita dalla specifica TCG.*

[1] conosciuto prima come *Trusted Computing Platform Alliance, TCPA*

[2] il termine inglese presenta molti significati quali fiducia, confidenza, aspettazione e responsabilità; si usa questo termine nel resto del lavoro per evidenziare semplicemente i vari significati.

2.1 Trusted Platform e TCG

Una Trusted (Computing) Platform[3], TP, è una piattaforma di calcolo che presenta uno o più componenti *fidati*, visti sottoforma di hardware incorporato nella piattaforma stessa. Il concetto di fiducia è riferito alla capacità, per una TP, di garantire per la correttezza e l'inalterabilità dei dati prodotti da essa. I componenti introdotti vengono usati per creare una base di fiducia per i processi software della TP.

Per convertire una piattaforma di calcolo in una TP, è necessario introdurre delle *root of trust*, definite dalla specifica TCG. Queste root of trust rappresentano l'hardware che implementa funzioni di sicurezza che *devono* essere intrinsicamente fidate affinché tutta la piattaforma possa essere considerata tale.

Determinare il grado di fiducia che si può avere nella piattaforma, è una caratteristica critica di una TP. I meccanismi di sicurezza sono usati per fornire informazioni che sono necessarie per dedurre il livello di fiducia nella piattaforma. La fiducia risiede nella dimostrazione che determinati processi, protocolli o metodologie, funzionano nel modo per il quale sono stati progettati; decidere se le informazioni prodotte da una TP, attraverso processi fidati, sono significativi e sufficienti a decretare la fiducia dell'intera TP, è lasciata agli utenti che stanno interagendo con la TP.

In altre parole, gli utilizzatori di una TP possono *fidarsi* dell'operato della TP e del modo in cui essa produce o protegge le informazioni, ma la semantica associata alle informazioni prodotte, può essere giudicata soltanto da questi utilizzatori.

2.2 Root of Trust

Le TP soddisfano la necessità dell'aumento di fiducia nelle piattaforme. Questa fiducia deriva da dichiarazioni effettuate da terze parti fidate[4] che garantiscono che la piattaforma può essere fidata per gli scopi prefissati. Le TTP sono disposte a garantire per una TP perché l'hanno esaminata, valutata e sono pronti a dichiarare che i meccani-

[3]Esistono diversi tipi di TP ma in questo lavoro si focalizza sulle TP secondo specifica TCG

[4]Trusted Third Part, TTP

smi messi a disposizione dalla TP, nonchè la proprietà di fiducia sulla quale si basa la piattaforma stessa, sono genuini.

Gli utenti locali e remoti si fidano[5] del giudizio delle TTP, così se la piattaforma dimostra la sua *identità* e la proprietà di fiducia può essere univocamente verificata, gli utenti hanno fiducia nella piattaforma e sanno che si comporterà in un modo fidato e "predicibile". Un primo passo per garantire la correttezza del software presente nella TP consiste nel garantirne l'integrità; a tal fine una Trusted Platform presenta due root of trust:

- *root of trust* che inizia il processo di misurazione dell'ambiente hardware e software della piattaforma. Questo processo permette di determinare in modo sicuro e inopinabile, ad esempio, che il software in esecuzione su una TP sia inalterato, genuino e si comporti nel modo per il quale è stato progettato. In altre parole, il meccanismo determina l'integrità dell'ambiente software di una TP. Questo componente è indicato come *Root of Trust for Measurement, RTM*

- *root of trust* che memorizza i risultati del processo svolto dall'RTM in un modo irreversibile. Riporta crittograficamente i valori delle misurazioni attuali e impedisce il rilascio di un segreto se i valori attualmente misurati non combaciano con i valori memorizzati con quel segreto. Questo componente è il *Root of Trust for Storing, RTS* e *Root of Trust for Reporting, RTR.*

RTM, RTS e RTR sono i due[6] root of trust necessari che devono essere assiomaticamente fidati affinchè ci sia prova che i meccanismi della TP possano operare nel modo per il quale sono stati progettati.

I due root of trust cooperano per permettere un processo attraverso il quale è possibile ottenere metriche d'integrità. Le metriche d'integrità rappresentano misurazioni effettuate sull'ambiente hardware e software della piattaforma e vengono usate per

[5] concetto di *fiducia sociale*; è lo stesso tipo di fiducia che si adotta nei confronti di una *Certification Authority* in un contesto PKI.

[6] RTS ed RTR sono parte di un unico componente hardware chiamato *Trusted Platform Module, TPM.*

dimostrare che la piattaforma è nello stato voluto, condizione necessaria per poter processare dati sensibili.

Le metriche d'integrità possono essere utilizzate per dimostrare ad un utente locale, o ad una terza parte, che la piattaforma sta operando come voluto per impedire il rilascio di segreti, a meno che la piattaforma stia eseguendo particolari software e non si trovi quindi in particolari stati reputati fidati dagli utenti.

La misurazione delle metriche d'integrità rappresenta un concetto fondamentale in tutta la specifica TCG. Come vedremo nella sezione 2.5 e più dettagliatamente nel capitolo 3, le metriche d'integrità rappresentano la base sulla quale TCG ha costruito e messo a disposizione altri meccanismi peculiari delle TP, quali boot sicuro (secure boot), boot autenticato (authenticated boot), identità della piattaforma (platform identity) e memorizzazione protetta di dati (protected storage).

Per potersi fidare delle misurazioni effettuate, è necessario potersi fidare delle root of trust e in generale di tutta la Trusted Platform.

2.2.1 Attestazione di una Trusted Platform

Sono molte le entità e organizzazioni coinvolte nel processo di attestazione di una Trusted Platform e delle sue componenti al fine di garantirne la proprietà di fiducia. Il seguente elenco e la figura 2.1 illustrano queste entità:

Trusted Platform Module Entity. Il TPME garantisce che il TPM sia *genuino*. Viene inserita, nel TPM, una chiave crittografica asimmetrica, l'*Endorsement Key* e il TPME firma digitalmente una credenziale, l'*Endorsement Credential*, che include le seguenti informazioni:

- Nome del costruttore del TPM
- Numero di serie del TPM
- Versione del TPM
- Parte pubblica dell'Endorsement Key

Conformance Entity. La CE garantisce che la progettazione del Trusted Platform Subsystem (vedere paragrafo successivo) sia conforme alla specifica TCG. A

questo proposito la CE rilascia una credenziale, la *Conformance Credential*, che include le seguenti informazioni:

- Nome dell'entità che ha rilasciato la credenziale
- Nome del costruttore della piattaforma
- Versione della piattaforma (se disponibile)
- Nome del costruttore del TPM
- Numero di serie del TPM
- Versione del TPM

Platform Entity. La PE identifica il costruttore della piattaforma e ne descrive le proprietà; l'entità garantisce che la piattaforma incorpori un TPM genuino, come da specifica TCG. A questo proposito la PE rilascia una credenziale, la *Platform Credential*, che contiene riferimenti all'Endorsement Credential e alla Conformance Credential e inoltre include le seguenti informazioni:

- Nome del costruttore della piattaforma
- Numero del modello della piattaforma
- Versione della piattaforma (se disponibile)
- Riferimento all'Endorsement Credential
- Riferimento alla Conformance Credential

La Platform Credential identifica in modo *univoco* la piattaforma e per questa ragione questo certificato potrebbe essere considerato un dato privato.

Privacy Certification Authority, P-CA questa CA è scelta dal proprietario del TPM ed esiste per semplificare il processo di scambi di certificati nei processi che vedono coinvolta la TP, nel momento di generazione di un'identità (vedere sezione 3.1) e per risolvere un problema di privacy[7].

[7] nella versione 1.2 sono stati introdotti altri meccanismi che, utilizzando *blind signature*, risolvono il problema alla fonte.

Il quadro delle relazioni che devono intercorrere tra queste entità e delucidazioni sul significato dell'Endorsement key, verrà completato nella sezione 3.1.

Concetti essenziali nel modello TP

TCG ha pubblicato dei documenti ([10], [11] e [12]) che specificano come una TP deve essere costruita.
In ogni TP c'è un Trusted Platform Subsystem che contiene:

- Trusted Platform Module, TPM
- Core Root of Trust for Measurement, CRTM
- Software di supporto, Trusted (Platform) Support Service, TSS

Il TPM è un chip hardware separato dalla CPU principale. Il CRTM è il primo "software"[8] ad essere eseguito durante il processo di boot della piattaforma. Il TSS esegue varie funzionalità, come quelle necessarie per comunicare con il resto della piattaforma e con altre piattaforme.

Le funzionalità del TSS non devono essere fidate assiomaticamente; è sufficiente che sia possibile dimostrare l'integrità del TSS affinché la piattaforma possa essere considerata fidata.

Oltre al TSS, per i discorsi accennati relativi alla fiducia sociale, è necessario coinvolgere Certfication Authority nella costruzione e nell'utilizzo della TP per poterne attestare la genuinità.

2.3 Le necessità

La proprietà di fiducia è fondamentale in tutti i processi che coinvolgono scambio di dati sensibili. Il meccanismo di fiducia in una TP genera in modo attendibile, memorizza e riporta le misurazioni riguardo l'ambiente software della piattaforma, permettendo

[8] l'RTM è il componente hardware; il CRTM rappresenta il codice che deve essere eseguito da tale hardware. Ne rappresenta quindi il *firmware*.

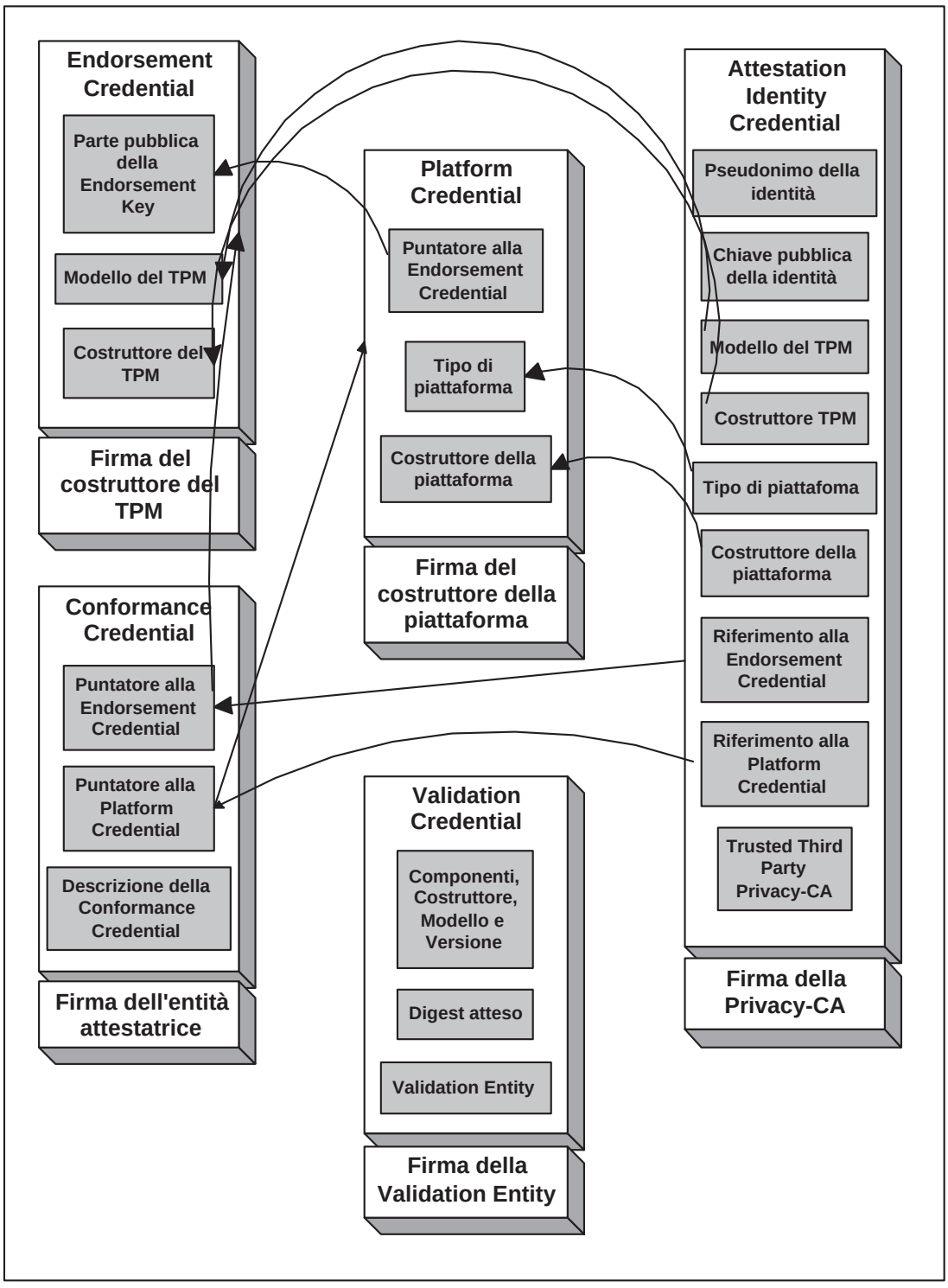

Figura 2.1: Credenziali e relazioni tra di esse

di conoscerne, in qualsiasi momento, lo stato. Una volta determinato lo stato, grazie anche alle garanzie di genuinità delineate nel paragrafo precedente, fornite dalle TTP, è possibile decidere se tale stato può essere considerato fidato per le necessità e scopi del momento.

2.3.1 Piattaforme fidate e piattaforme sicure

Le piattaforme fidate introducono quindi una componente che non era mai stata considerata fin'ora nel processo nato per rendere sicure le tecnologie informatiche esistenti; vediamo come tale fiducia possa migliorare o coadiuvare le esistenti infrastrutture di sicurezza dei sistemi informatici e le principali caratteristiche, che intercorrono tra piattaforme sicure e piattaforme fidate.

- le piattaforme sicure non possono provare che stanno operando nel modo atteso. Convenzionalmente i computer sicuri sono più costosi e complessi da utilizzare
- una piattaforma di calcolo ha integrità se le applicazioni che vengono eseguite su di essa funzionano senza interferenza di nessun tipo. Quindi se una piattaforma deve fornire in modo affidabile la sua integrità, deve esistere un meccanismo che fornisce queste misurazioni d'integrità, in modo *fidato*.
- le piattaforme fidate forniscono meccanismi per la memorizzazione protetta dei dati (chiavi di cifratura, dati di autorizzazione, ...). Inoltre forniscono meccanismi per associare segreti cifrati ad una piattaforma fisica, assicurando che tali dati siano accessibili solo su quella stessa piattaforma. Quando questi dati vengono cifrati, possono anche essere vincolati, alla piattaforma, e venire decifrati solo quando la stessa si trova con un determinato ambiente software, il che è testimonianza di un determinato stato fidato.

2.4 I benefici introdotti

I benefici portati dall'impiego delle Trusted Platform sono molteplici. Alcuni sono di carattere generale, altri invece si possono classificare in base al tempo necessario entro

il quale le TP saranno impiegate nei sistemi informatici.

Un vantaggio portato dall'impiego delle TP è che le funzioni svolte dall'hardware sicuro operano su piccole quantità di dati, permettendo quindi buone performance pur mantenendo il costo dell'hardware stesso decisamente contenuto. Inoltre, le root of trust hardware forniscono protezioni per i processi software della TP. Un co-processore crittografico, di contro, è un dispositivo hardware sicuramente più costoso che *non* offre garanzie ai processi software della piattaforma, ma soltanto ai suoi processi di sicurezza interni.

Breve termine

Gli utenti di una TP possono utilizzare il meccanismo di Protected Storage (sezione 2.5 e 3.3) messo a disposizione dalla specifica TCG per proteggere la riservatezza dei dati sui loro dispositivi di massa, in un modo fondamentalmente più sicuro rispetto all'utilizzo di soluzioni software.

Tramite il Protected Storage, il TPM:

- agisce come portale per cifrare i dati
- fornisce un'opzione, che non deve necessariamente essere usata, che permette ai dati cifrati di essere decifrati soltanto sulla stessa piattaforma
- fornisce e protegge chiavi usate in processi di firma digitale

Medio termine

In questo periodo, l'utilizzo delle TP offriranno, oltre ai benefici apportati nel breve periodo, la possibilità di misurare le integrità relative all'ambiente software della piattaforma, per uso interno della stessa.

Il TPM agirà da portale per cifrare i dati in modo tale che gli stessi possano essere decifrati solo se la piattaforma ha l'insieme giusto di metriche d'integrità, ovvero solo se la piattaforma si trova in un particolare stato fidato.

Lungo termine

La specifica TCG potrà essere sfruttata al meglio garantendo, oltre che ai benefici delineati nel medio termine, anche quelli dovuti alla comunicazione delle metriche d'integrità relative all'ambiente software misurato a terze parti. Questo *modus operandi* richiede sicuramente l'utilizzo di Public Key Infrastructure.

Privacy

Nel contesto informatico, la privacy fornisce un modo per impedire a terzi di ottenere accesso alle informazioni senza che il legittimo proprietario ne dia il consenso.

Il controllo sulla privacy dovrebbe determinare se è permesso rivelare l'esistenza delle informazioni e sotto quali circonstanze esse possono essere divulgate o utilizzate.

Alcuni dati associati con le TP non richiedono protezioni di sicurezza, ma potrebbero essere considerati sensibili da un punto di vista della privacy. Per poter mantenere privati tali dati, la specifica TCG richiede che gli accessi a questi dati siano sotto il controllo del proprietario del dato stesso. Inoltre, come notato nei benefici a medio termine, le TP sono in grado di fornire un'altra forma di protezione della privacy, impedendo la rivelazione dei segreti a meno che lo stato del software di una piattaforma non sia quello approvato.

La specifica TCG rispetta anche la privacy di un utente di una TP. La specifica infatti differenzia tra utenti di una TP e owner di una TP; l'owner, chiaramente, possiede determinati privilegi, ma non è un *super user*; non può accedere, perlomeno in linea di principio, ai dati privati di altri utenti, senza il consenso di questi.

2.5 Una panoramica architetturale

Alla luce di quanto detto nei paragrafi precedenti, una TP non è nient'altro che una piattaforma di calcolo "normale", che è stata modificata per poter garantire una proprietà fondamentale, oggi giorno sempre più ricercata in una piattaforma; la fiducia.

I sistemi per la protezione delle informazioni richiedono il controllo della priva-

cy per permettere ai proprietari delle piattaforme di calcolo di averne il controllo, sopratutto riguardo l'identità e l'attivazione.

L'introduzione nelle piattaforme di calcolo di questa proprietà, insieme alle funzionalità dettate dalla specifica TCG, garantisce le seguenti caratteristiche di base proprie di una TP:

1. Protezione contro i furti e uso errato dei segreti mantenuti sulla piattaforma (concetto di protected storage: vedere il sommario funzionale e la sezione 3.3 per maggiori dettagli a riguardo) essendo i dati protetti dall'hardware root of trust.

2. Meccanismo per permettere ad una piattaforma di calcolo di dimostrare che è una Trusted Platform e nello stesso tempo provare la sua identità (concetto di platform identity: vedere il sommario funzionale e la sezione 3.1 per maggiori dettagli a riguardo).

3. Meccanismo per mostrare che la piattaforma sta eseguendo il software voluto; dimostrazione che la piattaforma è in uno stato approvato, di fiducia (concetto di integrity reporting: vedere sezione 3.2).

La conversione di una piattaforma di calcolo ordinaria in una TP richiede, come già accennato nella sezione 2.2, la necessità di fornire *due* importanti meccanismi:

1. Un modo fidato per raccogliere informazioni su un sistema e memorizzarle. Queste informazioni vengono chiamate metriche d'integrità e il processo attraverso il quale tali misurazioni possono essere effettuate verrà delineato nella sezione 3.2.

2. Un modo fidato per recuperare tali metriche, così da poterle confrontare e prendere le decisioni del caso.

Questi meccanismi sono forniti dalle due root of trust introdotte nella sezione 2.2, il Root of Trust for Measurement, RTM, e il Trusted Platform Module, TPM.

L'RTM si preoccupa di misurare le metriche d'integrità di una Trusted Platform. A tal fine deve:

- eseguire *solo* i programmi permessi dall'entità che lo attesta,
- resistere a forme di attacco software e hardware nei termini previsti dal *Protection Profile*[9] della piattaforma,
- misurare almeno una metrica d'integrità al fine di indicare l'ambiente software di una TP e permettere di determinarne quindi il livello di fiducia,
- registrare un sommario delle metriche misurate in un Trusted Platform Module,
- registrare il dettaglio del processo di misurazione di tutte le metriche d'integrità in un Trusted Platform Measurement Store (TPMS)

Il TPM rappresenta il Root of Trust for Storing e il Root of Trust for Reporting e deve:

- resistere a forme di attacco software e hardware nei termini descritti dal Protection Profile della piattaforma,
- accettare le metriche d'integrità misurate e registrarle,
- fornire un sommario accurato di tutte le sequenze delle metriche misurate e memorizzate

In pratica l'RTM effettua le misure[10] delle metriche d'integrità mentre il TPM è responsabile della loro memorizzazione e comunicazione.

Questo significa che l'RTM è il primo programma eseguito dalla piattaforma, o i programmi che vengono eseguiti prima dell'RTM devono essere assiomaticamente fidati.

Implementare un RTM vuol dire fornire le istruzioni macchina che permettano all'RTM di effettuare il suo compito. Questo insieme di istruzioni prende il nome di

[9] i *Protection Profile* sono metodi usati per descrivere le proprietà di sicurezza della piattaforma. I metodi sono descritti nei *Common Criteria* [6] e [1].

[10] tecnicamente è sempre il TPM che effettua la misurazione, ma il processo è coordinato dall'RTM

Core Root of Trust for Measurement (CRTM). Allo stato attuale, riferendosi all'architettura dei PC, il CRTM può essere implementato come BIOS Boot Block (nel caso di BIOS composti) o come BIOS.

In accordo con la specifica TCG, una Trusted Platform deve avere:

- almeno una root of trust per misurare le metriche d'integrità: è rappresentata dall'RTM,
- esattamente una root of trust per memorizzare e riportare le metriche d'integrità: è rappresentata da RTR e RTS → TPM,
- almeno un TPMS,
- esattamente un Trusted Platform Agent (vedere alla fine del capitolo; concettualmente svolge una funzionalità simile all'RTM, ma non rappresenta una root of trust)

Sommario architetturale

Come accennato nelle sezioni precedenti, i meccanismi di sicurezza possono essere incrementalmente adattati alle piattaforme esistenti ma l'introduzione delle Trusted Platform richiedono un cambiamento fondamentale all'architettura della piattaforma stessa.

Descriviamo brevemente i componenti architetturali e i principali meccanismi che le Trusted Platform introducono. Una trattazione completa delle funzionalità introdotte dalle TP, relativa a platform identity, integrity measurement, recording and reporting e protected storage, viene rimandata al capitolo 3.

TPM

Il TPM, cuore dell'architettura TCG, è un chip hardware saldato sulla scheda madre[11] e rapprenta un motore di calcolo separato dalla CPU principale. Rappresenta una delle

[11] attualmente è situato sul bus Low Pin Count [3], bus che rimpiazza l'obsoleto ISA.

due root of trust aggiunte, previste dalla specifica ed è illustrato nella figura 2.2.

Proprietà crittografiche

Il TPM implementa funzionalità tipiche di un sistema crittografico[12]:

- Generazione di numeri pseudo-casuali (PRNG),
- Hash e Message Authentication Code,
- Generazione di chiavi asimmetriche,
- Cifratura/decifratura asimmetrica

Protected Capability e Shielded Location

Il TPM fornisce funzionalità, *protected capability*, e locazioni di memoria schermate, *shielded location*, che sono protette da attacchi di interferenza. Grazie a queste funzionalità, il TPM è in grado di garantire fiducia anche a entità remote e non solo al proprietario del TPM stesso.

Il dispositivo contiene tutte le funzioni che *devono* essere fidate affinchè possa essere considerata fidata l'intera piattaforma. Tutte le altre funzionalità previste dalla specifica TCG che non devono essere protette dal TPM, come ad esempio il software facente parte del Trusted Support Service (TSS), devono comunque operare nel modo corretto. Il corretto operato delle funzionalità del TSS può tuttavia, essere controllato tramite il meccanismo di misurazione di integrità di cui si rimanda alla sezione 3.2. Questo controllo d'integrità, ovviamente, non può essere effettuato sulle funzionalità intrinseche del TPM, ma questo non rappresenta un problema: altre entità si sono preoccupate di attestare per la genuinità del TPM, della sua progettazione e integrazione su una TP.

Una protected capability è quella il cui corretto operato è *necessario* affinchè la piattaforma sia considerata trusted. Le protected capability sono le uniche funzionalità che hanno accesso alle shielded location.

[12] far riferimento alla sezione 6.1 per una descrizione più dettagliata.

Una shielded location è un'area nella quale i dati vengono protetti da attacchi di interferenza e snooping. Soltanto una piccola quantità di dati sensibili vengono memorizzati in queste zone di memoria incorporate nel TPM perché uno degli obiettivi della specifica è il contenimento dei costi. Questo non rapprenta un problema, perché, come avremmo modo di descrivere più avanti (sezione 3.3), una quantità virtualmente illimitata di dati può essere memorizzata al di fuori del TPM, sottoforma di "blob" cifrato dal TPM stesso, in modo tale che solo il TPM possa decifrarlo per poterlo gestire.

Platform Configuration Register

I PCR rappresentano delle shielded location e sono contenuti all'interno del TPM. Questi registri servono per memorizzare le misurazioni, coordinate dal CRTM o da altri agenti di misurazione fidati, delle metriche d'integrità. La memorizzazione di tali metriche è effettuata in un modo tale da poter misurare un numero di oggetti potenzialmente infinito e garantire, così, l'integrità della piattaforma.

La descrizione delle misurazioni delle metriche d'integrità viene affrontata dettagliatamente nella sezione 3.2.

Data Integrity Register

I DIR rappresentano anch'essi delle shielded location e sono contenuti all'interno del TPM. Questi registri contengono anch'essi un sommario della misurazione delle metriche d'integrità, ma sono usati in un modo tale da garantire che la piattaforma esegua un processo di boot sicuro. Tale processo verrà descritto nella sezione 3.2.2.

Meccanismi di autorizzazione

Generalmente i termini *autorizzazione* e *autenticazione* vengono usati come sinonimi, ma non lo sono. Con autenticazione, generalmente, si mira a dimostrare l'*identità* di un qualcosa/qualcuno (l'autenticità di un programma, entità, persona, ...) mentre con il termine autorizzazione si mira ad autorizzare una certa entità a compiere una determinata azione; entità che può essere stata autenticata o no.

Un tipico esempio di autenticazione-autorizzazione è dato dal processo di login-password; tipico esempio di challenge/response relativo all'autenticazione. Una volta che l'utente ha provato la sua identità mediante qualcosa che sa (login e password), è, in questo caso, autorizzato implicitamente a fare tutto *tranne* compiti particolari[13] per i quali bisogna avere autorizzazioni diverse.

Il TPM, come accennato sopra, implementa comandi che possono accedere a shielded location oppure comandi che non richiedono accesso a particolari zone di memoria protette. Per questo motivo molte capability del TPM richiedono prova di autorizzazione[14] mediante l'utilizzo di un protocollo di challenge/response. Il protocollo utilizzato prevede che non venga mai richiesta la presentazione dell'informazione di autorizzazione in modo *esplicito* (a differenza del classico scambio login/password su un canale insicuro). Questo permette, ad esempio, ad un'entità remota di poter utilizzare capability che richiedono autorizzazione, senza esporre il segreto stesso in chiaro, via rete, su un canale insicuro (lo stesso meccanismo viene comunque utilizzato anche in locale, dal TPM, all'interno della TP).

I dati di autorizzazione

Lo scopo del meccanismo di autorizzazione è di autenticare il proprietario di un TPM[15] o di autorizzare, come accennato nel paragrafo precedente, l'utilizzo di capability particolari del TPM.

Ogni oggetto protetto dal TPM, chiave o dato arbitrario che sia, nella gerarchia del protected storage, che verrà spiegato nella sezione 3.3, ha un oggetto *parent* e uno o più oggetti *figli*.

Esistono tre tipi di dati di autorizzazione gestiti dal TPM:

Owner Authorization Data: è unico. Questo dato di autorizzazione è usato dal TPM per controllare l'accesso a funzionalità che richiedono autorizzazione del pro-

[13] si pensi ad un utente non privilegiato e all'utente *root* nei sistemi Unix o *Administator* nei sistemi MS Windows.

[14] l'autorizzazione è legata all'oggetto memorizzato nelle shielded location.

[15] concetto di *Take Ownership*, per il quale si rimanda alla sottosezione 3.1.1.

prietario del TPM. Questo è l'unico oggetto al quale il TPM associa un ruolo preciso. Fornire la prova della conoscenza di questo dato di autorizzazione, significa *possedere* il TPM.

Authorization Data: è generico. Questo dato di autorizzazione non è associato con nessun ruolo o utente. È il tipo di autorizzazione più comune; è creato insieme ad ogni oggetto, chiave o dato arbitrario, che deve essere protetto dal TPM. La conoscenza del segreto legato all'oggetto dimostra di avere l'autorizzazione necessaria per utilizzarlo.

Migration Authorization Data. È possibile legare un oggetto protetto dal TPM ad una TP particolare, impedendo quindi la sua migrazione verso altre TP. Questo è reso possibile dalle separazione del segreto di autorizzazione da quello necessario per la migrazione; la divisione permette, altresì, a determinati utenti di usare l'oggetto senza aver la possibilità di controllarne la migrabilità verso altri TPM o viceversa.

I dati di autorizzazione per il proprietario del TPM sono mantenuti in memoria Non-Volatile protetta dal TPM, le shielded location. I dati di autorizzazione degli altri oggetti protetti dal TPM fanno parte, invece, della struttura dati che rappresenta gli oggetti stessi, che viene cifrata nel momento in cui l'oggetto viene memorizzato al di fuori del TPM.

Grazie alla definizione di questi tre tipi di dati di autorizzazione, il proprietario del TPM non rappresenta un *super user*: l'utilizzo di qualsiasi oggetto protetto dal TPM, richiede la conoscenza del suo dato di autorizzazione.

I protocolli di autorizzazione, che forniscono i meccanismi di autorizzazione con i quali è possibile fornire la prova della conoscenza di un segreto necessario per utilizzare gli oggetti protetti dal TPM, previsti dalla specifica TCG verranno descritti dettagliatamente nel capitolo 7, dove verrà presentata anche un'analisi di sicurezza su questi protocolli, fondamentali per l'intera architettura.

Agenti di misurazione

La specifica TCG definisce anche le funzioni tipiche che devono essere implementate da un TSS. Queste funzioni non devono essere root of trust, ma, affinché la piattaforma operi nel modo che ci si aspetta, devono essere fidate in ogni caso. Le "nuove" funzioni trusted possono essere usate per garantire la fiducia di altre funzionalità e così via fino a poter garantire la fiducia del software ordinario della TP. In questo modo si viene a creare una *catena di fiducie* transitive a partire dai root of trust presenti sulla TP, l'RTM e il TPM, fino alle applicazioni eseguite da un sistema operativo. L'errato comportamento di qualsiasi funzione/software presente all'esterno del TPM, può quindi essere rilevato attraverso controlli d'integrità. Soltanto gli utenti finali possono fidarsi di un determinato stato nel quale si trova la TP. È la necessità dell'utente e lo scopo per il quale viene utilizzata la TP che determina il livello di fiducia che l'utilizzatore ha nella piattaforma.

Gli agenti di misurazione sono funzionalmente simili all'RTM, cooperano per poter coordinare il processo di misurazione d'integrità della TP, ma non sono root of trust. Fanno parte pertanto della catena di fiducia transitiva che si viene a creare a partire dalle root of trust, come illustrato nell figura 3.3

Trusted Platform Measurement Store

Quando l'RTM o gli altri agenti di misurazione presenti nel sistema effettuano misurazioni, è necessario memorizzare le informazioni dettagliate dei componenti misurati in un *repository* presente nella TP. Il TPMS rappresenta questo deposito.

Trusted Platform Agent

Il TPA è il software che coordina il processo di fornitura delle metriche misurate ad un utente che ne faccia richiesta. Tale processo, insieme alla descrizione di un framework progettato a tal fine, viene descritto nella sezione 3.2.

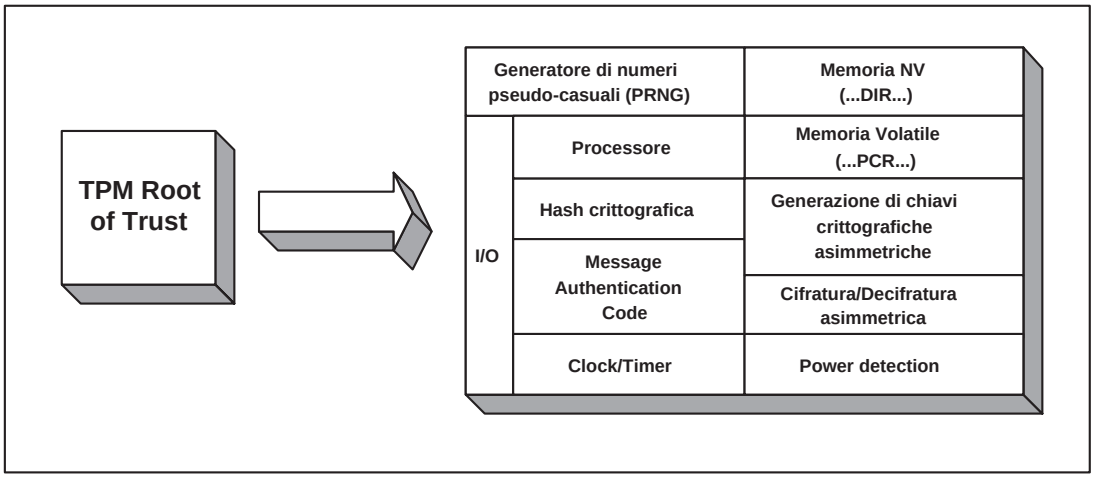

Figura 2.2: Architettura logica del TPM

Capitolo 3

Funzionalità delle TCG Trusted Platform

La specifica TCG oltre a fornire i dettagli necessari per poter trasformare una piattaforma di calcolo in una Trusted Platform, fornisce anche una serie di meccanismi che possono essere impiegati per aumentare la sicurezza, e non solo la fiducia (concetto introdotto nel capitolo precedente), di una piattaforma di calcolo.

Nel capitolo esamineremo una rassegna delle principali funzionalità offerte dalle TP, quali il concetto e la creazione di identità associate alla TP al fine di garantire per i dati prodotti da essa (sezione 3.1), il concetto di misurazione, memorizzazione e comunicazione delle metriche d'integrità il cui scopo è garantire lo stato fidato[1] *o meno della piattaforma (sezione 3.2) e il concetto di protezione di dati e chiavi, simmetriche o asimmetriche, create dal TPM o al di fuori di esso; protezione che può sfruttare sofisticati meccanismi di sicurezza e liberare i segreti soltanto in determinati stati considerati fidati.*

I meccanismi di autorizzazione e i modi operazionali del TPM previsti dalla specifica, seppur di grande importanza, non rappresentano esattamente funzionalità offerte e pertanto verrano analizzati in altra sede, rispettivamente nel capitolo 7 e nella sezione 6.2.

[1] la fiducia o meno nello stato di una piattaforma non può essere decisa a priori; sono gli utenti che, utilizzando i dati prodotti dal TPM, determinano se lo stato in cui si trova la TP è fidato o meno; TCG offre meccanismi per garantire sull'autenticità dei dati forniti.

3.1 Identità della piattaforma

L'identificazione in modo univoco di una Trusted Platform permette di fornire fiducia e confidenza nei dati prodotti da quella TP particolare, rispetto a chiunque (utenti o entità) interagisca con essa. Il meccanismo di determinazione dell'identità di una Trusted Platform è quindi un meccanismo fondamentale per garantire la proprietà di trust fornita da questa tipologia di piattaforme e viene descritta in particolare in [13], [10], [11] e [12]. L'identità di una TP garantisce in modo inopinabile che i dati prodotti sono inalterati e genuini, ossia prodotti su una TP dotata di un TPM particolare[2].

Il primo scopo del meccanismo per l'identificazione della piattaforma rappresenta perciò un modo di fornire ad un destinatario di dati prodotti dalla TP, un certo di grado di confidenza e fiducia nei dati ricevuti.

Il processo di fiducia che sta alla base di questo meccanismo è un processo che vede coinvolte diverse entità che garantiscono ognuna per un particolare aspetto che compone la Trusted Platform; queste entità, menzionate nella sezione 2.2.1, cooperano non solo per garantire la genuinità di una TP, ma anche per poter permettere la creazione di identità associate con una TP particolare. Le identità cosi create e associate ad una TP particolare, donano fiducia ai dati prodotti da quella TP particolare. L'utilità dei dati prodotti dipende dal contesto di utilizzo dei dati e dalle entità che ne fanno uso; la specifica TCG, però, con questo meccanismo, è in grado di *garantire* per l'autenticità dei dati prodotti da una TP. TCG, infatti, distingue tra:

1. provare che la piattaforma è una Trusted Platform *genuina*. A questo proposito le varie entità descritte nella sezione 2.2.1 producono un certificato digitale che attesta diverse proprietà della TP, come la genuinità del TPM, incorporazione del TPM su una TP secondo specifica TCG e così via (concetto di *attestazione* di una TP).

2. provare che la piattaforma è una *certa*, particolare TP, diversa da un'altra (con-

[2] il meccanismo di creazione di Platform Identity è usato anche nel processo di restituzione delle metriche d'integrità della piattaforma come spiegato nella sezione 3.2.

cetto di *identità di una piattaforma*, spiegato successivamente e in particolare in 3.1.2 e in 3.1.3).

Anche se le due distinzioni possono sembrare apparentemente simili, la specifica sottolinea le differenze relative agli scopi per i quali vengono utilizzati i due meccanismi.

3.1.1 Endorsement Key

Il meccanismo delineato nel punto 1 consiste nell'inserimento di una chiave crittografica, l'*Endorsement Key*, nel TPM, da parte del TPME (sezione 2.2.1). Questa chiave rappresenta in realtà una coppia di chiavi crittografiche asimmetriche localizzate nella memoria Non Volatile del TPM.

- il TPM ha una sola coppia di Endorsement Key.
- la parte pubblica l'Endorsement Key viene usata per cifrare dati all'esterno del TPM in modo tale che la decifratura sia effettuata dal TPM cui corrisponde la corretta chiave privata.
- la parte privata dell'Endorsement Key non viene mai usata per cifrare o firmare.

Take Ownership

L'utilizzo dell'Endorsement Key permette la comunicazione al TPM di alcuni dati in modo tale da garantirne la confidenza. Tali dati vengono cifrati con la parte pubblica dell'Endorsement Key e inviati al TPM. Soltanto il TPM a cui corrisponde la chiave privata giusta sarà in grado di decifrare tali dati e compiere le dovute operazioni. In particolare, l'Endorsement Key viene usata dal proprietario del TPM, nel momento in cui egli decide di prenderne il possesso durante una fase denominata *Take Ownership*. Prendere il possesso del TPM è un'operazione necessaria per l'owner se si vogliono sfruttare tutti i meccanismi e le funzionalità messe a disposizione dal TPM stesso. Tale processo consiste nella creazione di una chiave asimmetrica, la *Storage*

Root Key (SRK)[3], e nell'inseriremento di due segreti, l'*Owner Authorization* e la *Storage Root Key Authorization*, nel TPM. Al fine di garantire la confidenza nel processo di comunicazione e inserimento, questi segreti vengono cifrati con la parte pubblica dell'Endorsement Key e inviati al TPM interessato. L'inserimento di questi segreti è parte fondamentale per l'utilizzo di una TP; essi rappresentano dati di autorizzazione importanti che sono necessari per poter usufruire delle funzionalità descritte dalla specifica[4] e anche per poter creare *identità* associate ad una TP con un TPM particolare.

3.1.2 Attestation Identity Key

Il meccanismo delineato nel punto 2 consiste nel dichiarare che un'entità, a conoscenza di un segreto, ha precedentemente garantito per la genuinità del TP, del TPM e dell'incorporamento del TPM su quella particolare TP. Questa entità, come vedremo più avanti, si chiama *Privacy Certification Authority (P-CA)*.
Dotare una TP di *identità*, significa:

- Creare una coppia di chiavi crittografiche asimmetriche, l'*Attestation Identity Key*[5], all'interno del TPM.
- Le identity key sono usate soltanto per firmare dati prodotti dal TPM, come ad esempio dati prodotti dal processo di misurazione delle metriche d'integrità da parte del TPM, come descritto nella sezione 3.2.
- Le identity key possono essere usate anche per attestare altre chiavi protette e create dal TPM.
- L'owner del TPM può associare delle stringhe a queste chiavi; tali stringhe rappresentano il nome dell'identità.

[3] il ruolo di questa chiave viene affrontato nella sezione 3.3.

[4] alcuni di questi meccanismi verranno descritti nelle sezioni successive.

[5] useremo i termini Attestation Identity Key, AIK, identity key, TPM identity e platform identity per indicare la stessa cosa

- Un TPM può avere più di un'identità associata; in questo senso le identità rappresentano degli pesudonimi per identificare lo stesso TPM particolare, cablato come da specifica, su una particolare TP.

Privacy Certification Authority

La P-CA è una Certification Authority scelta dall'owner del TPM che garantisce la genuinità relativa all'identità di una Trusted Platform.

Una delle proprietà delle TP è quella di fornire fiducia nei dati da esse prodotti. Come già accennato nelle sezioni precedenti, alla base del processo di fiducia, esiste un meccanismo di *fiducia sociale*, che vede coinvolte entità che garantiscono per diversi aspetti delle TP, con garanzie mostrate sottoforma di certificati digitali o credenziali.

Ogni qualvolta si rende necessario attestare per la genuinità di una TP o dei dati prodotti da essa, è necessario esibire più certificati digitali e questo processo potrebbe rivelarsi tedioso e scomodo. A questo proposito, la specifica TCG ha previsto l'utilizzo di un'altra entità, la P-CA, che si preoccupa di rilasciare un certificato digitale per ogni identità che viene associata alla TP. Il rilascio di tale credenziale, l'*Identity Credential*, è subordinato alla verifica delle garanzie fornite dalle altre entità, verifica resa possibile esaminando le credenziali fornite. Soltanto dopo il processo di attestazione di una TP che coinvolge le entità descritte nella sezione 2.2.1 è possibile rilasciare un certificato, da parte della P-CA, che garantisce per l'identità associata ad una TP genuina.

L'*Identity Credential* prodotto dalla P-CA, garantisce per la genuinità di una TPM identity. La credenziale contiene, tra le altre, le seguenti informazioni:

- espressione che indica che questa è un'identity credential,
- etichetta che identifica il nome scelto per questa identity dall'owner del TPM,
- la parte pubblica della chiave identity asimmetrica creata dal TPM per conto del suo owner (vedere oltre),
- descrizione generale del TPM e sue proprietà di sicurezza (copia dell'Endorsement Credential),

- descrizione generale della piattaforma e sue proprietà di sicurezza (copia del Platform Credential),
- riferimento alla P-CA che ha emesso questa credenziale.

3.1.3 La generazione di Attestation Identity Key

Il protocollo usato per creare una AIK, si basa su due comandi TPM (protected capability) e due comandi non-TPM (facenti parte del TSS). Questo protocollo è stato scelto al fine di minimizzare il lavoro svolto dal TPM e utilizzare le potenze di calcolo della CPU principale ove non siano richieste operazioni sensibili.

Le entità coinvolte nella creazione delle identità sono: TPM owner, TPM, Trusted (platform) Support Service e P-CA.

Le fasi previste dall'algoritmo eseguito per creare e ottenere una TPM identity sono delineate qui sotto ed illustrate nella figure 3.1 e 3.2:

TPM_MakeIdentity: Il TPM owner esegue questo comando[6], per poter iniziare il processo di creazione di identity key. A questo proposito deve fornire:

- il dato di autorizzazione, in forma cifrata, per poter usare in futuro l'identità,
- un hash della stringa associata all'identity e della chiave pubblica della P-CA prescelta,
- informazioni riguardo i parametri relativi alla nuova identity key che deve essere creata. Queste informazioni riguardano, tra le altre, il tipo di chiave che si sta creando, identity key in questo caso, indicazione sulla proprietà di migrabilità della chiave[7] e il tipo di algoritmo usato per creare le chiavi.

Senza entrare nei dettagli più tecnici del comando, di cui se ne rimanda a [11], il TPM, dopo aver effettuato i controlli di validità del caso, genera una coppia di

[6] protected capability del TPM.

[7] le identity key *non* possono mai migrare da un TPM ad un'altro perchè il processo coinvolto nella loro creazione utilizza, come vedremo, l'Endorsement Key che indentifica un TPM particolare.

chiavi asimmetriche, AIK, e crea una struttura, `TPM_IDENTITY_CONTENTS` nella quale inserisce la chiave pubblica dell'identità appena creata insieme alla sua stringa identificativa. Il comando restituisce `TPM_IDENTITY_CONTENTS`, struttura dati firmata con la parte privata dell'AIK, insieme ad un'altra struttura, `TPM_KEY`, che contiene, tra le altre informazioni, la parte pubblica in chiaro dell'identità appena creata, e la sua parte privata cifrata con la parte pubblica della Storage Root Key. La parte privata dell'identity key viene cifrata per garantirne la sicurezza, mentre il perché viene usata proprio la SRK per cifrarla verrà delineata nella sezione 3.3.4.

In questo modo, quindi, il TPM crea un *identity-binding*, una firma che collega assieme la nuova chiave asimmetrica generata dal TPM, il nome scelto dal TPM owner associato a l'identity e la P-CA scelta sempre dall'owner, coinvolta nel processo al fine di garantire per la nuova AIK.

TSS_CollateIdentityRequest è usata al fine di poter recuperare e assemblare tutte le informazioni e credenziali necessarie alla P-CA per poter rilasciare il certificato che garantirà per l'identity appena creata dal TPM. Tali informazioni riguardano:

- identity-binding, la struttura `TPM_IDENTITY_CONTENTS` firmata che non ha mai lasciato il TPM ma che l'owner può facilmente ricostruire a partire dai dati ritornati dal precedente comando,
- una struttura che contiene le stesse informazioni della struttura firmata al punto precedente,
- le varie credenziali che attestano per la genuinità della piattaforma e TPM: Endorsement Credential, Platform Credential e Conformance Credential.

Identity Credential. In questa fase, i dati recuperati nel passo precedente vengono spediti, cifrandoli con la chiave pubblica della P-CA, dall'owner del TPM alla P-CA[8].

[8] per motivi di privacy, più che per sicurezza

La P-CA deve verificare la firma dell'identity-binding fornito usando la parte pubblica dell'identity e determinare se è consistente con i dati ricevuti (parte pubblica dell'identity key, stringa associata all'identità e chiave pubblica della P-CA). Se i dati sono consistenti, la P-CA è consapevole del fatto che deve essere lei a fornire un'attestazione per l'identità e non un'altra CA.

La P-CA non sa però se la chiave pubblica ricevuta rappresenta veramente un'AIK di un qualche TPM genuino descritto secondo le credenziali (potrebbe esser stata falsificata), quindi crea l'*Identity Credential* ma la cifra con una chiave simmetrica effimera che verrà successivamente cifrata con la parte pubblica dell'Endorsement Key che compare nell'Endorsement Credential ricevuta nella fase precedente. In questo modo soltanto un TPM genuino, al quale corrisponde quell'Endorsement Key sarà in grado di decifrare la chiave simmetrica effimera che verrà utilizzata successivamente per attivare l'identity.

L'Identity Credential così creata e cifrata viene spedita all'owner che la utilizzerà con il TPM nella prossima fase.

TPM_ActivateIdentity: il TPM owner esegue questo comando TPM passandogli come parametro la credenziale appena ricevuta.

Soltanto il "corretto" TPM sarà in grado di decifrare la credenziale usando la parte privata dell'Endorsement Key e quindi soltanto lui sarà in grado di recuperare la chiave di sessione effimera usata dalla P-CA per cifrare la credenziale creata per attestare l'identity. Il comando rilascia al chiamante la chiave simmetrica effimera in chiaro, così da permettere la decifratura dell'Identity Credential e quindi l'attestazione dell'identità tramite l'utilizzo del comando *TSS_recover_TPM_Identity*.

Le TPM identity sono usate per firmare alcune strutture dati generati dal TPM in modo tale da garantirne, grazie all'Identity Credential, la validità e la fiducia (che i dati sono stati prodotti da un TPM genuino su una TP genuina).

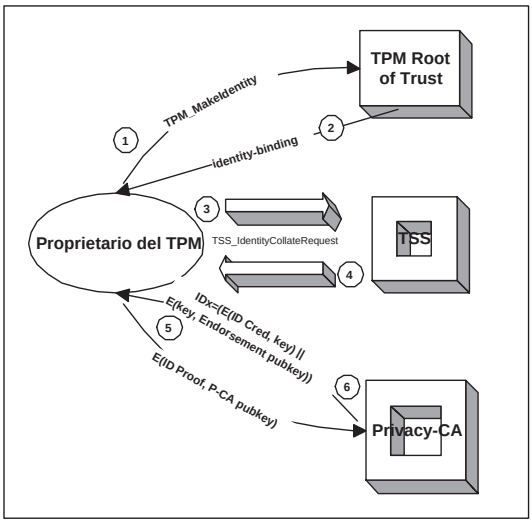

Figura 3.1: Creazione di TPM identity

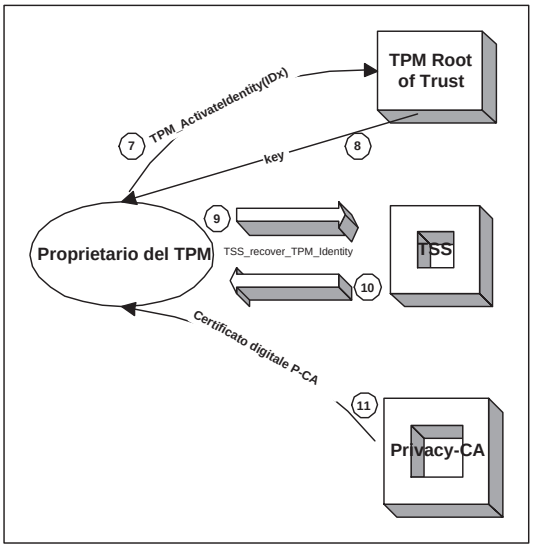

Figura 3.2: Recupero e attestazione di TPM identity

3.2 Metriche d'integrità

La nascita di molte discipline informatiche quali autonomic computing, grid computing, on demand computing e commercio elettronico ha portato la necessità di dover identificare in modo sicuro la piattaforma coinvolta nella comunicazione[9] (sezione 3.1) ma anche la genuinità del suo software al fine di permettere agli utilizzatori di poter decidere se fidarsi o meno di un determinato stato della piattaforma.

Per questo motivo, grazie a delle modifiche architetturali, introdotte da TCG 2.5, le Trused Platform, forniscono prova dello stato della piattaforma tramite misurazioni delle metriche d'integrità, che rappresentano una sorta di *impronta digitale* associata ad ogni software.

Le misurazioni delle metriche d'integrità rappresentano, forse, la caratteristica fondamentale di una TP:

1. forniscono prova del comportamento della piattaforma, determinandone quindi lo stato,

2. vengono usate dalla piattaforma stessa per permettere ad un'altro tipo di funzionalità, il protected storage (sezione 3.3), di divulgare segreti protetti soltanto in determinati stati della piattaforma, stati determinati dai proprietari dei segreti,

3. vengono usate da TTP per controllare che la TP target sia nel corretto stato. Una TTP si fida delle misurazioni d'integrità perchè sono state firmate dalla piattaforma usando una TPM identity (sezione 3.1) che attesta il fatto che le misurazioni quindi provengono da una TP fidata e genuina e quindi sono state effettuate correttamente. Rimane compito della TTP decidere se lo stato riportato in modo corretto è da considerarsi fidato o meno,

4. vengono usate anche dalla TP per implementare un meccanismo di *secure boot* controllando quindi che il processo di boot stia procedendo come aspettato. Tale processo verrà spiegato più avanti nella sezione corrente.

[9] concetto di platform identity.

L'integrità di un software o hardware è quindi una proprietà che indica se l'entità, o il suo ambiente, è stata modificata in un qualche modo non autorizzato.

3.2.1 Il processo di misurazione e i componenti coinvolti

Il processo di misurazione coinvolge molte entità, come illustrato nella figura 3.3, tra cui il CRTM, che, rappresentando una root of trust della TP, costituisce il punto di partenza della misurazione.

Il TPM è coinvolto nel processo di misurazione perché è grazie alla sua garanzia di genuinità (root of trust della TP e quindi attestato da TTP) che ci si può fidare del suo operato. Tuttavia il TPM si limita ad effettuare le misurazioni ma non rappresenta il coordinatore di questa attività. Due componenti principali del TPM sono usati per questo scopo, i PCR e i DIR.

PCR

Come già accennato nella sezione 2.5, i Platform Configuration Register rappresentano delle shielded location e un registro di questo tipo ha una dimensione di 20 byte, pari alla dimensione di un digest SHA-1. In particolare:

1. contiene il valore che riassume *tutti i risultati* di misurazione che sono stati presentati ad esso e *preserva l'ordine* in cui tali misurazioni sono state riportate,

2. un numero indefinito di risultati può essere memorizzato in un singolo registro così da permettere di considerare *tutti* i risultati di misurazione senza scartarne neanche uno.

Il processo di misurazione si riduce quindi a determinare l'integrità di un'entità software o hardware, calcolandone l'equivalente di un'"impronta digitale", un *digest*, usando una funzione di hash crittografica in modo tale che l'entità possa essere rappresentata ed indentificata in modo univoco attraverso questo digest. La funzione di hash crittografica prevista dalla specifica TCG è SHA-1[10].

[10]Secure Hash Algorithm

Il valore contenuto nei PCR quindi rappresenta il digest della concatenazione del digest del più recente risultato di misurazione riportato a questo PCR, con il valore di questo PCR prima di quest'ultimo "aggiornamento". In altri termini:

> Sia $h1$ il digest della misurazione appena eseguita su un componente che si vuole presentare (memorizzare) in *PCR*, allora
>
> $$PCR' = H(PCR||h1) \tag{3.1}$$
>
> rappresenta il nuovo valore associato al registro *PCR*

Il valore di un PCR è usato dalle altre funzionalità descritte dalla specifica TCG, nel seguente modo:

1. Una firma d'integrità usa questo valore per fornire prova della genuinità del processo di misurazione rappresentante lo stato della piattaforma, nel momento in cui i dati vengono firmati. Il TPM concatena i dati di dimensione di un digest con i valori del PCR interessati e li firma. Questo concetto di *integrity challenge e response*, verrà spiegato più avanti nella sezione 3.2.3.

2. Il protected storage usa questo valore per controllare se i segreti da esso protetti possono essere rivelati nello stato in cui si trova la piattaforma.

3. Il processo di secure boot usa questo valore per controllare se il processo di boot sta avanzando come pianificato e non ci sono stati tentativi di manomissione hardware o software.

Le misurazioni d'integrità vengono sempre calcolate, anche se il TPM è *disabilitato* (sezione 6.2) per poter avere una rappresentazione coerente delle integrità dell'ambiente nel caso venisse riabilitato a boot già avvenuto.

DIR

Anche il DIR è rappresentato all'interno del TPM come shielded location e viene usato per memorizzare, come il PCR, valori di digest, risultato della misurazione di una me-

trica d'integrità. Il registro DIR viene usato dal CRTM o da altri agenti di misurazione in un contesto di boot sicuro, caratteristica che sarà spiegata più avanti.

Come già accennato all'inizio della sezione, è possibile determinare due ambienti nei quali opera il processo di misurazione delle metriche d'integrità: ambiente pre-boot e ambiente post-boot.

3.2.2 Ambiente di misurazione pre-boot

L'ambiente pre-boot è un ambiente particolare nel quale ci si trova nel momento in cui si accende fisicamente la piattaforma di calcolo. La specifica TCG si preoccupa di dare dettami solo riguardo questo ambiente mentre lascia libera scelta sulla strategia da adottare per attuare il processo di misurazione nell'ambiente post-boot.

Nell'ambiente pre-boot non c'è ancora un kernel del sistema operativo in memoria quindi, normalmente, il primo componente al quale la piattaforma cede il controllo è il BIOS. In una TP, il BIOS è stato modificato per accogliere una delle due root of trust previste dall'architettura, il CRTM[11], che si preoccuperà di coordinare ed iniziare il processo di misurazione delle componenti del sistema, fino a quando non viene caricato ed eseguito il kernel del sistema operativo, come illustrato dalla figura 3.3.

La specifica TCG distingue due tipologie di differenti boot, sicuro o autenticato, adottate nel processo di misurazione delle componenti hardware e software della TP nella fase pre-boot. Indipendentemente dal tipo di boot adottato, comunque, lo scopo del processo di misurazione è quello di creare una catena di fiducia transitiva partendo dal CRTM, l'unico componente, insieme al TPM, considerato assiomaticamente fidato, come descritto nel processo di attestazione della TP (sottosezione 2.2.1 e di creazione dell'identità della stessa (sezione 3.1).

[11] parte del *BIOS Boot Block, BBB*, o il BIOS completo a seconda di come è stato progettato questo componente.

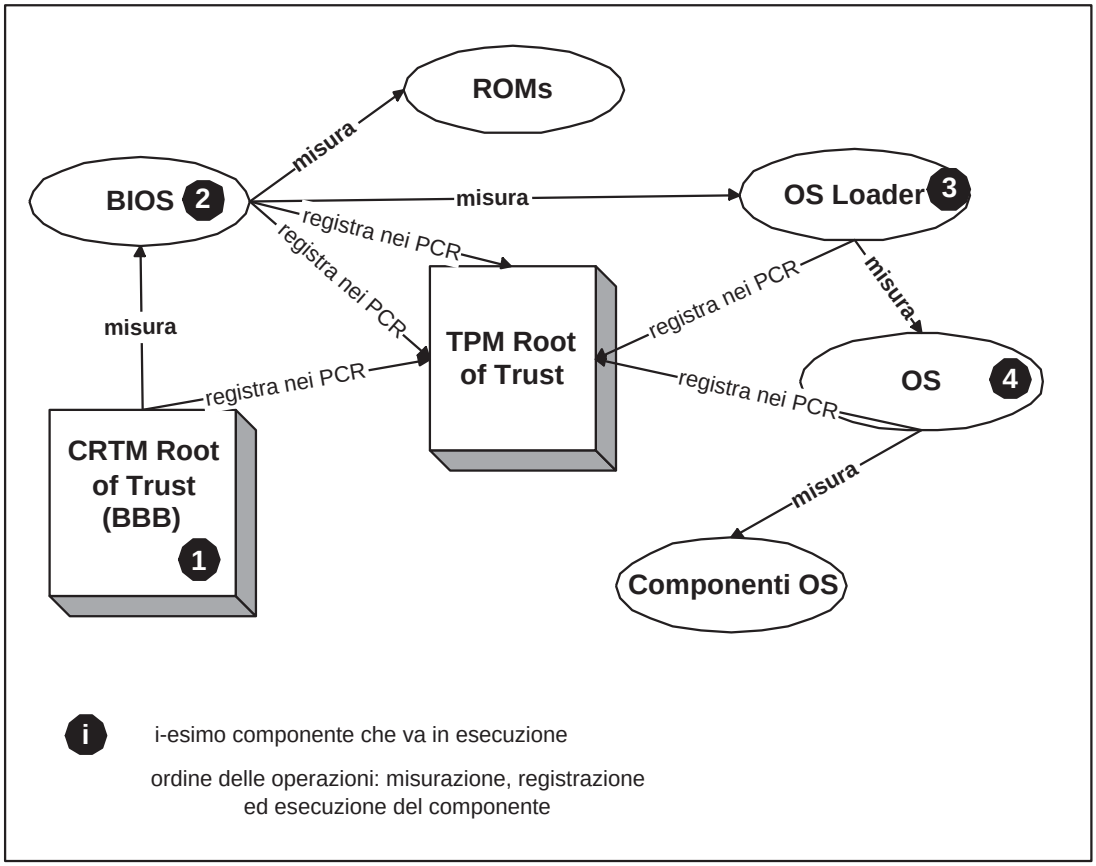

Figura 3.3: Catena di fiducia

Boot autenticato

Il CRTM inizia con il misurare l'integrità di alcuni componenti hardware[12] per poi passare l'esecuzione a questi componenti misurati, che si preoccuperanno di misurarne altri e così via, fino a quando non verranno misurati il *Master Boot Record* che, una volta eseguito, si preoccuperà di misurare il kernel del sistema operativo che verrà mandato in esecuzione successivamente.

Ogni passo di misurazione prevede semplici ma importanti punti:

[12]solitamente il CRTM inizia a misurare il POST BIOS, nel caso in cui il CRTM sia rappresentato dal BBB.

1. il componente che coordina il processo di misurazione, l'agente di misurazione, si prende carico di annotare, tramite comandi offerti dal TPM[13], nei PCR di competenza, le misure appena effettuate, secondo la "regola" 3.1, prima di mandare in esecuzione questi componenti.

2. l'agente di misurazione si preoccupa, sempre utilizzando le funzionalità del TPM, di annotare anche uno storico dettagliato dell'integrità appena misurata in una tabella non fidata del sistema[14] ma della quale è possibile controllarne l'integrità, applicando sempre la regola 3.1 agli elementi della tabella seguendo l'ordine di misurazione effettuata: se le integrità finali coincidono, si possono ritenere validi i valori memorizzati nella tabella.

3. l'agente di misurazione può cedere volontariamente l'esecuzione al componente misurato che diventerà in questo caso agente di misurazione ed eseguirà i passi appena visti, partendo dal punto 1. La misurazione, in questo ambiente pre-boot, avrà fine nel momento in cui viene misurato ed eseguito il kernel del sistema operativo

Il processo qui delineato, prende il nome di *boot autenticato*, in quanto è proprio durante la fase di boot che si costruisce, misurandola, l'integrità e quindi lo stato della TP fino al momento dell'esecuzione del kernel del sistema operativo.

Questo processo di misurazione lascia libero arbitrio alle entità utilizzatrici della TP riguardo la determinazione della fiducia nel sistema; il meccanismo di misurazione qui delineato si preoccupa di misurare l'integrità della TP in modo inopinabile, grazie all'utilizzo del TPM, root of trust. Che significato dare alle misurazioni effettuate e quindi determinare se per un certo utente la TP può essere considerata fidata[15] o meno è lasciata all'utilizzatore stesso.

---

[13] `TPM_SHA1Start`, `TPM_SHA1Update`, `TPM_Extend`, `TPM_CompleteExtend`.

[14] in questa fase la tabella designata fa parte dell'ACPI data la natura "povera" dell'ambiente operativo.

[15] ci sono le patch di sicurezza installate, il kernel è una versione particolare, il software annunciato è quello che mi aspettavo, ...

La specifica TCG non è l'unica ad aver proposto un meccanismo di boot autenticato; l'IBM si è mossa in questo campo, progettando e sviluppando un co-processore crittografico, l'IBM 4758 [18], capace di offrire funzionalità di boot autenticato, sicuro e altre caratteristiche molto simili a quelle dettate dalla specifica TCG, sebbene il meccanismo di boot autenticato e sicuro risulti essere più complesso e preveda la verifica di firme digitali ad ogni passo della misurazione.

Boot sicuro

Il processo di *boot sicuro* è molto simile a quello di boot autenticato; anche in questo caso il sistema, all'accensione, esegue il CRTM che, tramite la costruzione della catena di fiducie transitive, porterà all'esecuzione del kernel del sistema operativo. Tuttavia in questo tipo di boot, gli agenti di misurazione, oltre ad eseguire i passi delineati nella sottosezione precedente, si preoccupano di confrontare i valori dei PCR con il valore memorizzato nel DIR[16], registro delle shielded location, non volatile. Alla tabella precedente è quindi sufficiente aggiungere un punto intermedio:

2bis. se i valori dei PCR[17] e del DIR coincidono allora la proprietà d'integrità è conservata e il processo di boot può continuare al prossimo punto, altrimenti l'intero processo viene interrotto.

Questo tipo di coercizione è molto forte in quanto non lascia la scelta agli utilizzatori della piattaforma di determinare se lo stato raggiunto è da considerarsi fidato o meno; la decisione viene presa dalla TP, prima che il kernel del sistema operativo venga eseguito. Ne consegue che se una TP esegue un determinato kernel e il processo di boot eseguito è quello di boot sicuro, allora nessun componente di misurazione coinvolto nel processo del calcolo delle metriche d'integrità può essere stato alterato. L'integrità "matematica" della TP è assicurata.

[16] il registro è riempito con misurazioni effettuate sulla TP in uno stato, considerato dall'owner, fidato.

[17] viene effettuato in realtà un hash su tutti i PCR, in modo da ottenere un unico digest confrontabile con quello del DIR.

Lo stesso co-processore crittografico dell'IBM, il 4758, e il progetto AEGIS [14] possono adottare un meccanismo di boot sicuro, sebbene leggermente più complesso di quello previsto dalla specifica TCG.

3.2.3 Ambiente di misurazione post-boot

La specifica TCG descrive soltanto come deve essere effettuato il processo di misurazione delle metriche d'integrità relativo all'ambiente pre-boot, cioè fino al punto in cui viene caricato ed eseguito il kernel del sistema operativo. Una volta che il kernel è in esecuzione, per non render vano il processo di misurazione effettuato a partire dalla root of trust e poter mantenere una coerenza nelle misurazioni dell'ambiente che si sta eseguendo, è necessario che altri agenti di misurazioni procedano con l'attività per la quale sono stati preposti.

Come accennato, la specifica non prevede nulla in questa fase e lascia il compito ai ricercatori di proporre soluzioni che si avvalgano del TPM e quindi in grado di sfruttare le caratteristiche ulteriori offerte dall'architettura.

In questo senso, i ricercatori del *T. J. Watson Research Center* dell'IBM, del gruppo *Global Security Analysis Lab, GSAL* [2], hanno progettato e implementato un'architettura per la misurazioni d'integrità [26], basata sulla specifica TCG. I prossimi paragrafi offriranno una panoramica su tale architettura, evidenziandone i contributi, gli scopi e le basi del suo funzionamento.

Integrità

L'obiettivo dell'architettura proposta è quello di permettere ad un sistema remoto, il *challenger*, di dimostrare che un programma su un altro sistema, il sistema *attestato* di proprietà dell'*attestatore*, rappresenta sufficiente integrità per essere utilizzato. L'*integrità* di un programma è una proprietà binaria che indica se il programma o il suo ambiente è stato modificato in modo non autorizzato tale da non garantire più, ad un eventuale challenger, il suo corretto funzionamento.

Contributi

I contributi apportati dall'architettura proposta dai ricercatori dell'IBM, sono:

- un meccanismo di attestazione remoto del software presente su una TP, verificabile e non intrusivo che utilizza il TPM descritto dall'architettura TCG,
- un sistema di misurazione efficiente di eseguibili e oggetti dinamici,
- un meccanismo di attestazione del software presente, tracciabile che non richiede nuove modalità operative della CPU o nuovi sistemi operativi[18].

Obiettivi

Parte essenziale dell'architettura proposta consiste nella capacità dei challenger di poter determinare che la lista delle misurazioni prodotte e ottenute dalla TP, sia:

- *recente* e *completa*, cioè includa tutte le misurazioni effettuate fino al momento in cui viene eseguito il protocollo di *challenge*.
- *inalterabile*, cioè che le misurazioni effettuate sull'eseguibile caricato e sui dati statici ad esso associati non siano stati modificati in alcun modo.

Progettazione di un'architettura di misurazione di integrità

Prima di descrivere l'architettura proposta, è bene delineare il contesto nel quale tale architettura può operare. Le ipotesi non sono restrittive ma sono doverose. Tramite il processo descritto nella sezione 2.2.1 è possibile definire la piattaforma come una TP genuina, con un TPM genuino attestati dalle varie TTP. Il TPM non può impedire attacchi diretti all'hardware del sistema, quindi si assume che tali attacchi non facciano parte delle minacce. Si assume inoltre che il challenger possieda un certificato o una credenziale, l'identity credential, che associ una chiave pubblica RSA, AIK_{pub}[19], al TPM attestata dalla P-CA.

[18] resta inteso che l'architettura proposta *prevede* la modifica del kernel di un sistema operativo.

[19] AIK_{pub} identifica la parte pubblica dell'indentity key, mentre AIK_{priv} la sua parte privata.

Fatte queste assunzioni, è possibile descrivere i tre meccanismi principali dell'architettura:

Meccanismo di Misurazione implementato sul sistema attestato, la TP, determina quali componenti misurare dell'ambiente run-time, quando effettuare tali misurazioni e come mantenere in modo sicuro le misurazioni effettuate.

Meccanismo di Integrity Challenge che permette ai challenger autorizzati di recuperare la lista delle misurazioni effettuate, di verificarne la completezza e la coerenza in termini temporali.

Meccanismo di validazione delle integrità che si preoccupa di validare l'operato effettuato dal meccanismo soprastante, garantendo così la fiducia nei confronti di tutto il software e dati misurati nella Trusted Platform.

La figura 3.4 descrive l'interazione che intercorre tra i meccanismi sopra enunciati. Le misurazioni sono iniziate dagli agenti di misurazione che misurano, ad esempio, un file, memorizzando la misurazione effettuata in una lista ordinata nel kernel e riportando l'hash di questa *lista delle misurazioni* al TPM.

È possibile notare subito una differenza tra il meccanismo di misurazione pre-boot descritto dalla specifica TCG e quello proposto per l'ambiente post-boot dai ricercatori del T. J. Watson Research Center GSAL; mentre la specifica prevede che gli agenti di misurazione, "capitanati" dal CRTM, riportino per ogni oggetto (hardware, firmware o software) la misurazione effettuata[20] al TPM, l'architettura proposta dall'IBM riporta al TPM soltanto l'hash effettuato sulla lista delle misurazioni e non l'hash di ogni oggetto misurato. Questo approccio permette di impegnare al minimo il TPM, pur garantendo per l'integrità dell'integrità dell'ambiente misurato, fornendo prova di non alterazione della lista di misurazione.

Il meccanismo di *integrity challenge* permette ai challenger remoti di richiedere la lista delle misurazioni insieme alla firma dell'hash di tale lista (passo 1, figura 3.4). La firma è effettuata dal TPM utilizzando un'Attestation Identity Key (sezione 3.1.2); in

[20] tramite l'utilizzo di comandi messi a disposizione dal TPM.

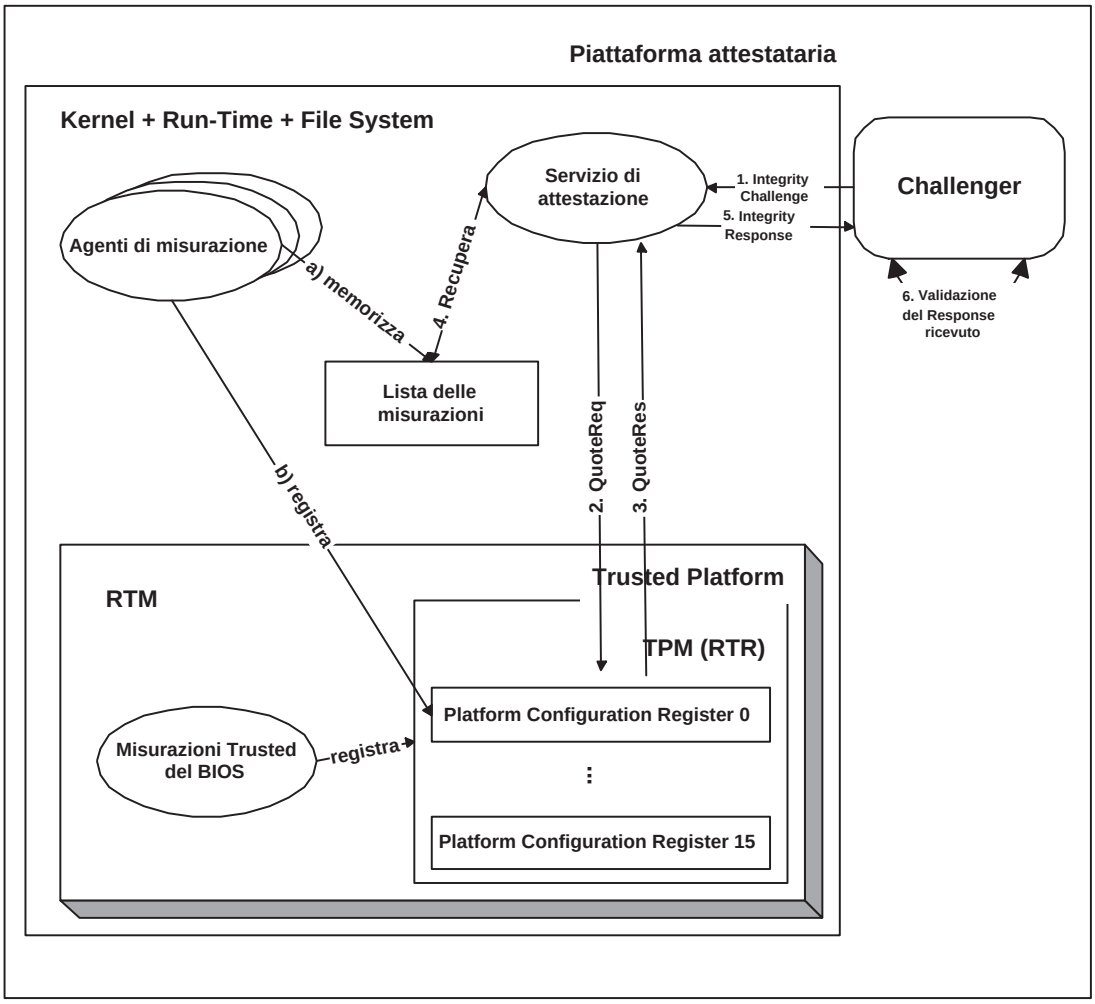

Figura 3.4: Misurazione d'integrità basata sul TPM

questo modo è possibile determinare la genuinità dei dati ricevuti e la loro integrità. La TP coinvolta nel challenge, recupera i dati firmati dal TPM (passo 2 e 3) e la lista delle misurazioni dal kernel (passo 4). Entrambi vengono spediti al challenger (passo 5) che alla fine può validare le informazioni ricevute, decidendo se fidarsi dell'ambiente software misurato in modo inopinabile dalla TP.

Meccanismo di Misurazione Questo meccanismo permette la misurazione degli eseguibili, nel momento in cui vengono caricati in memoria, e la possibilità di misurare anche i file contenenti dati sensibili per gli eseguibili. L'idea consiste nel sfruttare il meccanismo di misurazione pre-boot e, successivamente, permettere al kernel di misurare cambiamenti apportati a se stesso[21] e la creazione di processi utente[22]. Il codice così misurato può partecipare anch'esso all'attività, misurando altri componenti sensibili, quali le librerie dinamiche o gli script eseguibili.

La misurazione consiste nel calcolare un digest SHA-1 su tutto il contenuto dei file, al fine di identificare univocamente qualsiasi *eseguibile*[23] del sistema.

Gli hash così ottenuti sono memorizzati in una lista, la *lista delle misurazioni*, che rappresenta lo storico dell'integrità del software eseguito sulla TP. Modifiche a questa lista, sebbene non tecnicamente impedite, verrebbero rilevate grazie al controllo dell'integrità eseguito sull'intera lista. Infatti, a differenza del processo di misurazione effettuato nella fase pre-boot, viene memorizzato nel PCR del TPM soltanto il digest calcolato sulla lista delle misurazioni e non i digest contenuti in essa, al fine di migliorare le prestazioni e utilizzare il TPM in modo oculato. La memorizzazione del digest segue sempre la "regola" 3.1, prevista dalla specifica, in quanto la scrittura nei PCR avviene sempre utilizzando un comando esportato dal TPM. La misurazione e l'estensione del digest calcolato nel PCR viene effettuata *prima* di cedere il controllo al nuovo processo; in questo modo si garantisce l'integrità della lista (tramite PCR) anche se la stessa viene successivamente modificata dal processo maligno. Quindi un

[21] ad esempio tramite il caricamento di moduli kernel.

[22] tramite l'esecuzione della syscall fork(2) e similari o execve(2).

[23] con il termine, si intende indicare anche librerie dinamiche, script eseguibili, etc oltre che esegubili in senso stretto (di tipo ELF ET_EXEC).

sistema corrotto può manipolare la lista delle misurazioni, ma questo è rilevato semplicemente ricalcolando l'hash sulla lista e confrontandolo con quello contenuto nel PCR.

Meccanismo di Integrity Challenge Il protocollo di *integrity challenge* descrive come delle terze parti che vogliono determinare lo stato della TP, recuperano la lista delle misurazioni e le informazioni di validazione dalla piattaforma stessa. Il protocollo deve proteggere dalle seguenti minacce:

1. *Reply attack*: un sistema attestatario (TP) maligno può rispedire informazioni di attestazione (la lista delle misurazioni più i valori prodotti dal TPM) riferite ad un tempo antecedente la sua compromissione.

2. *Manomissione*: una TP maligna o un attaccante intermediario, *man in the middle (MITM)*, può modificare la lista delle misurazioni e i valori prodotti dal TPM prima che queste vengano consegnate alla parte che ne fa richiesta.

3. *Mascheramento*: una TP maligna o un MITM può rimpiazzare i valori originali della lista delle misurazioni o dei valori prodotti dal TPM con quelli di una TP non compromessa.

Il protocollo sotto riportato, illustra il meccanismo di *challenge* usato da una terza parte *C* per validare in modo sicuro le integrità proposte da un sistema attestatario *AS*, la Trusted Platform.

1. *C*: crea un *nonce*[24] di 160 bit.

2. $C \rightarrow AS$: ChReq(*nonce*).

3a. *AS*: carica AIK_{priv} nel TPM (protetta).

3b. *AS*: recupera $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$.

3c. *AS*: recupera la lista delle misurazioni, *ML*.

[24] numero pseudo casuale

4. $AS \rightarrow C$: ChRes(*Quote, ML*).

5a. C: determina la fiducia della chiave AIK recuperando $cert(AIK_{pub})$, il certificato digitale associato ad essa.

5b. C: valida la firma $sig\{PCR, nonce\}_{AIK_{priv}}$.

5c. C: valida il *nonce* e la *ML* usando i PCR ricevuti.

Nei punti 1 e 2, C crea un numero pseudo-casuale di 160 bit, detto *nonce*, e lo spedisce al sistema attestatario *AS*, la TP, in un messaggio di richiesta di challenge, ChRes. Nel punto 3, la TP carica la chiave RSA protetta e generata dal TPM, come spiegato nella sezione 3.1, AIK_{priv}, nel TPM stesso. Questa chiave, come vedremo nella sezione 3.3, è cifrata con un'altra chiave creata e memorizzata nel TPM, la *Storage Root Key (SRK)*, introdotta nel processo di Take Ownership nella sottosezione 3.1.1. A questo punto, *AS* richiede l'esecuzione dello pseudo-comando *Quote* al TPM che si preoccuperà di firmare i PCR selezionati e il *nonce* fornito da C con la chiave privata AIK_{priv}. Per terminare il punto 3, *AS* recupera la lista *ordinata*[25] di tutte le misurazioni dal kernel e risponde con un messaggio di risposta al challenge, ChRes (punto 4), includendo la firma effettuata sulle misurazioni e sul *nonce*, insieme alla lista. Nel punto 5a, C recupera il certificato $cert(AIK_{pub})$ che attesta per l'identity key, rilasciato dalla P-CA (sezione 3.1); questo permette al challenger di fidarsi di AIK_{pub} rendendo vano ogni tentativo di *mascheramento* da parte di qualche MITM o TP maligna. Nel punto 5b, C verifica la firma effettuata nel punto 3b, garantendo la consistenza e quindi l'integrità e la "freschezza" dei PCR; infatti, grazie al *nonce*, si ha la certezza che il messaggio è recente e si riferisce alla richiesta corrente, vanificando, in questo modo, ogni tentativo di attacco di tipo *reply*. Inoltre, attraverso questa verifica, si ha la garanzia implicità che nessun manipolazione dei dati è avvenuta, garantendo la genuinità dell'informazione ricevuta da C.

Meccanismo di validazione delle integrità Spetta al challenger decidere se fidarsi o meno dello stato riportato dal protocollo di Challenge/Response. Questa de-

[25] per il modo in cui vengono calcolate le misurazioni, è necessario preservare l'ordine.

cisione si riconduce a confrontare ogni singolo elemento della lista delle misurazioni ricevuta dalla TP, con una lista di misurazioni *fidata*[26], mantenuta dal challenger. Tramite il protocollo sopra descritto, si ottiene protezione da tentativi di mascheramento, manipolazione e attacchi reply sui dati d'integrità spediti ma la decisione sul contenuto di tali dati, spetta, ovviamente, al challenger utilizzatore della TP.

3.3 Protected Storage

La root of trust TPM è un dispositivo crittografico resistente alle manomissioni ed è quindi una scelta naturale che esso fornisca un servizio di riservatezza[27] dei dati alla TP.

Sfortunatamente, vincoli sui costi di fabbricazione impongono che il TPM abbia delle shielded location limitate[28] che devono essere usate quando c'è effettivo bisogno e non per proteggere dati che potrebbero essere protetti usando metodi convenzionali di cifratura al di fuori del TPM.

Per ovviare a questo limite, il TPM fornisce un servizio, il *Protected Storage*, che rappresenta un *portale crittografico* ad una quantità di dati illimitati che devono essere mantenuti confidenziali, riservati, al di fuori del dispositivo.

Il TPM è stato pensato e progettato per permettere la protezione di segreti particolari, come chiavi, che possono contribuire alla protezione di segreti di tipologia e dimensione arbitrari. Gli oggetti protetti dal TPM sono rappresentati, quindi, principalmente da *chiavi* e *dati arbitrari*.
Memorizzare dati al di fuori del TPM determina i seguenti vantaggi:

1. facilità nel processo di migrazione di dati protetti da una TP ad un'altra.

2. la cifratura di grosse quantità di dati, *bulk encryption* viene fatta dalla CPU principale della TP e può essere effettuata, se si vuole, solo quando la TP si tro-

[26] la costruzione di tale lista può avvenire in vari modi, ad esempio, appena dopo l'installazione della macchina, in modo tale da fidarsi dell'integrità del software installato.

[27] confidenzialità, cifratura.

[28] la specifica TCG v1.1 prevede la presenza di 16 PCR e almeno 1 DIR.

va in uno stato particolare; si trasforma la TP in una specie di co-processore crittografico.

3. non ci sono problemi su eventuali insidie di import/export legate al mondo della crittografia.

3.3.1 Caratteristiche concettuali

Il meccanismo di Protected Storage può proteggere, come accennato, sia dati arbitrari che chiavi. I dati arbitrari sono rivelati dal TPM all'utente, mentre un TPM usa le chiavi internamente e *non l'esporta* mai[29].

Le caratteristiche offerte dal servizio di protected storage sono molteplici e possono essere usate a seconda delle necessità:

1. il protected storage permette di memorizzare chiavi private usate per firmare, in modo tale che il TPM possa utilizzarle senza esportarle mai, in chiaro, alla piattaforma.

2. le chiavi usate per la bulk encryption o i dati arbitrari di autorizzazione possono essere memorizzati in modo tale da render necessaria la cooperazione del TPM per poterle utilizzare.

3. i dati protetti possono essere memorizzati in modo da permettere la loro duplicazione oppure no; questa opportunità è comunque controllata dal proprietario dei dati che si avvale delle funzionalità offerte dal protected storage.

4. i dati protetti possono essere memorizzati in modo tale da impedirne l'uso a meno che la piattaforma non si trovi in uno stato particolare, fidato, desiderato.

È importante notare che tutti gli oggetti protetti dal TPM tramite questa funzionalità, vengono memorizzati *fuori* dal TPM stesso in forma cifrata e secondo una gerarchia ad albero, come illustrato in figura 3.5. La gestione della gerarchia degli oggetti,

[29] il discorso non vale per le chiavi create esternamente dal TPM.

tuttavia, è lasciata al TSS, il software di supporto e non al TPM che si limita solo a fornire un API per garantire un corretto supporto a queste funzionalità.

La gestione delle chiavi merita particolare attenzione, rispetto ai dati arbitrari perchè le chiavi sono usate per proteggere altre chiavi, dati arbitrari e per fornire quel concetto di fiducia introdotto dalle TP.

3.3.2 Funzionalità crittografiche

Il TPM utilizza funzioni crittografiche per assicurare che gli oggetti protetti, creati dal protected storage abbiano proprietà di confidenzialità e integrità.

Le funzionalità che il TPM garantisce attraverso l'uso di funzioni crittografiche, sono:

1. confidenzialità, riservatezza, sotto forma di cifratura, agli oggetti protetti. A questo scopo il TPM utilizza *crittografia asimmetrica* invece che simmetrica per due principali motivi:

   - la crittografia asimmetrica viene già utilizzata dal TPM per creare le TPM identity, come spiegato nella sezione 3.1. In questo modo si riutilizzano blocchi di firmware, come da buona pratica dell'ingegneria del software, e si mantengono contenuti i costi di realizzazione del dispositivo.
   - il testo in chiaro può esser originato al di fuori del TPM e la crittografia asimmetrica ne permette la cifratura senza rivelare la chiave di decifratura.

   Purtroppo questa scelta porta anche dei difetti, in particolare:

   - tempi più lunghi necessari per la generazione di chiavi asimmetriche e per effettuare le operazioni di cifratura/decifratura.
   - viene imposto un vincolo sulla dimensione degli oggetti protetti dal TPM; per ragioni matematiche tale dimensione deve coincidere quella della chiave usata per l'operazione crittografica[30]

[30] se l'oggetto protetto è più piccolo, è necessario formattare i dati in modo particolare per evitare attacchi sulle chiavi.

⇒ la dimensione di un oggetto protetto dal TPM è molto limitata. Per questo motivo TCG ha deciso di rendere le strutture dati utilizzate dal protected storage grandi quanto la dimensione di una chiave RSA a 2048 bit.

2. integrità degli oggetti dati al fine di rivelarne l'eventuale alterazione. Il protected storage offre un controllo implicito di consistenza per i dati decifrati, nel momento in cui vengono controllate le autorizzazioni necessarie per poter usare la chiave che permette la decifratura dei dati in questione. Infatti un oggetto protetto dal TPM include già un dato di autorizzazione cifrato che viene confrontato con il dato di autorizzazione presentato al TPM al fine di provare (al TPM) conoscenza dell'autorizzazione necessaria per usare l'oggetto. La probabilità di una corrispondenza delle autorizzazioni dopo la decifratura di un oggetto protetto corrotto, è piccola perchè la dimensione del dato di autorizzazione coincide con quella di un digest SHA-1. Una descrizione dettagliata dei protocolli di autorizzazione alla base di questo funzionamento è necessaria per capire a fondo il meccanismo; tale descrizione viene affrontata nel capitolo 7.

3.3.3 Migrabilità e non migrabilità

La garanzia sulla genuinità e consistenza delle chiavi e dati migrabili, oggetti protetti dal TPM, è rappresentata dalla fiducia che gli utilizzatori della TP hanno nel proprietario degli oggetti perché possono essere stati migrati tante volte e non c'è garanzia, se non ci si fida del proprietario, che esse siano state migrate in modo corretto verso un altro TPM.

Gli oggetti chiavi, *migrabili*, possono essere creati fuori o all'interno del TPM. Gli oggetti chiavi, *non migrabili*, possono essere creati solo all'interno del TPM.

Senza entrare nel dettaglio del processo, il proprietario di una chiave migrabile crea il dato di autorizzazione necessario alla migrazione, ma l'autorizzazione per chiavi *non* migrabili (non relativo all'uso, ma sempre al concetto di migrazione) è segreto e conosciuto solo dal TPM.

Siccome le chiavi del TPM non lasciano mai il dispositivo sottoforma di testo in

chiaro e le chiavi non-migrabili sono create dal TPM al suo interno, questa autorizzazione di "non-migrazione", *tpmProof*[31], è sufficiente a limitare l'uso di tali chiavi solo nel TPM che le ha generate, essendo impossibile determinare tale segreto.

Il TPM non usa dati arbitrari e non ne crea. Gli oggetti TPM che rappresentano dati arbitrari sono sempre migrabili perchè sono sempre generati e usati *fuori* dal TPM. Tuttavia anche questi oggetti possono essere considerati non-migrabili se il loro proprietario ne ha creato solo una copia e l'ha incapsulata (memorizzata) sotto una chiave non-migrabile. Il proprietario può sempre duplicare il testo in chiaro rappresentante il dato ma, poiché il dato è memorizzato sotto una chiave non-migrabile, impedisce al TPM di usare meccanismi definiti dalla specifica TCG per duplicarlo.

3.3.4 La gerarchia degli oggetti protetti

Un oggetto, memorizzato al di fuori del TPM viene trasformato come un oggetto protetto che può essere memorizzato ovunque. Ogni oggetto siffatto è cifrato da una chiave di cifratura che solitamente è memorizzata al di fuori del TPM, anch'essa come oggetto protetto.

> ⇒ Gli oggetti protetti dal TPM formano una gerarchia ad *albero* dove ogni figlio è cifrato con la chiave di cifratura del padre. L'albero è radicato alla *Storage Root Key, SRK*, non migrabile, creata durante la fase di Take Ownership (sezione 3.1.1). Tale gerarchia è illustrata nella figura 3.5.

Gli oggetti TPM rappresentanti nodi interni nell'albero, contengono chiavi, le *storage key*, che sono usate per cifrare/decifrare i nodi figli. Siccome è ritenuta pratica scorretta usare la stessa chiave per eseguire operazioni di cifratura e firma, il TPM si rifiuta di usare una signature key come storage key e viceversa, così come si rifiuta di usare una chiave migrabile come parent di una chiave non-migrabile.

I dati arbitrari sono rappresentati dalle foglie dell'albero, così come le chiavi usate per firmare; il TPM non usa mai dati come chiavi, anche se i dati possono rappresentare chiavi per l'applicazione che ha protetto questi segreti.

[31] è un numero pseudo casuale.

Per poter utilizzare un oggetto protetto dal TPM, è necessario determinare un *cammino* tra l'SRK e l'oggetto in questione. Supponiamo di voler utilizzare la chiave denominata *child1*; è necessario caricare l'SRK nel TPM per poter sbloccare la chiave *parent1*, che rappresenta l'oggetto parent di child1. L'utilizzo dell'SRK è vincolato alla conoscenza del suo dato di autorizzazione, ovviamente, ma una volta conosciuto, sarà possibile caricare anche *parent1*, della quale è necessario conoscere l'autorizzazione, che servirà per decifrare la chiave *child1* per poter permetterne l'utilizzo.

Se l'oggetto protetto finale è a sua volta una chiave, può essere usato per cifrare, e quindi "avvolgere", un altro oggetto TPM o per firmare dati[32]. Invece, se l'oggetto protetto finale è un dato arbitrario allora il TPM lo esporta all'entità che ne conosce il segreto di autorizzazione.

A causa di problematiche dovute al tipo di codifiche utilizzate per evitare particolari attacchi alle chiavi ([7], [8] e [9]), la specifica definisce diverse tipi di chiavi:

Storage key. Usate da applicazioni conformi alla specifica TCG, per cifrare altre chiavi e dati arbitrari che usano caratteristiche avanzate dell'architettura, quali il *sealing*, descritto nella sottosezione 3.3.5.

Signature key. Usate da applicazioni conformi alla specifica TCG, per firmare dati arbitrari.

Identity key o AIK. Usate da applicazioni conformi alla specifica TCG allo scopo di dimostrare che i dati prodotti dalla TP sono integri e originati da un TPM genuino. Le AIK sono sempre figli della Storage Root Key al fine di fornire un accesso il più veloce possibile a queste chiavi, anche nell'ambiente pre-boot, qual'ora dovesse essere necessario.

Bind key. Usate da applicazioni "moderne" che non vogliono avvalersi delle caratteristiche avanzate previste dalla specifica TCG.

[32] dipende dal tipo di chiave, ovviamente. Non è possibile eseguire entrambe le operazioni con la stessa chiave generata dal TPM.

Legacy key per applicazioni che vogliono usare la stessa chiave per cifare e firmare; sono applicazioni *legacy*[33], non conformi alla specifica TCG.

Sintetizzando:

- le signature key possono firmare dati.
- le identity key possono firmare dati che sono sotto il controllo, e quindi generati, del TPM.
- le storage key possono cifrare altre chiavi e/o dati arbitrari in un primo modo, sfruttando tutte le funzionalità della TP.
- le bind key possono farlo in un secondo modo senza sfruttare le caratteristiche avanzate dell'architettura.
- le legacy key possono cifrare dati come le bind key ma anche in un altro modo e possono anche firmare. A causa di possibili attacchi crittografici portati su chiavi che si comportano in questo modo, il loro uso è sconsigliato.

3.3.5 Seal e Unseal

Quando si usa il protected storage per memorizzare dati arbitrari o chiavi, è possibile dichiarare lo stato dell'ambiente software futuro che deve esistere prima che il TPM permetta il rilascio dell'oggetto protetto, per essere usato. Questa caratteristica aggiunge una dimensione extra al controllo degli accessi: la capacità di dichiarare l'ambiente software nel quale il segreto sarà usato. L'ambiente software desiderato è rappresentato dai valori dei PCR.

Come al solito, per semplicità e per contenere i costi, il TPM esegue solo i compiti che sono sensibili dal punto di vista della sicurezza, demandando gli altri incarichi a software esterni al TPM, ma sempre verificabili.

[33] applicazioni "vecchie".

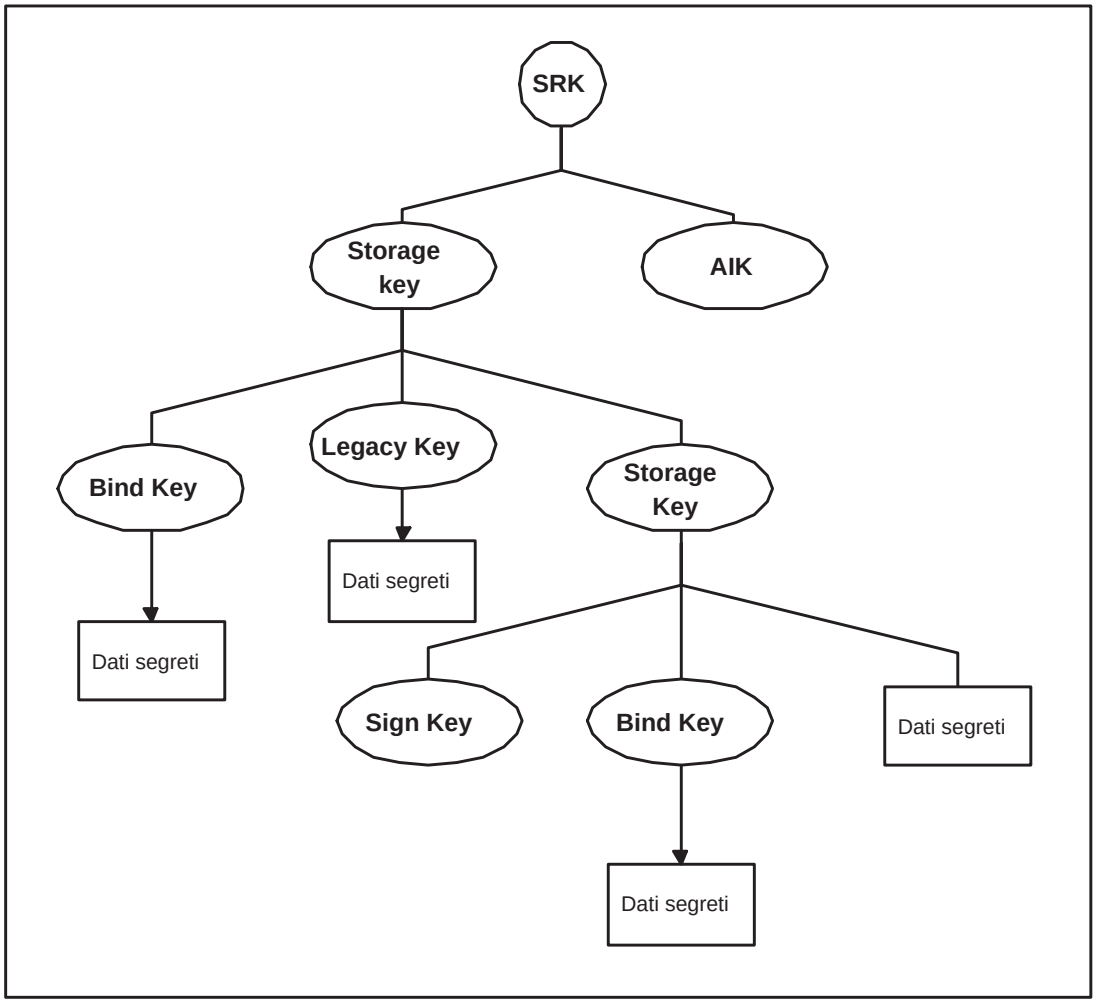

Figura 3.5: Gerarchia degli oggetti del Protected Storage

Seal[34] è il nome dato da TCG al processo di blocco di un segreto ad una particolare configurazione software, in modo da impedirne l'*unseal* da parte del TPM a meno che la TP non sia proprio in quello stato voluto, specificato, al momento del seal.
I vantaggi nell'eseguire il sealing di chiavi e dati possono essere molteplici:

- il sealing di dati arbitrari ne impedisce l'uso a meno che la piattaforma non sia in uno stato particolare, stato probabilmente giudicato fidato da parte del proprietario dei dati.
- il sealing di signature key e legacy key è utile in quanto impedisce l'utilizzo di tali chiavi a meno che la piattaforma non si trovi nello stato desiderato.
- il sealing di storage key, legacy key e bind key è un modo semplice per bloccare *tutti* i figli di quelle chiavi, siano essi dati arbitrari, bind key e legacy key.

Gli oggetti del TPM "sigillati" alla TP, usano tre campi speciali per questo scopo; una *bitmap* che seleziona i PCR di interesse e un digest sulla bitmap e sul valore dei PCR scelti dalla bitmap, *digestAtCreation* e *digestAtRelease*.

digestAtCreation registra i valori dei PCR selezionati nel momento in cui l'oggetto TPM viene creato. L'altro campo, digestAtRelease, registra il valore finale che i PCR devono avere per far si che il TPM permetti l'utilizzo dell'oggetto.

digestAtRelease è usato dal TPM per controllare che possa eseguire l'operazione di unseal, mentre digestAtCreation può servire a qualche software nel caso in cui voglia sapere se l'oggetto è stato creato in un particolare stato considerato "integro" della TP.

È possibile individuare dei passi fondamentali eseguiti dall'utilizzatore della TP, dal TPM e dal TSS, durante le fasi di seal e unseal:

Seal. Quando si crea un oggetto protetto dal TPM, il software di gestione dell'albero radicato con la SRK, il TSS, presenta al TPM una lista di indici rappresentanti i PCR di interesse, la bitmap, e un digest sia sulla lista che sui valori di questi registri. Un esempio del valore di *digestAtRelease*, in pseudo C, potrebbe essere:

[34] letteralmente sigillatura, sigillo.

```
/*
 * 0000 0000 0000 1010
 * sono interessato solo in PCR1 e PCR3
*/
bitmap = 0xA;

/*
 * stato dei PCR desiderati: X e Y rappresentano
 * i digest che si vuole avere nei PCR selezionati
 * per poter rilasciare l'oggetto protetto
*/
PCR1_target = X; PCR3_target = Y;

si presenta al TPM

   1.  bitmap
   2.  H(bitmap || PCR1_target || PCR3_target)
```

Una volta ricevuto il comando contenente le informazioni sopra determinate, il TPM calcola un digest in tempo reale su quella stessa lista e sul valore corrente dei PCR indicati nella bitmap. In questo modo viene calcolato *digestAtCreation*.

Il TPM memorizza nell'oggetto da proteggere, insieme ad altre informazioni, la bitmap dei PCR prescelti, digestAtCreation, digestAtRelease. L'oggetto viene pertanto cifrato con la chiave[35] dell'oggetto parent (nell'albero gerarchico) scelto e viene restituito al chiamante come oggetto protetto.

Il TSS si preoccupa di ottenere una lista dei valori correnti dei PCR dal TPM, gli stessi usati per creare digestAtCreation, e la memorizza, insieme alla bitmap, con l'oggetto protetto che può essere registrato ovunque, in un posto anche non fidato (l'oggetto è cifrato e si viene a conoscenza solo dei PCR attuali).

Unseal. Quando si decifra un segreto, sia esso un oggetto protetto riferito a dati arbitrari o a chiavi, il TPM calcola il digest sulla *lista* dei PCR e sul loro *valore*.

[35] soltanto di tipo Storage Key.

Se il digest è lo stesso di digestAtRelease presente nell'oggetto protetto, il TPM sblocca il segreto, decifrandolo, all'interno di una shielded location, e permettendone l'uso. In caso contrario questo non avviene e il segreto rimane bloccato e protetto dal TPM.

Utilizzo. Dopo che un oggetto protetto, "sigillato", è stato caricato o restituito dal TPM, il software di gestione, il TSS ad esempio, può verificare lo stato dei programmi della TP quando questo oggetto è stato creato.

Il fatto che il TPM abbia potuto decifrare l'oggetto ed eseguirne l'unseal significa, oltre a garantire che lo stato della TP è quello desiderato, che digestAtCreation, presente nell'oggetto protetto è proprio quello calcolato dal TPM nel momento in cui l'oggetto è stato cifrato. Il TSS può usare i valori dei PCR e la bitmap memorizzata con l'oggetto durante l'operazione di seal, calcolarne l'hash e confrontare questo valore con digestAtCreation. Se i due valori sono uguali allora la lista dei valori dei PCR rappresenta lo stato del software registrato al momento della creazione dell'oggetto, altrimenti no.

Il controllo del campo digestAtRelease è importante al fine del rilascio del segreto, ma anche digestAtCreation merita la sua importanza in quanto permette di capire se lo stato in cui è stato protetto l'oggetto era lo stato voluto, fidato o meno. Le condizioni per usare un oggetto protetto mediante "sealing", sono garantite dal TPM. L'autenticità dei valori dei PCR presenti nell'oggetto al momento della cifratura dipendono da chi ha creato l'oggetto cifrato. Se è stato creato dal TPM allora la sua autenticità è garantita, altrimenti dipende.

$\Rightarrow$ soltanto dati arbitrari e chiavi non-migrabili che sono state sigillate dal TPM garantiscono la descrizione di uno stato software autentico, al tempo della creazione dell'oggetto. Nel caso di chiavi migrabili, questa fiducia è da imputare al proprietario della chiave. Nel caso di dati arbitrari il problema non sussiste perchè il TPM non permette di eseguire operazione di unseal di dati arbitrari che son stati sigillati al di fuori del TPM stesso.

Per meglio spiegare questi punti, è necessario dare uno sguardo alle funzionalità messe a disposizione dal TPM per eseguire operazioni di seal/unseal su dati e chiavi; queste sono diverse perchè gli oggetti in questione sono di diversa natura e alcuni possono essere creati fuori dal TPM ed associati, successivamente, ad esso.

Le principali differenze tra le operazioni di seal/unseal effettuati su dati o chiavi sono delineate nella lista sottostante.

Dati arbitrari. `TPM_Seal` e `TPM_Unseal` sono i comandi del TPM usati per lavorare su dati arbitrari che sono cifrati usando storage key non migrabili.

I dati arbitrari cifrati con chiavi bind o legacy non contengono i campi extra, bitmap e digest, necessari per descrivere la funzionalità di seal; pertanto per fare il seal di questi oggetti è necessario operare in modo indiretto:

- si effettua il seal della chiave bind o legacy che li protegge.
- si effettua il seal della storage key che cifra le chiavi bind o legacy.

Quindi i dati arbitrari memorizzati da un'applicazione conforme alla specifica TCG possono essere sigillati usando una storage key non migrabile. In questo caso i dati non possono essere duplicati usando i meccanismi definiti dalla specifica anche se è possibile recuperare il testo in chiaro contenuto all'interno dell'oggetto protetto, effettuando l'operazione di unseal su di esso.

Chiavi. `TPM_CreateWrapKey` è un comando del TPM che crea una nuova chiave nel chip, mentre `TSS_WrapKeyToPcr` è una funzione del TSS, eseguita fuori dal TPM.

Non esiste un comando per eseguire l'unseal delle chiavi perchè questa operazione viene eseguita automaticamente dal TPM quando la chiave viene usata; ogni volta che la chiave protetta viene usata, il TPM verifica lo stato del software della TP, al fine di permettere l'utilizzo o meno dell'oggetto.

Tutti i tipi di chiavi possono essere legati a dei PCR o no. Il proprietario di chiavi *migrabili* può comunque duplicarle a proprio piacimento, come testo in chiaro o come oggetti protetti; ovviamente i valori PCR che hanno senso su

una piattaforma potrebbero non averlo su un'altra e, a differenza delle chiavi migrabili, quelle *non migrabili* non possono essere duplicate perchè sono create e conosciute solo da un TPM specifico.

Le chiavi bind, legacy o storage possono essere migrabili o no. Anche in questo caso, la fiducia sul valore del campo digestAtCreation ricade sul TPM, per gli oggetti non migrabili, o sul proprietario della chiave, per quelle migrabili.

Il meccanismo descritto è abbastanza complesso ma nello stesso tempo dovrebbe garantire maggior sicurezza per i dati protetti, dati per i quali il TPM non ne conosce comunque la gerarchia nell'albero, gerarchia costruita e gestita dal TSS. Rimane tuttavia un problema irrisolto dalla specifica TCG al meccanismo descritto. Il problema riguarda il meccanismo di seal; non risulta chiaro, infatti come sia possibile alterare digestAtRelease in modo genuino, nel momento in cui è necessario un aggiornamento permanente del sistema software, come un aggiornamento del sistema operativo della TP. L'unica soluzione disponibile, ma completamente onerosa e tediosa, è quella di effettuare l'unseal di tutti gli oggetti protetti, aggiornare il sistema e ripetere l'operazione di seal. Soluzione che "lascia il tempo che trova".

Autorizzazione

Come già accennato precedentemente, i protocolli di autorizzazione che regolano le attività più critiche del TPM sono descritti ed analizzati nel capitolo 7; basti pensare per il momento che, come descritto brevemente nella sezione 2.5, esistono dei dati di autorizzazione associati agli oggetti protetti dal TPM, la cui conoscenza ne permette l'utilizzo.

Esistono due tipi diversi di autorizzazioni associati con oggetti nel protected storage. Un primo tipo dimostra il privilegio di *usare* l'oggetto, l'altro il privilegio che l'oggetto possa essere *migrato* e quindi esportato al di fuori del TPM, magari in altre TP. La separazione di questi valori di autorizzazione permette di separare, volendo, le entità che usano l'oggetto da quelle che lo gestiscono. La scelta di questi dati di autorizzazione è sempre lasciata al proprietario del dato. La specifica TCG detta soltanto i

vincoli tecnici relativi a questi dati: la dimensione del dato di autorizzazione, pari a 20 byte, è la stessa di un hash SHA-1.

L'uso dei dati di autorizzazione è lasciato a generiche funzioni di autorizzazione del TPM, funzioni in grado di creare sessioni all'interno delle quali eseguire comandi che fanno uso di tali dati. Inoltre, il campo riferito a tale dato, viene anche usato in fase di creazione dell'oggetto, mediante l'utilizzo di altri protocolli, che verranno descritti sempre nel capitolo 7.

Come già spiegato, gli oggetti possono anche non avere bisogno di un'autorizzazione per essere utilizzati, ma, visto che il dato di autorizzazione è utilizzato implicitamente come garante dell'integrità dell'oggetto, è consigliabile, dove non si vuole specificare l'autorizzazione, metterne una *conosciuta*, *pubblica* così da non render vano il controllo d'integrità e permettere allo stesso tempo, l'utilizzo dell'oggetto da parte di tutti.

Parte II

Progettazione dell’emulatore dell’architettura TCG

Capitolo 4

Architettura dell'emulatore

Il capitolo dopo un'introduzione alle motivazioni che hanno spinto alla progettazione di un'emulatore di Trusted Platform, secondo la specifica TCG, introduce il concetto di ambiente virtuale e ambiente reale che rappresentano l'architettura generale dell'emulatore.

4.1 Motivazioni

Le motivazioni che sono alla base del lavoro di progettazione dell'emulatore di un'architettura hardware, sono sia legate a vantaggi generici portati dalla realizzazione di emulatori, che strettamente riferite a questo particolare lavoro. Tra questi vantaggi, è possibile annoverare:

- possibilità di un continuo adattamento alla specifica, senza esser vincolati alle politiche aziendali di rilascio specifiche e componenti hardware o software.
- aggiunta o rimozione di nuove funzionalità all'emulatore, proponendo e sperimentando situazioni e scenari difficilmente realizzabili via hardware. Le difficoltà di un approccio diretto in hardware sono dovute, ad esempio, ai costi e tempi di realizzazione, a volte troppo elevati, competenze specifiche, difficoltà di debug e difficoltà di interfacciamento con altre infrastrutture che mettono a disposizione API robuste e ben documentate.

- possibilità di integrare nella TP emulata componenti hardware emulati e software che interagiscano con la piattaforma stessa tramite componenti TCG.
- possibilità di simulare più ambienti Trusted su un'unica architettura fisica, facilitando il processo di debug, se necessario.
- maggior possibilità di analisi di problemi di sicurezza dell'architettura, dal momento che ogni componente può essere "facilmente" modificato e divenire oggetto del processo di debug, e quindi maggior possibilità della soluzione di questi problemi, agendo direttamente sui componenti dell'emulatore.
- maggior comprensione dell'architettura proposta da TCG per chi sviluppa l'emulatore, dovendo obbligatoriamente considerare tutti i dettagli della specifica che possono venire, a volte, superficialmente letti, ma che rappresentano fondamenta importanti per il corretto funzionamento dell'architettura stessa.

4.2 Architettura generale

L'architettura dell'emulatore individua due ambienti fondamentali, quello reale e quello virtuale, illustrati in figura 4.1.

4.2.1 Ambiente reale

L'*ambiente reale* è rappresentato da una piattaforma di calcolo fisica, sulla quale viene eseguito un sistema operativo, *GNU/Linux*, e grazie alla quale è possibile progettare e realizzare in software i componenti hardware rappresentanti le root of trust, CRTM e TPM, e la Platform, componente software che simula rozzamente una scheda madre e che funge da tramite nella comunicazione tra i chip emulati, facenti parte di questo ambiente, e l'intero ambiente virtuale.

4.2.2 Ambiente virtuale

L'*ambiente virtuale* è un'ambiente che rappresenta una sorta di macchina virtuale dotata di un sistema operativo e di tutti i componenti software necessari al suo funzionamento. In questo ambiente è possibile individuare, tra gli altri, due importanti componenti, il TPM driver e l'I/O Manager.

Il *TPM driver* è il driver del chip TPM, scritto dai ricercatori dell'IBM, T. J. Watson Research Center. Questo driver si aspetta di trovare e di comunicare con un TPM fisico, reale e non uno emulato. A questo proposito entra in campo l'*I/O Manager*, che gioca un ruolo fondamentale nell'ambiente virtuale, che è necessario per permettere l'uso del driver TPM reale, presente all'interno di questo ambiente, nonostante i componenti hardware dell'architettura TCG non siano presenti fisicamente sulla piattaforma, ma siano stati emulati in software nell'ambiente reale. L'I/O Manager, infatti, si preoccupa di intercettare le istruzioni di basso livello che il driver usa per comunicare con il chip e dirottarle, attraverso la Platform, al TPM emulato nell'ambiente reale. L'infrastruttura che rende possibile tale comunicazione è descritta nella sezione 5.2.

4.3 User-Mode Linux

L'ambiente virtuale è stato realizzato tramite *User-Mode Linux* [17], un porting del kernel di Linux capace di esser eseguito come processo utente su di una macchina fisica che esegue un kernel Linux.

User-Mode Linux crea una sorta di macchina virtuale che può avere più hardware e software virtuale di quello realmente presente fisicamente. I dischi, ad esempio, per la macchina virtuale sono rappresentati e contenuti all'interno di un singolo file della macchina fisica. Così, il kernel di User-Mode Linux è stato modificato in modo tale da permettere l'inserimento del driver originale del TPM e la realizzazione dell'I/O Manager, componente fondamentale in questo ambiente virtuale, per offrire l'illusione, al driver reale, della comunicazione con un chip fisicamente presente nel sistema che in realtà è emulato.

La progettazione di questi ambienti non è solo legata all'architettura TCG e per-

tanto è possibile, volendo, aggiungere altri componenti hardware emulati in software nell'ambiente reale, introducendo, nell'ambiente virtuale, i relativi device driver originali, senza la necessità di modificare[1] in linea di principio, nessun'altro componente dell'emulatore. Questo è reso in parte possibile grazie alle interazioni presenti tra ambiente reale e virtuale, portate avanti da I/O Manager e Platform, come descritto nella sezione 5.2.

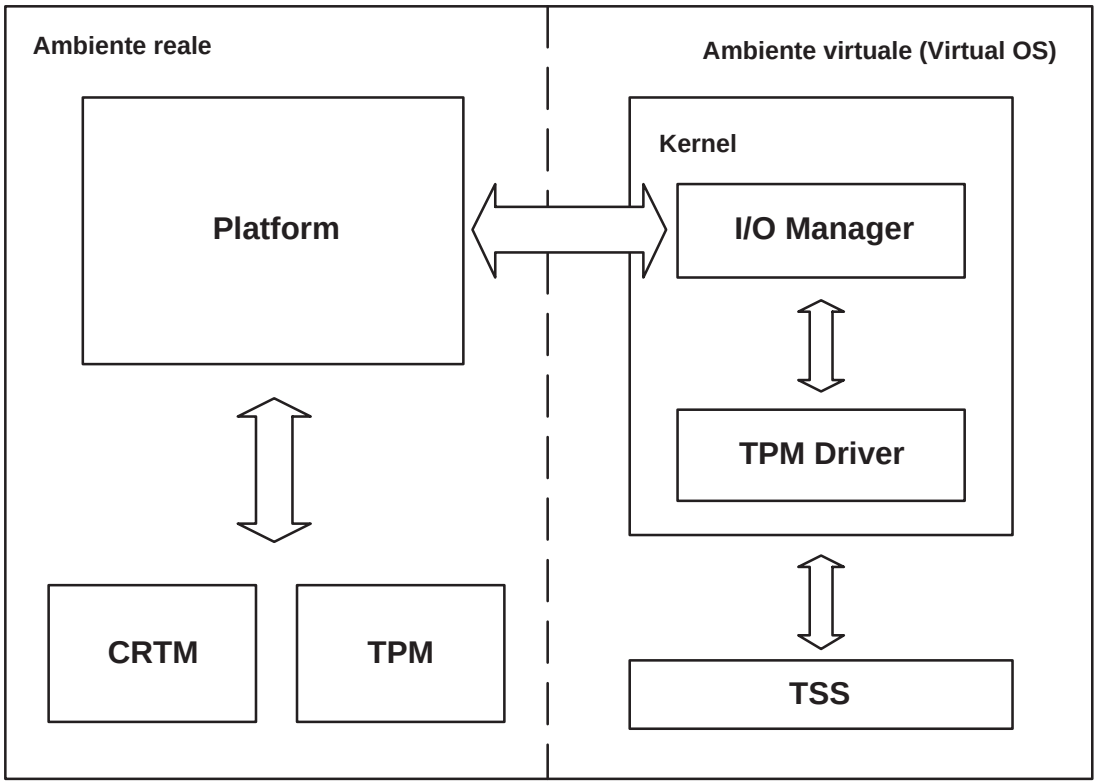

Figura 4.1: Emulatore infrastruttura TCG: ambiente reale e virtuale

[1] l'architettura progettata rappresenta comunque un prototipo.

Capitolo 5

Progettazione dell'emulatore

Il capitolo presenta la progettazione dell'emulatore software dell'architettura TCG. Il linguaggio di modellazione UML[1] *è stato usato per questo scopo data la sua ricchezza di strumenti adatti ad esprimere vari aspetti progettuali, quali i componenti principali e le interazioni temporali esistenti tra di essi.*

È possibile individuare due infrastrutture principali, presenti nell'architettura proposta: la prima, riferita all'ambiente pre-boot, si preoccupa di descrivere le componenti e le azioni svolte tra esse che iniziano con il boot del sistema ed arrivano fino al caricamento del kernel del sistema operativo, mentre la seconda, si preoccupa di delineare le interazioni esistenti tra le varie componenti al fine di rendere possibile la comunicazione tra l'ambiente pre-boot, reale, e quello post-boot, virtuale, rendendo di fatto possibile il dialogo tra sistema reale, che emula, in software, tra le altre cose il componente hardware del TPM, e quello virtuale, rappresentante il sistema operativo.

Il TPM, data la sua importanza e complessità, viene presentato nel capitolo 6.

5.1 Infrastruttura dell'ambiente pre-boot

Questo è l'ambiente in cui ci si trova una volta "accesa" la macchina: non esiste sistema operativo, non esistono applicazioni user space. L'unico processo che può aver

[1] Unified Modeling Language [19].

luogo in questa fase è quello di boot. Grazie alla collaborazione dei componenti coinvolti, è possibile procedere con la misurazione iniziale dell'ambiente[2] hardware e firmware della piattaforma che rende disponibile funzionalità di boot sicuro o autenticato. La sezione illustra la fase di inizializzazione dell'emulatore, così come le fasi relative alla misurazione dell'integrità dell'intero pre-boot, realizzato sulla piattaforma reale. Questo processo terminerà con l'esecuzione del kernel del sistema operativo virtuale che renderà, infine, possibile la comunicazione tra i due ambienti.

È importante ricordare che gli agenti di misurazioni sotto citati si limitano a *coordinare* il processo di misurazione, ma è sempre il TPM che esporta le funzionalità, adatte allo scopo, usate dagli agenti di misurazione.

5.1.1 Componenti

I componenti, illustrati in figura 5.1, che interagiscono in questa fase sono[3]:

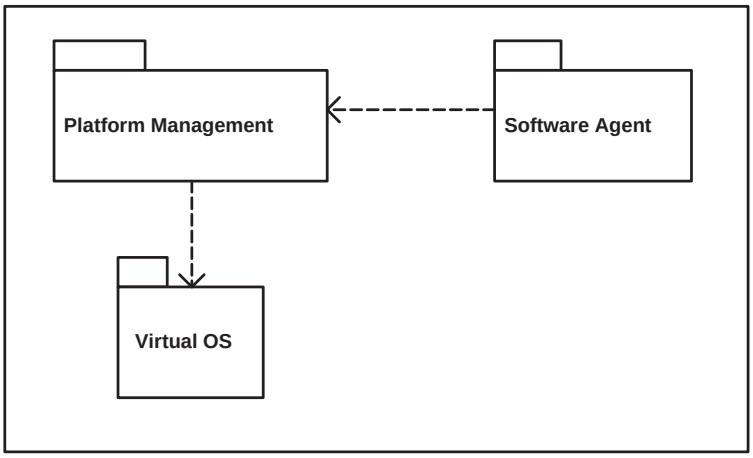

Figura 5.1: Relazioni di dipendenza tra i *package* dell'emulatore

[2]ambiente che comprende componenti, ovviamente, emulati in software.

[3]ogni componente rappresenta in realtà un *software package*, che racchiude al suo interno altri componenti correlati.

Platform Management. Questo pacchetto comprende i componenti che sono fondamentali per il funzionamento dell'emulatore; tra questi troviamo i chip emulati[4], la *Platform*, che rappresenta nel nostro caso, come in una piattaforma reale, il punto di comunicazione tra chip (emulati) e sistema operativo (virtuale), e altri componenti utilizzati per memorizzare le misurazioni effettuate e impiegati nel processo di comunicazione tra l'ambiente reale (pre-boot) e quello virtuale (post-boot). La figura 5.2 mostra le relazioni esistenti tra i componenti contenuti in questo pacchetto.

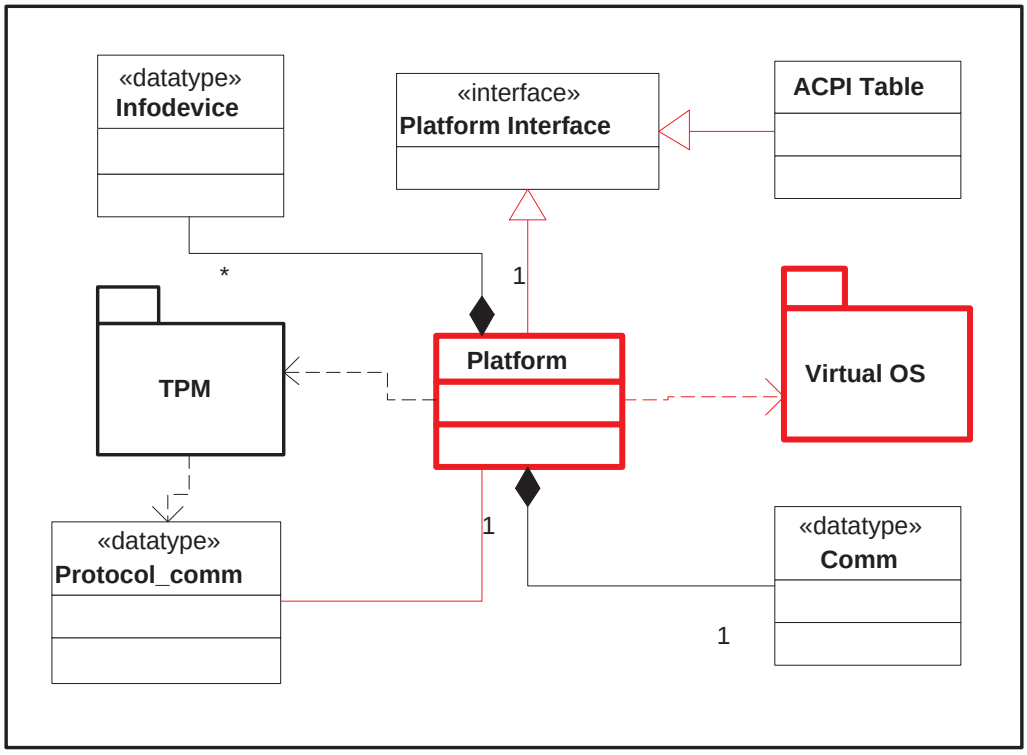

Figura 5.2: Diagramma delle classi del *package* Platform

Software Agent. Il pacchetto comprende gli agenti di misurazione previsti dalla specifica per questa fase. Tra questi possiamo annoverare la root of trust CRTM,

[4]il TPM è visto come pacchetto anch'esso, data la sua complessità.

il POST BIOS, l'Initial Program Loader (IPL), responsabile in un sistema reale della transizione dallo stato di pre-boot a quello di post-boot che avviene nel momento in cui si carica il kernel del sistema operativo, così come i driver che rendono possibile il processo di inizializzazione del TPM e la misurazione dell'ambiente pre-boot. La figura 5.3 mostra le relazioni esistenti tra i componenti contenuti in questo pacchetto.

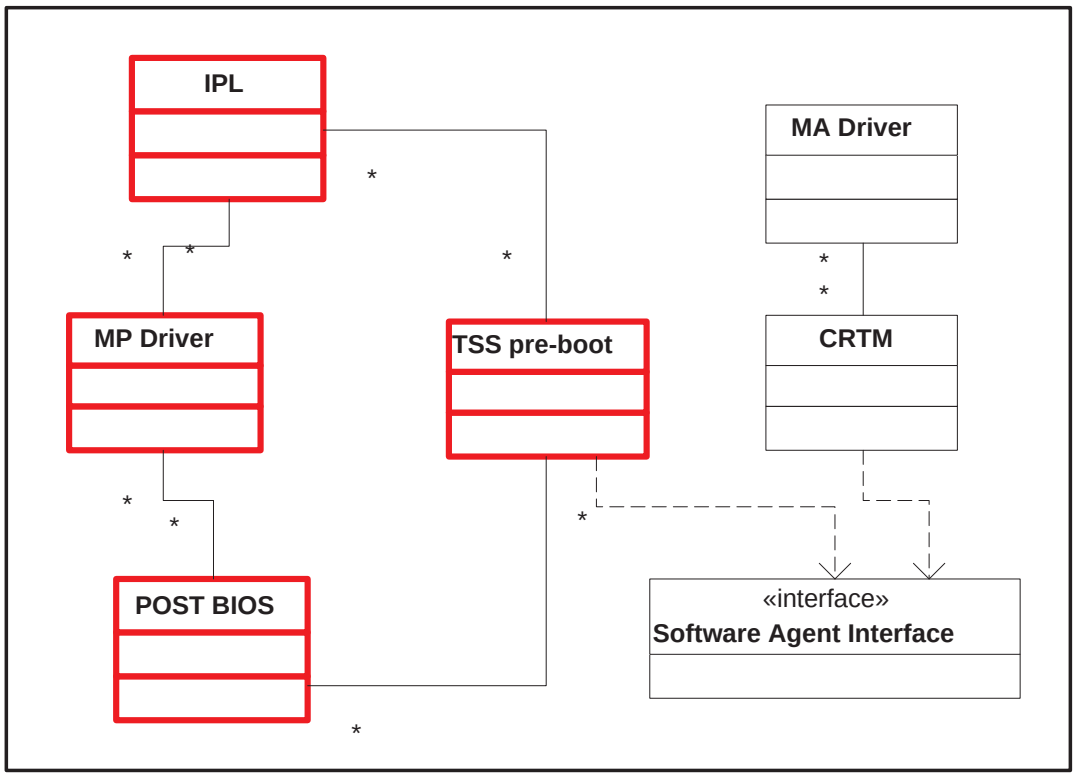

Figura 5.3: Diagramma delle classi del *package* Software Agent

Virtual OS . Il pacchetto rappresenta tutto ciò che non è pre-boot. Rappresenta tutto l'ambiente post-boot, il sistema operativo e quindi l'intero ambiente virtuale.

La fase di inizializzazione dell'emulatore software e quindi dell'intero sistema e la creazione del processo di misurazione che offre funzionalità di boot sicuro o autenticato, è meglio comprensibile analizzando le sequenze di operazioni che intercorrono

tra i componenti sopra elencati. Dopo una prima fase di inizializzazione tecnica dell'emulatore, inizia il processo di misurazione dell'integrità dell'ambiente pre-boot che si conclude con la misurazione e l'esecuzione[5] della macchina virtuale rappresentata dal pacchetto Virtual OS.

5.1.2 Inizializzazione del sistema

Lo schema di inizializzazione del sistema è illustrato nella figura 5.4 che rappresenta, uno schema semplificato del diagramma di sequenza UML, realizzato in fase di progettazione[6].

Questa è la fase iniziale che prevede la lettura di un file di configurazione dell'emulatore che contiene, tra le altre, informazioni sui chip da emulare, le porte di I/O usate per per comunicare con essi, informazioni riguardo i plugin[7] che implementano i chip e il tipo di boot desiderato, sicuro o autenticato.

Successivamente vengono creati i canali utilizzati per la comunicazione tra la piattaforma e i chip e tra la piattaforma e la macchina virtuale; in pratica la piattaforma si preoccupa di smistare i messaggi provenienti dal sistema operativo virtuale indirizzati ai chip e viceversa.

La fase di inizializzazione si conclude con la creazione del *thread* rappresentante il TPM, che eseguirà il suo processo di inizializzazione, delineato nel capitolo 6, e con quello rappresentante il CRTM, le due root of trust previste dalla specifica TCG. L'esecuzione viene passata al CRTM, assiomaticamente fidato che inizierà la misurazione dell'ambiente pre-boot.

---

[5] solo nel caso di boot autenticato.

[6] a causa della complessità dei diagrammi delle classi e di sequenza realizzata per la progettazione dell'emulatore, il lavoro di tesi riporta solo versioni *semplificate*. Anche gli appendici non sono in grado di rappresentare in modo adeguato tali diagrammi.

[7] tecnicamente è un ELF ET_DYN.

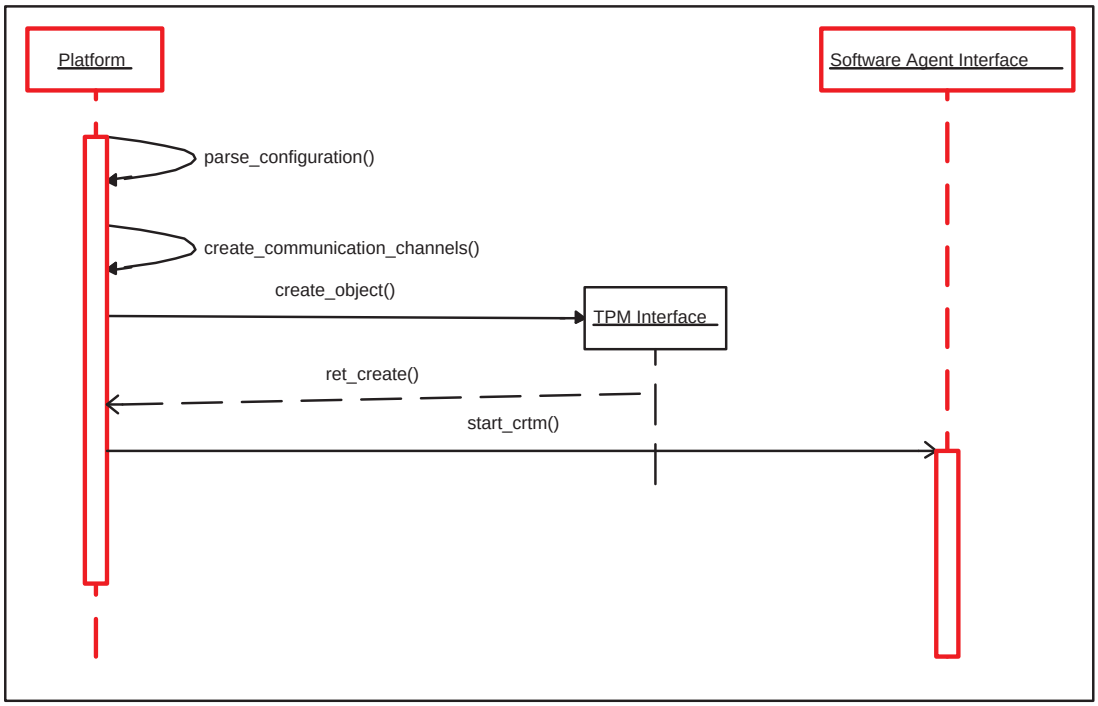

Figura 5.4: Inizializzazione dell'emulatore

5.1.3 Misurazioni del CRTM

La figura 5.5 illustra, in modo semplificato, il processo di misurazione effettuato dal componente e descritto qui di seguito.

Prima di iniziare la fase di misurazione dell'integrità del sistema, il CRTM invia un comando al TPM, `TPM_Startup(TCPA_ST_CLEAR)`, provocandone il reset, il caricamento di alcuni dati peristenti dalla memoria non volatile a quella volatile e l'impostazione della sua modalità operativa (sezione 6.2 e capitolo 6 in generale).

La seconda operazione svolta dal CRTM è quella di misurare se stesso; in realtà viene registrato, sempre tramite funzionalità offerte dal TPM, nei PCR il numero di versione del CRTM, più a scopo informativo che di misurazione vera e propria: il CRTM è una root of trust assiomaticamente fidata, grazie al processo di fiducia (sociale) coinvolto nell'attestazione della TP e non necessita, pertanto, di garantire in modo

ulteriore la sua genuinità.

Il CRTM prosegue il suo compito misurando l'integrità del codice[8] del POST BIOS e, dopo averne annotato il risultato nel TPM, crea il POST BIOS *thread* al quale viene ceduto il controllo e inizia il suo processo di misurazione.

5.1.4 Misurazioni del POST BIOS

Il POST BIOS thread, dopo aver controllato la presenza dell'interfaccia BIOS conforme alla specifica TCG, ottiene un accesso diretto[9] al TPM disabilitando la possibilità di eliminare, se presente, il TPM Owner sia tramite l'uso di un comando autorizzato, che tramite l'uso di un comando che richiede presenza fisica (sezione 6.3), come illustrato, in modo semplificato, dalla figura 5.6.

Il POST BIOS prosegue la sua attività misurando l'integrità delle seguenti componenti:

- tutti i firmware disponibili. L'attività consiste, oltre alla misurazione, nel registrare il digest calcolato all'interno del `PCR0`[10] e nel registrare un sommario più dettagliato nel TPMS rappresentato da una parte di tabella ACPI[11] nell'ambiente pre-boot.

- microcode della CPU, se supportato. Anche in questo caso, dopo aver eseguito la misurazione, viene memorizzato all'interno del `PCR1` il digest calcolato e un sommario più dettagliato nella tabella ACPI.

- tutte le ROM opzionali presenti sulla piattaforma. Come nei casi precendenti, il digest viene memorizzato all'interno del TPM, nel `PCR2`, e il sommario nella tabella ACPI.

[8] analogia al firmware di un POST BIOS reale.

[9] attraverso l'utilizzo di un driver particolare, il *Memory Present Driver (MP Driver), che esegue le operazioni sul TPM*.

[10] il primo PCR del TPM; la specifica v1.1 ne prevede 16, numerati da 0 a 15.

[11] anche questo componente hardware viene emulato, ma solo nella sua funzione di repository software.

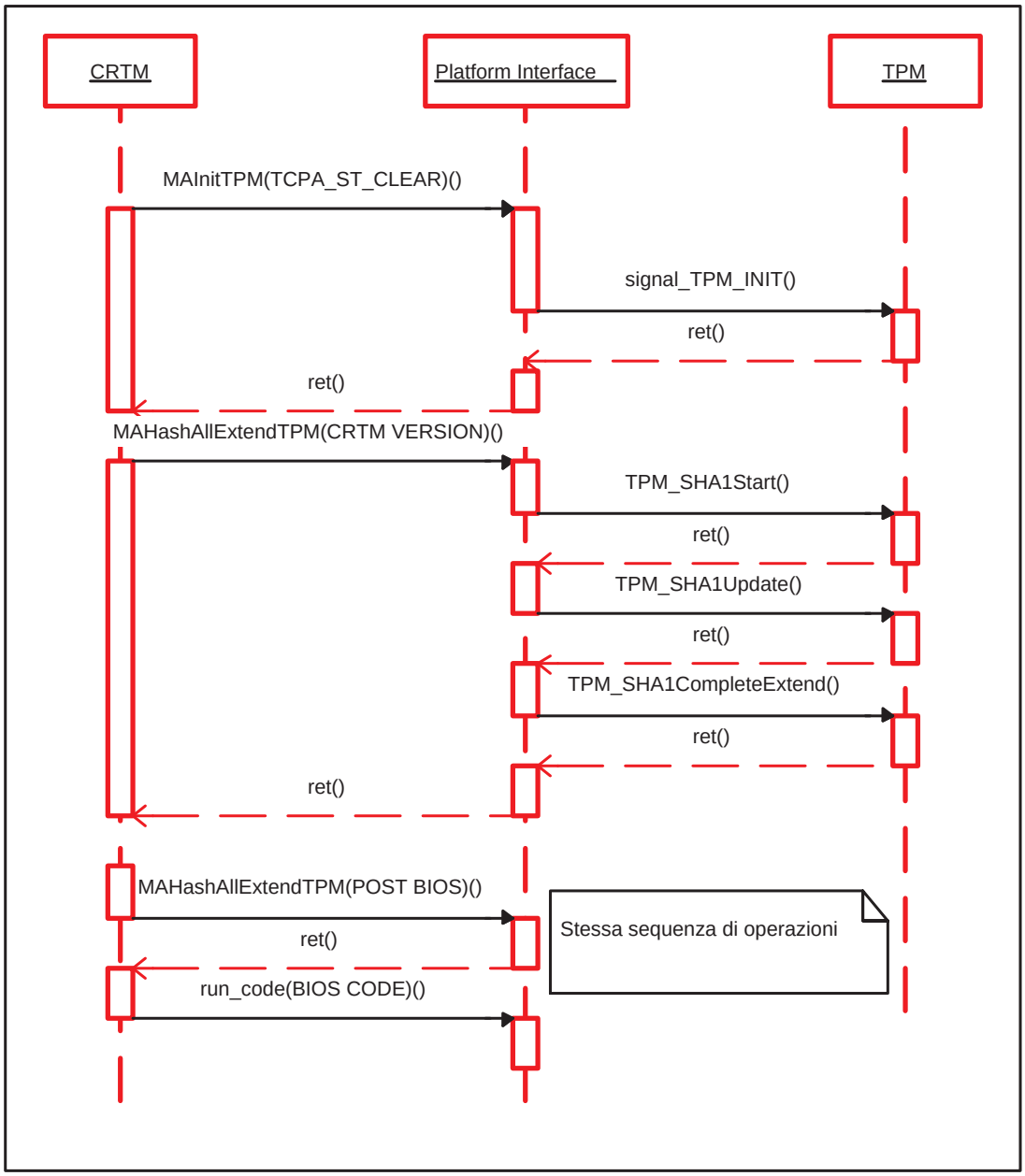

Figura 5.5: Misurazioni d'integrità effettuate dal CRTM

- tutti i dati delle ROM opzionali. Il digest è memorizzato nel PCR3 e il sommario dettagliato nella tabella ACPI.

L'attività di misurazione eseguita dal POST BIOS termina con il calcolo del digest relativo al codice dell'*Initial Program Loader, IPL*, memorizzato nel PCR2 e nella tabella ACPI e con l'annotazione, nel PCR4 e nel TPMS, del passaggio di stato, da *pre-boot* a *post-boot*. Il codice IPL infatti è responsabile del caricamento del kernel del sistema operativo e della sua misurazione, nonché della transizione dello stato di esecuzione della piattaforma. Il POST BIOS, dopo aver misurato e memorizzato tale codice, crea l'IPL *thread* che continuerà l'esecuzione e il processo di misurazione dell'integrità del sistema.

5.1.5 Misurazioni dell'IPL

Questo agente di misurazione recupera il codice del *Master Boot Record (MBR)*, lo misura e ne registra il risultato nel PCR5 insieme ad un sommario dettagliato nel TPMS. Successivamente viene recuperato il codice del kernel del sistema operativo, viene misurato e il digest è memorizzato anch'esso nello stesso PCR usato precedentemente, con un sommario dettagliato, come al solito, nel TPMS. La figura 5.7, cerca di illustrare, in modo semplificato, il processo di misurazione sopra accennato e qui di seguito continuato.

A questo punto è necessario determinare il tipo di boot scelto tramite il file di configurazione analizzato nella fase di inizializzazione del sistema. Se il boot specificato è quello autenticato, allora l'esecuzione viene ceduta al kernel misurato precedentemente e si completa la transizione di stato, entrando nell'ambiente post-boot, quello virtuale. Se il boot scelto è invece di tipo sicuro, l'agente di misurazione recupera tutti i valori dei PCR, li memorizza in una lista e ne calcola un digest. Questo digest viene confrontato con il registro DIR presente nel TPM e, se i due valori coincidono, si possono considerare fidate le componenti misurate nei passi precedenti e il processo di boot può completarsi con il caricamento e l'esecuzione del kernel. Nel caso in cui i due digest non coincidano significa che qualche componente è stato manomesso e

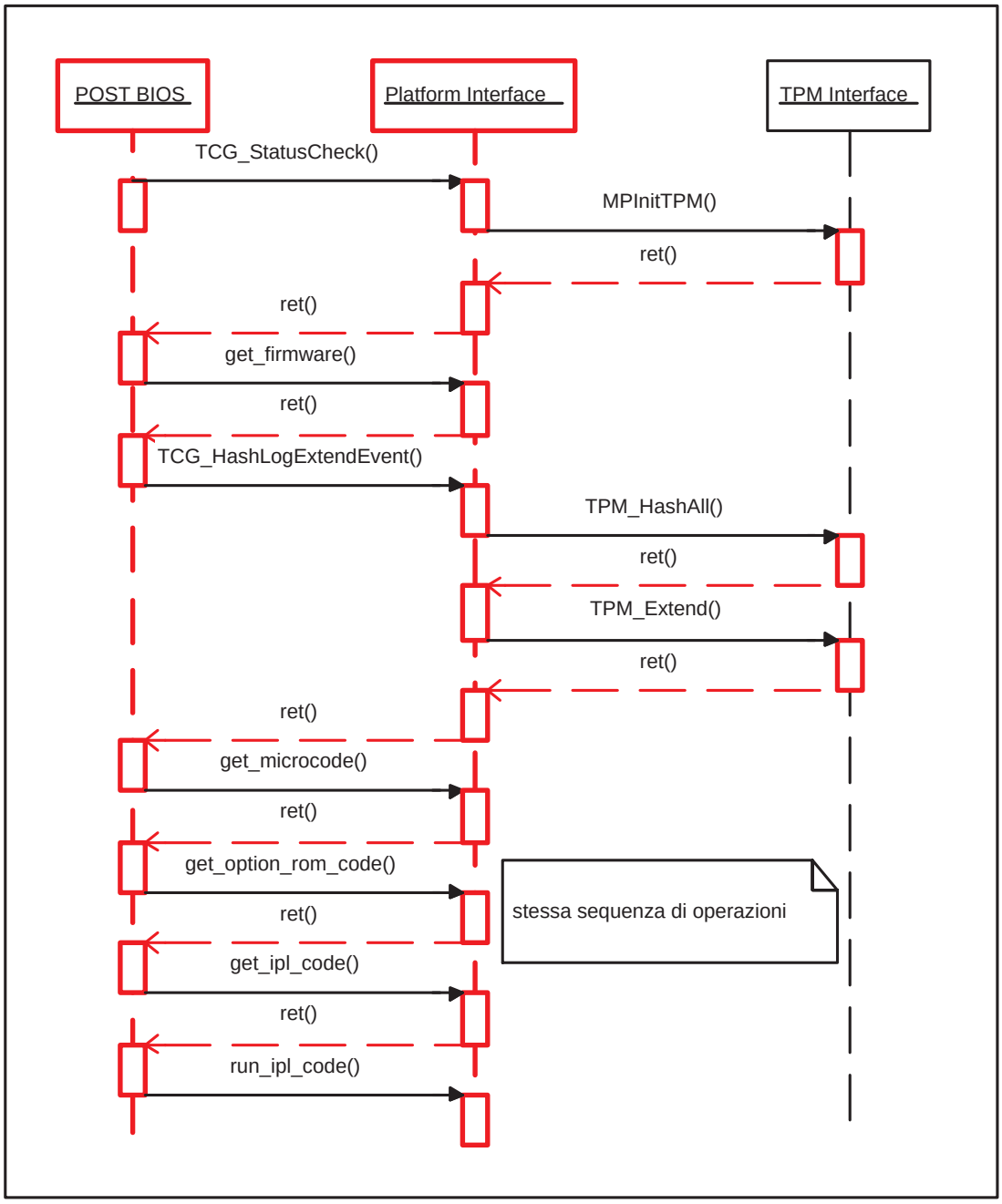

Figura 5.6: Misurazioni d'integrità effettuate dal POST BIOS

quindi lo stato della piattaforma non può essere considerato fidato; per questo motivo il processo di boot viene interrotto e terminato.

5.2 Comunicazione tra ambiente virtuale e reale

Una volta eseguito il boot del sistema, l'ambiente che si presenta coinvolge principalmente il TPM e il sistema operativo caricato. In realtà è possibile individuare due ambienti, quello *virtuale* relativo al sistema operativo comprensivo di driver per il dispositivo TPM, e quello *reale* che ospita le root of trust progettate, TPM e CRTM.

Lo scopo di questa infrastruttura consiste nel mettere in comunicazione questi ambienti. A tal proposito verranno analizzati, oltre ai componenti coinvolti nella comunicazione, anche le strutture dati, i protocolli e i canali di comunicazione, evidenziando il flusso di esecuzione seguito per l'esecuzione di un comando del TPM, da parte di un'applicazione user space presente nel sistema operativo virtuale.

5.2.1 Componenti

In questa fase progettuale è possibile individuare i componenti principali dell'emulatore, già illustrati nella figura 4.1.

I componenti coinvolti appartengono ai due ambienti sopra introdotti, quello reale e quello virtuale e l'appartenenza ad un'ambiente o all'altro è esplicitata nell'elenco sottostante:

Platform. È il *thread* principale dal quale ha inizio tutto l'emulatore; come già notato precedentemente, è responsabile per la comunicazione tra l'ambiente virtuale e i chip emulati nell'ambiente reale di cui ne fa parte. Vista la sua natura di arbitro nel processo di comunicazione, ne deve conoscere le strutture dati e i protocolli usati.

Virtual OS. È il *thread* rappresentante tutto l'ambiente virtuale; comprende il sistema operativo e le applicazioni user space che, tra le altre, sfruttando l'architettura Trusted. La comunicazione tra il Virtual OS, ambiente virtuale, e il chip del

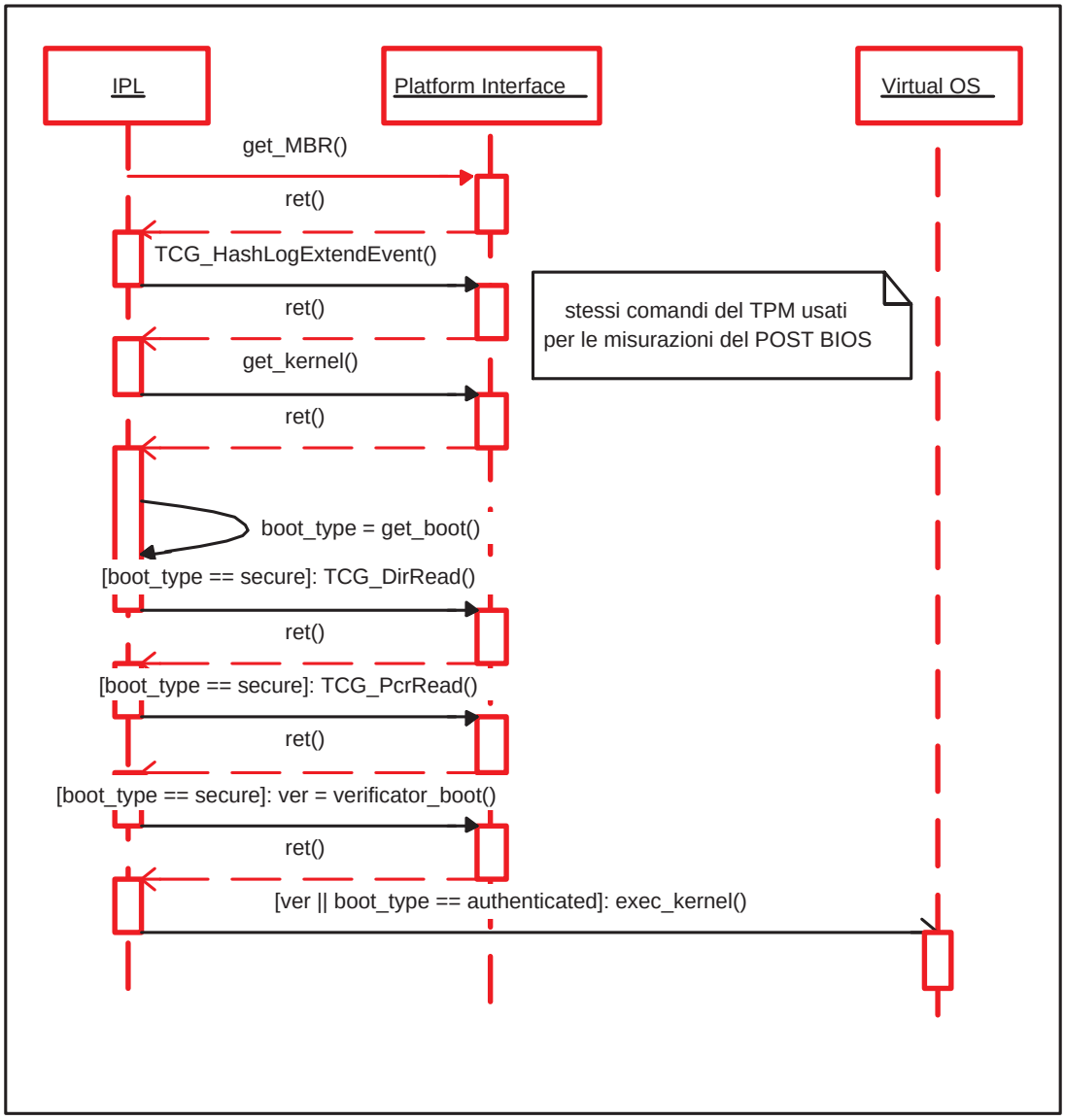

Figura 5.7: Misurazioni d'integrità effettuate dall'IPL

TPM, ambiente reale, è effettuata tramite un componente appartenente al Virtual OS, l'I/O Manager, che gestisce la comunicazione con la piattaforma.

I/O Manager. Componente del Virtual OS responsabile della comunicazione tra ambiente virtuale e reale. A tale scopo, fornisce anche le "istruzioni" di basso livello, *in e out*, utilizzate generalmente dai device driver per comunicare con i device.

TPM Driver. È il driver utilizzato per gestire il device TPM, sviluppato dal gruppo di ricerca del T. J. Watson Research dell'IBM [2] e, vista la sua natura, fa parte dell'ambiente virtuale, compreso nel Virtual OS. Il driver comunica con il device TPM, nell'ambiente reale, tramite le usuali funzioni usate generalmente dai device driver, in e out, messe a disposizione dall'I/O Manager del sistema operativo virtuale modificato per questo scopo.

TSS. Rappresenta il componente software che definisce le librerie di supporto utilizzate dalle applicazioni conformi alla specifica TCG. La libreria definisce funzioni ad alto livello che si traducono nei rispettivi comandi TPM.

5.2.2 Operazioni assembly *in* e *out*

La comunicazione con le periferiche di I/O, come il TPM, avviene mediante scrittura e lettura di registri che fanno parte di tali dispositivi. In genere i registri dei device possono essere referenziati tramite due spazi di indirizzi, che prendono il nome di *memory mapped address space* e *I/O space*. Nel primo caso, i registri sono mappati in memoria e quindi sono raggiungibili attraverso il normale spazio di indirizzamento usato dall'*address bus* della piattaforma, mentre nel secondo caso, i registri delle periferiche di I/O sono referenziati utilizzando un altro spazio di indirizzamento. Il più delle volte un dispositivo ha più registri e questi sono referenziati, in entrambi gli spazi, usando indirizzi consecutivi.

Per l'hardware non c'è distinzione tra regioni di memoria e regioni di I/O; entrambe vengono referenziate asserendo segnali elettrici sul bus degli indirizzi e sul bus di controllo, segnali che identificano operazioni di *read* e di *write*.

Alcuni processori, però, tra cui il diffusissimo x86, hanno delle linee separate per asserire tali segnali nello spazio di indirizzamento I/O. Questo porta all'introduzione di istruzioni macchina, e relative mnemoniche assembly, dedicate a questo tipo di operazioni. L'istruzione *in* serve per leggere un dato da una porta di I/O rappresentante un registro di un device, mentre l'istruzione *out* rappresenta la corrispondente scrittura [25].

Il dispositivo TPM di riferimento[12] utilizza 4 porte di I/O per comunicare con l'ambiente esterno; ogni comunicazione prevede quindi l'utilizzo delle istruzioni in e out appena introdotte. Tuttavia, tali istruzioni non sono presenti nel componente rappresentato dal Virtual OS ma vengono utilizzate dal TPM driver originale per la comunicazione con il TPM. Il Virtual OS, infatti, è un ambiente virtuale nel quale tutti i meccanismi di gestione periferiche di I/O sono stati emulati, dall'autore di UML, in modo da non permettere l'uso di driver originali, relativi ai device emulati. Nella comunicazione, che avviene per passi, tra il TPM Driver e il chip TPM emulato, è stato necessario, pertanto, fornire tali istruzioni. Le istruzioni in e out sono fornite dal *nuovo* componente introdotto come modifica alla macchina virtuale, I/O Manager, che fa parte del Virtual OS. In questo modo il TPM driver può comunicare con il dispositivo TPM emulato in modo trasparente, senza accorgersi dell'emulazione in corso.

Per fornire questa trasparenza nella comunicazione, l'I/O Manager fornisce le operazioni *inb* e *outb*[13] così realizzate:

```
unsigned char
inb(unsigned short p)
{

   struct Protocol_comm comm;
   unsigned char info;

   comm.io_port = p;
```

[12] implementato dalla società *Atmel* [15].

[13] la *b* identifica che le operazioni operano su dati di dimensioni di 1 byte.

```
   comm.datasize = 0;
   comm.data = 0;

   if (tcpa_comm.type == TCPA_COMM_TYPE_FD) {

      user_write_fd( tcpa_comm.tcpa_comm_fd, \
         (char *) &comm, \
      sizeof(struct Protocol_comm));

      /*
       * lettura dal canale di comunicazione
       * tcpa_comm_fd
      */
      user_read_fd(tcpa_comm.tcpa_comm_fd, &info, \
         sizeof(char));
   }

   return info;
}

void
outb(unsigned char data, unsigned short port)
{

   struct Protocol_comm comm;

   comm.io_port = port;
   comm.datasize = 1;
   comm.data = data;

   /* fd descriptor */
   if (tcpa_comm.type == TCPA_COMM_TYPE_FD) {

      /*
       * lettura dal canale di comunicazione
       * tcpa_comm_fd
      */
```

```
        user_write_fd( tcpa_comm.tcpa_comm_fd, \
           (char *) &comm, \
           sizeof(struct Protocol_comm));

    }

}
```

Da questo piccolo estratto di codice del componente I/O Manager è possibile notare che le operazioni "assembly" inb e outb non comunicano più direttamente con l'hardware, ma si preoccupano di usare una struttura dati particolare, introdotta nella prossima sottosezione, al fine di comunicare con l'ambiente reale.

Da notare come è stata realizzata l'operazione di inb: la lettura del dato, proveniente dal TPM emulato nell'ambiente reale, e destinato al TPM driver che poi lo consegnerà alla relativa applicazione user space presente nell'ambiente virtuale, consiste nella segnalazione tramite scrittura al TPM che l'operazione che si vuole eseguire è una lettura che data la sua peculiarità, viene chiamata *lettura attiva*. Questa operazione permette di cambiare il comportamento predefinito – lettura da patre del TPM e scrittura da driver TPM – in lettura da parte del driver TPM e scrittura del TPM, per consentire il corretto svoglimento dell'istruzione "assembly" emulata in. Il dettaglio del protocollo di comunicazione che coinvolge diverse componenti dell'architettura, mostra il flusso di esecuzione di un comando eseguito dal TPM driver (ambiente virtuale), fino al TPM (ambiente reale), processo spiegato nella sottosezione 5.2.4.

5.2.3 Canali e protocollo di comunicazione

È possibile individuare diversi canali di comunicazione usati tra i componenti dell'architettura sopra descritti:

- La comunicazione tra Platform e Virtual OS viene effettuata tramite l'utilizzo di socket `AF_UNIX` che stabiliscono un *canale bidirezionale* tra i due componenti. In realtà il tipo di canale non è limitato all'uso di socket; è possibile infatti associare anche un semplice file o file descriptor per poter controllare, ad esempio,

la corretta comunicazione tra i due ambienti, in modo da semplificare eventuali processi di debug.

- L'I/O Manager rappresenta un componente dell'ambiente virtuale, pertanto la comunicazione tra di esso e il Virtual OS è inesistente; svolgono concettualmente compiti diversi ma sono "lo stesso oggetto".

- L'I/O Manager fornisce le operazioni di basso livello, inb e outb, per permettere al TPM driver di comunicare con l'ambiente reale, rappresentato da Platform, in modo tale da raggiungere il proprio chip TPM.

- La comunicazione tra Platform e i chip emulati, che nel nostro caso sono rappresentati esclusivamente dal solo TPM[14], avviene, anche qui, mediante l'utilizzo di un canale bidirezionale, rappresentato il più delle volte da socket `AF_UNIX`.

- Rappresentando soltanto un componente software dell'ambiente virtuale, il TSS funge da "wrapper" per i comandi che verranno inviati al TPM, nell'ambiente reale, mediante il TPM driver.

Al fine di permettere la corretta comunicazione tra i due *end-point* principali individuati nell'architettura, il TPM driver facente parte dell'ambiente virtuale e il chip del TPM dell'ambiente reale, il protocollo scelto deve:

1. fornire l'informazione sul tipo di operazione a basso livello destinata al TPM, eseguita in ultimo dal TPM driver,

2. fornire la corretta informazione per permettere alla Platform di poter individuare il chip al quale destinare tale operazione. L'unico chip emulato nell'architettura è il TPM, ma la progettazione non fa nessun tipo di assunzione a riguardo, rimanendo il più generica possibile.

Per questo proposito, la struttura dati usata dal protocollo di comunicazione tra I/O Manager $\leftrightarrow$ Platform e tra Platform $\leftrightarrow$ TPM è rappresentata dalla figura 5.8

[14] la root of trust CRTM gioca un ruolo fondamentale, soltanto nell'ambiente pre-boot.

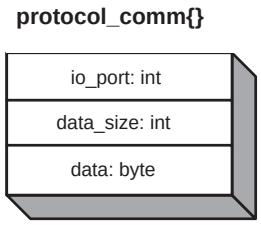

Figura 5.8: Struttura dati protocol_comm

Il campo `io_port` indica la porta di I/O sulla quale deve essere effettuata l'operazione di outb o inb. Il campo `data_size` indica la dimensione del dato che segue, specificato da `data` che, nel nostro caso, è di 1 byte se definisce l'operazione di outb, o 0 byte se indica l'operazione di inb (lettura attiva).

Un'altra struttura dati, fondamentale per la comunicazione tra Platform $\leftrightarrow$ TPM, è rappresentata da un vettore di interi di 65536 elementi. Tale vettore rappresenta tutto lo spazio di porte I/O dell'architettura[15] e gli elementi dell'array vengono riempiti opportunamente con il descrittore del canale di comunicazione creato nella fase di inizializzazione del sistema (sottosezione 5.1.2) riferito al chip emulato. Ad esempio, il chip reale TPM, secondo implementazione Atmel utilizza le porte di I/O `0x4E`, `0x4F`, `0x400` e `0x401`. Per poter permettere alla Platform di inoltrare le operazioni, rappresentate dalla struttura dati in figura 5.8 e ricevute dall'ambiente virtuale al TPM, il vettore sopra citato conterrà nelle locazioni corrispondenti alle porte di I/O elencate[16], il descrittore del canale di comunicazione bidirezionale.

5.2.4 Flusso di esecuzione di un comando

Diversi componenti interagiscono per poter eseguire un comando sul TPM emulato. Il seguente elenco evidenzia i passi che rendono possibile la comunicazione e quindi il

[15] Intel IA-32.

[16] in realtà le locazioni si riferiscono alle porte di I/O elencate -1 in quanto gli array in C partono da 0.

flusso di esecuzione di un comando.

1. un'applicazione user space facente parte dell'ambiente virtuale, utilizza una funzione di libreria messa a disposizione dal TSS. Tale funzione comunica con il TPM driver attraverso gli usuali meccanismi offerti dal sistema operativo.

2. il TPM driver traduce il comando ricevuto dalla funzione del TSS nelle corrispondenti sequenze outb/inb necessarie per comunicare con il dispositivo. Le funzioni outb e inb utilizzate, non si espandono nelle corrispondenti funzioni assembly out e in, ma eseguono il protocollo di comunicazione spiegato nella sottosezione precedente.

3. l'I/O Manager, che ha fornito le operazioni outb e inb, esegue il protocollo di comunicazione per comunicare le corrispondenti informazioni alla Platform. A tal proposito, la struttura `Protocol_comm`, illustrata in figura 5.8, viene riempita in modo opportuno, al fine di trasmettere, sul canale bidirezionale, tutte le informazioni necessarie alla Platform per poter determinare la porta di I/O destinataria dell'operazione e il tipo di operazione, out o in.

4. la Platform, ricevuto il pacchetto di tipo `Protocol_comm` dal canale bidirezionale instaurato con l'I/O Manager nell'ambiente virtuale, è in grado di determinare il tipo di operazione voluta, inb o outb e la porta di I/O destinataria dell'operazione che permette, esaminando il vettore descritto precedentemente, di determinare il chip destinatario dell'operazione.

   Se l'operazione determinata analizzando il pacchetto ricevuto è outb allora la Platform si limita ad eseguire una sua outb, diversa da quella fornita dall'I/O Manager[17], che trasferirà i dati al chip desiderato.

   Se l'operazione determinata è invece di tipo inb, allora la Platform istruirà in modo opportuno il chip emulato, il TPM, tramite una lettura attiva e si mettera,

[17] anche perché i contesti operativi sono diversi: la Platform è nell'ambiente reale, l'I/O Manager in quello virtuale.

quindi, in attesa di ricevere dati. In questo caso, verranno ripercorsi i passi a ritroso per poter trasferire i dati letti dall'ambiente reale nel quale è implementato il chip emulato, a quello virtuale, per arrivare tramite l'I/O Manager, fino al TSS e quindi all'applicazione user space che ha iniziato il comando.

Capitolo 6

Progettazione del TPM

Il capitolo che segue rappresenta forse il capitolo più importante della parte di progettazione svolta. Viene presentata la progettazione della root of trust TPM e vengono descritte alcune caratteristiche funzionali che sono state tralasciate fin'ora, quali le modalità operative.

Dopo una breve rassegna delle componenti architetturali di cui il TPM è dotato, vengono descritti i componenti principali che hanno permesso la progettazione del TPM emulato. Il capitolo si chiude con la descrizione dei passi necessari per inizializzare il dispositivo emulato e l'interazione che corre tra i vari componenti progettuali al fine di permettere la corretta esecuzione di un comando destinato al TPM.

6.1 Componenti architetturali del TPM

Il TPM è sostanzialmente un chip composto da più componenti che collaborano al fine di offrire le caratteristiche tipiche di una TP, come descritto nei capitoli 1, 2 e 3. Tra i suoi componenti architetturali, possiamo individuare:

I/O, che gestisce il flusso delle informazioni sul bus di comunicazione[1]. Esegue la codifica/decodifica del protocollo usato per comunicare sul bus interno ed esterno

[1]che nel caso reale, tipico di un'architettura ia32, è rappresentato dal bus LPC [3].

al fine di inoltrare i messaggi ai componenti di competenza. Questo componenta applica le politiche di accesso definite dalle modalità operazionali, descritte nella prossima sezione.

Co-processore crittografico, implementa le operazioni crittografiche nel TPM. Queste operazioni includono la generazione di chiavi asimmetriche RSA, cifratura/decifratura asimmetrica RSA, funzioni di hash SHA-1 e generazione di numeri pseudo casuali. Il TPM può usare la crittografia simmetrica per uso interno ma non espone all'esterno nessun algoritmo simmetrico.

Generatore di chiavi, responsabile per la creazione di coppie di chiavi asimmetriche RSA e di chiavi simmetriche.

Motore HMAC, usato dal TPM per fornire due tipi informazioni a se stesso: la prova della conoscenza dei dati di autorizzazione (vedere sezione 2.5 e capitolo 7) e l'integrità della richiesta autorizzata destinata al TPM. L'implementazione di questo componente segue lo standard RFC 2104 [23].

RNG, generatore di numeri casuali, è reponsabile della generazione di *nonce*[2] e chiavi crittografiche.

Opt-in, fornisce i meccanismi e le protezioni per permettere di accendere/spegnere, abilitare/disabilitare, attivare/disattivare il TPM. La modifica di alcuni flag richiedono l'autorizzazione dall'owner del TPM o l'asserzione della presenza fisica (vedere sezione 6.3).

Execution Engine. È il componente più importante che contiene il codice necessario per eseguire i comandi TPM ricevuti dalle porta I/O e decodificati dal modulo I/O.

Memoria Volatile e Non-Volatile. Oltre a contenere particolari strutture dati usate in fase di inizializzazione e durante la normale esecuzione dei comandi, autorizzati e non, sono le custodi dei PCR e dei DIR, le shielded location del TPM.

[2] numero pseudo casuale di 160 bit.

6.2 Modalità operazionali

Introduciamo di seguito le tre modalità operazionali tipiche di un TPM che definiscono la modalità del funzionamento del TPM e le caratteristiche che esso può offrire. Le tre modalità, *enable*, *activate* e *ownership*, sono identificate dalla presenza di tre stati discreti che, combinati opportunamente, offrono diverse modalità d'accesso.

Le tre modalità operazionali e i 2 meccanismi di autorizzazione, normale e tramite asserzione di presenza fisica, concetti che verranno ripresi o introdotti successivamente, contribuiscono a fornire una varietà di *controllo degli accessi* del TPM.

I tre stati discreti sono rappresentati da tre flag memorizzati nella memoria Volatile o NV del TPM.

Si distinguono due casi principali, TPM senza owner e con owner.

TPM senza owner. Il flag enable impostato a off disabilita la possibilità di installare un owner per via remota. Questo flag può essere modificato tramite l'asserzione di un segnale hardware il che richiede presenza fisica alla TP.

Il flag ownership dovrebbe essere usato al posto di enable se non si vuole essere assolutamente sicuri di impedire installazioni di owner del TPM per via remota. Questo perché il controllo di ownership può essere cambiato usando i normali dispositivi del computer, come la tastiera

Il flag activate è usato con il flag ownership per prevenire un attacco attraverso il quale un attacker cerca di prendere l'ownership prima dell'owner reale; Se non ci fosse questo flag o non venisse usato, l'owner reale potrebbe impostare il flag ownership a *on*, ma del software maligno potrebbe prendere l'ownership del TPM eseguendo il processo di take ownership, delineato nella sottosezione 3.1.1, prima dell'owner vero; grazie al flag activate però l'owner reale, dopo aver impostato ownership a *on* e activate a *off*, esegue il processo di Take Ownership e imposta activate a *on* solo *dopo* aver verificato che è effettivamente l'owner del TPM.

TPM con owner. Il flag enable può essere usato da chiunque abbia *accesso fisico* alla TP e possa quindi impostare a on/off il TPM; il TPM owner può autorizzare

l'impostazione del flag, per via locale o remota. Non è possibile sovvertire il meccanismo di presenza fisica e utilizzare in via remota enable per accendere il TPM.

Il flag activate può essere usato, da chiunque abbia *accesso fisico*, per mettere a off il TPM per un *intero* ciclo di boot e può essere usato da *qualsiasi* software per mettere a off il TPM per il resto del ciclo di boot. Questo espone la specifica ad un noto problema di DoS[3], ma questo è il prezzo da pagare per poter usufruire di queste funzionalità.

Operare sul flag ownership in questo scenario non produce alcun effetto.

Di seguito vengono illustrate le proprietà dei flag sopra introdotte; nella fattispecie vengono elencate le zone del TPM coinvolte nella memorizzazione di questi flag e la tipologia di comandi, autorizzati, non o che richiedono presenza fisica, necessari per poterli modificare.

Enable. Flag Non-Volatile (NV) del TPM il cui valore può essere cambiato tramite comando autorizzato dal TPM owner o comando di fisica presenza. La modifica del flag persiste tra i vari cicli di boot.

Il TPM opera normalmente con enable impostato a on mentre se enable è pari a off, le vere funzionalità del TPM sono "bloccate"; solo la misurazione nei PCR viene fatta in modo da permettere al TPM di conoscere sempre lo stato della TP.

Activate. Flag Volatile del TPM il cui valore può essere impostato a off, in ogni momento da chiunque, al fine di disabilitare il TPM per il resto della sessione di funzionamento, fino cioè al prossimo boot.

Ownership. Flag NV del TPM il cui valore è cambiato con un comando di presenza fisica. Il cambiamento di questo flag persiste attraverso vari cicli di boot (a differenza di activate) e può essere cambiato in continuazione (on/off) tramite comando di presenza fisica.

[3]Denial of Service, attacco che mira a impedire l'erogazione di un servizio.

6.3 Presenza fisica

Come già accennato nella sezione 2.5, il TPM implementa comandi autorizzati per poter usare le protected capability che effettuano operazioni sensibili.

Il modo comune per fornire privilegi sufficienti per poter eseguire una capability consiste nello spedire comandi autorizzati crittograficamente al TPM, sia localmente che per via remota.

In alcuni casi è necessario dimostrare di avere questo privilegio senza usare autorizzazioni crittografiche perchè, ad esempio, non sono ancora presenti nel TPM (owner non presente o autorizzazione associata dimenticata) o perchè la fase in cui si trova la TP non permette di aver un'infrastruttura capace di gestire/creare queste autorizzazioni crittografiche (si pensi ad esempio all'ambiente pre-boot).

La *presenza fisica (physical presence)* è il metodo usato in queste situazioni. Questo fa nascere un problema: determinare l'assoluta presenza fisica è generalmente costoso e questo va contro i principi di progettazione stabiliti da TCG. La specifica in questo caso fa un compromesso e cerca di garantire la presenza fisica al meglio; generalmente si preferisce quindi interagire con del software, ma limitando tale interazione a fasi iniziali come ad esempio la fase di Power-ON Self Test (POST) eseguita, al momento dell'inizializzazione hardware della piattaforma, dal BIOS. Questo garantisce la presenza fisica perchè è necessario premere un tasto della tastiera per poter entrare nella configurazione del BIOS, ad esempio.

Solo un comando che serve per abilitare fisicamente il TPM rappresenta un'eccezione al compromesso sopra descritto; accendere il TPM, infatti, è un'operazione così sensibile dal punto di vista della privacy che la specifica TCG richiede piena protezione per questo comando. `TPM_PhysicalEnable` deve essere implementato in un modo tale da non permettere sovversioni via software. I costruttori di schede madri, ad esempio, potrebbero utilizzare un jumper al fine di discriminare l'abilitazione o meno del comando.

6.4 Componenti progettuali del TPM

Per permettere maggior flessibilità nello svolgimento delle proprie operazioni, la progettazione del TPM emulato ha individuato compiti specifici e questo si è tradotto nella realizzazione di sei principali componenti, il *TPM Main Coordinator*, l'*HW Setup*, l'*Execution Engine*, il *TPM Components* e l'*I/O*, seguiti da componenti che rappresentano la *Volatile Memory* e la *NV Memory*. Alcuni di queste componenti, tra cui l'Execution Engine, TPM Components, Volatile e NV Memory, data la loro complessità, rappresentano dei *package* contenenti altri elementi necessari per lo svolgimento delle funzioni per le quali sono stati creati. La figura 6.1 aiuta a comprendere meglio le relazioni che intercorrono tra i principali componenti sopra enunciati.

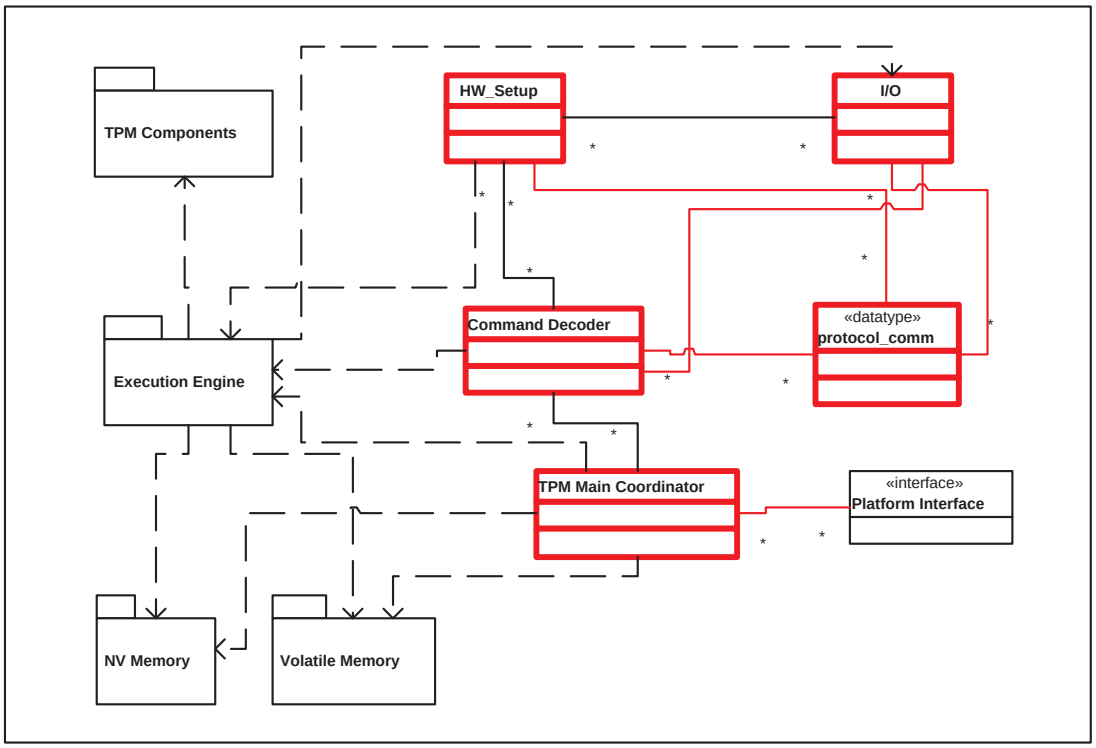

Figura 6.1: Diagramma delle classi del *package* TPM

6.4.1 TPM Main Coordinator

È il *thread* che viene creato dalla Platform in fase di inizializzazione del sistema, dell'emulatore. Il suo compito, come dice il nome, è quello di coordinare la creazione degli altri thread previsti nella progettazione del TPM, quali il Command Decoder, HW Setup e l'Execution Engine e di inizializzare le strutture dati previste dalla specifica TCG che permettono di gestire le modalità operazionali e il controllo degli accessi descritti nelle sezioni 6.2 e 6.3. Queste strutture dati, presenti nella memoria NV, vengono caricate nella memoria volatile al fine di garantire questo funzionamento.

Il TPM Main Coordinator è responsabile, oltre che della crezione dei thread sopra citati, anche del loro ripristino una volta che l'emulatore passa da uno stato di *sleep* ad uno di normale funzionamento.

6.4.2 Command Decoder

Questo *thread*, creato dal TPM Main Coordinator in fase di inizializzazione del TPM, è uno dei componenti software più importanti della progettazione.

Il suo compito è quello di determinare il tipo di comando che si vuole che il TPM esegua. Questo si traduce nel leggere, dal canale di comunicazione I/O, le informazioni necessarie a capire di che comando si tratta in modo da poter "avvisare" il componente di competenza, HW Setup o Execution Engine, a svolgere il proprio lavoro.

Vista la complessità di queste operazioni e i vari stati in cui il Command Decoder può entrare, è sembrato naturale modellare il suo funzionamento interno con un automa a stati finiti (Finite State Automata, FSA)[4].

Le figure 6.2 e 6.3 illustrano il funzionamento dell'automa nell'attività di riconoscimento delle due operazioni di basso livello, inb e outb.

Il Command Decoder, come già accennato, si preoccupa di determinare il tipo di operazione di basso livello necessaria per istruire successivamente i componenti software che gestiranno il comando decodificato. La determinazione del tipo di operazio-

[4] in realtà si tratta di un diagramma degli stati di UML che si differenzia dal classico FSA in quanto lo arricchisce fornendo caratteristiche quali azioni, guardie ed eventi.

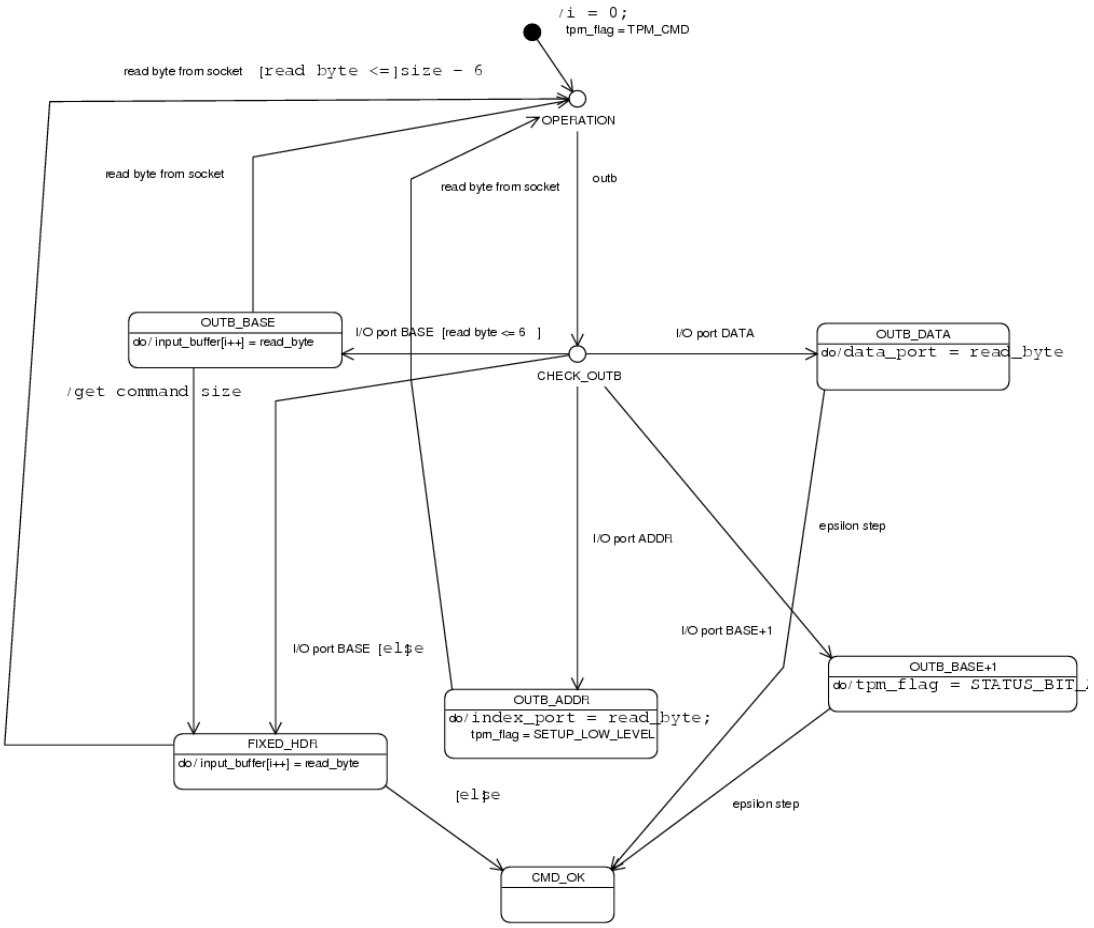

Figura 6.2: Finite State Automata (diagramma degli stati UML), operazione outb

ne, porta l'automa in nuovi stati di scelta, a seconda che l'istruzione sia inb (lettura) o outb (scrittura).

Se l'operazione determinata è quella di outb, allora tale istruzione può influenzare quattro porte di I/O, usate dal chip reale per ricevere comandi e quindi effettuare azioni. A seconda della porta di I/O selezionata, vengono svolti compiti differenti. Se la porta di I/O identificata è `BASE` che, nella specifica Atmel è `0x400`, allora il byte scritto su tale porta è parte del comando che il Command Decoder deve decodificare. Se la porta di I/O è `BASE+1`, pari a `0x401`, il byte scritto identifica un'azione di *abort*,

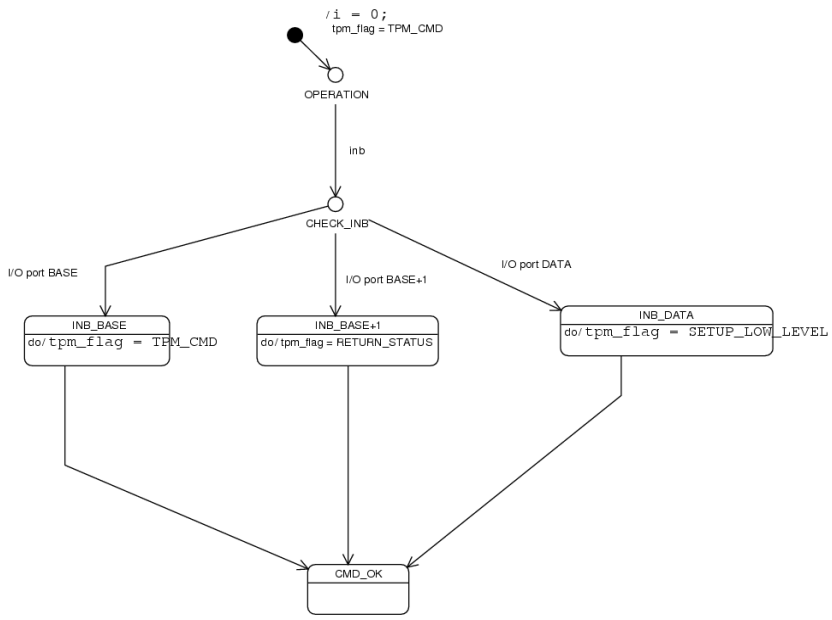

Figura 6.3: Finite State Automata (diagramma degli stati UML), operazione inb

causando la "terminazione"[5] dell'Execution Coordinator, nel caso fosse in esecuzione, per mano del componente HW Setup, avvisato dal Command Decoder. Le porte di I/O `ADDR`, pari a `0x4E`, e `DATA`, pari a `0x4F`, rappresentano rispettivamente le cosidette *porta index* e *porta data*. Sono porte usate per gestire alcuni aspetti funzionali di basso livello del TPM, che non hanno nulla a che vedere con l'esecuzione di comandi TPM veri e propri, comandi previsti dalla specifica TCG. Scrivendo un byte sulla *porta index*, si seleziona una funzione interna del chip TPM che dev'essere eseguita una volta che viene scritto un byte sulla *porta data*, che ne rappresenta anche l'argomento. Tali funzioni sono usate in fase di inizializzazione del dispositivo, dal TPM driver, e possono riguardare la disabilitazione degli interrupt, così come l'impostazione dei valori associati alle porte `BASE` e `BASE+1`.

Se l'operazione determinata è quella di inb, allora tale istruzione può influenzare

[5] in realtà non viene terminato in senso stretto; lo stato dell'Execution Coordinator e quindi del TPM passa da *busy* ad *idle*.

soltanto tre delle quattro porte di I/O usate dal chip reale. Se la porta di I/O identificata è BASE, significa che il comando inviato al TPM vuole leggere un byte, probabilmente di risposta ad un comando inviato precedentemente e per il quale il TPM, nelle vesti dell'Execution Engine, ha terminato la computazione[6], e il TPM quindi deve restituire tale valore. Se la porta di I/O è BASE+1, allora il TPM restituisce lo stato in cui si trova, *busy* o *idle* al chiamante, mentre se la porta di I/O è DATA, ci si riferisce, anche in questo caso, a operazioni di basso livello che non eseguono nessun comando TPM previsto dalla specifica; in questo modo è possibile recuperare, ad esempio, la versione del chip del TPM installato sulla TP.

6.4.3 HW Setup

Questo componente, rappresentato da un *thread* creato dal TPM Main Coordinator, gestisce tutti gli aspetti di basso livello legati all'emulazione software di un chip hardware. Viene risvegliato dal Command Decoder, ogni qualvolta il comando decodificato non riguarda l'esecuzione propria di un comando TPM. Tra i suoi compiti, ci sono quelli di cambiare lo stato del TPM, da busy ad idle, di restituirne il valore e di gestire tutti gli aspetti di basso livello, accennati nella sottosezione precedente.

6.4.4 Execution Engine

Questo componente, non certo meno importante del Command Decoder, data la sua complessità è composto da più elementi software che cooperano per poter offrire la funzionalità di un *motore di esecuzione* per i comandi del TPM. Possiamo individuare i seguenti componenti, illustrati in figura 6.4:

Execution Engine Coordinator. È il *thread* che viene creato dal TPM Main Coordinator nella fase di inizializzazione del TPM e si preoccupa di coordinare il processo di esecuzione dei comandi, determinando, tramite il componente Operational Mode Checker, se è possibile eseguire il comando decodificato dal Com-

[6] generalmente prima di ottenere un dato, il TPM viene interrogato sul suo stato; se è *idle* allora il risultato della computazione eseguita sarà coerente altrimenti no.

mand Decoder, comando previsto dalla specifica TCG. Se il comando può essere eseguito nella modalità operazionale corrente, allora l'Execution Engine Coordinator imposterà lo stato del TPM a *busy* e demanderà l'esecuzione del comando al Core Execution Engine. Una volta terminata l'esecuzione, è compito di questo thread ripristinare lo stato del TPM di nuovo ad *idle*.

Operational Mode Checker. Questo componente viene chiamato in causa dall'Execution Engine Coordinator e determina se il comando che si intende eseguire può essere eseguito nella modalità operazionale, descritta nella sezione 6.2, vigente in quel momento.

Core Execution Engine. È il componente rappresentante il cuore dell'esecuzione di un comando TPM. Grazie alle funzionalità implementate tramite *plugin*, descritti successivamente, è in grado di eseguire effettivamente il comando richiesto. Concettualmente, come l'Operational Mode Checker è integrato nel thread Execution Engine Coordinator.

Plug-in. Le funzionalità del TPM, i suoi comandi, vengono aggiunti e implementati mediante l'utilizzo di *plugin*. In questo modo è possibile demandare i dettagli dell'esecuzione di un comando esternamente all'infrastruttura di progettazione. I Plugin utilizzano un'API ben definita, fornita dall'infrastruttura in cui operano, al fine di poter interagire con essa ed eseguire il comando per il quale sono stati creati.

API (draft) Anche se in fase di "bozza", questo componente rappresenta l'interfaccia di programmazione messa a disposizione dall'infrastruttura ed usata dai *plugin* al fine di permettere l'interazione tra le due entità che portano all'esecuzione di comandi TPM.

6.4.5 TPM Components

Questo componente software racchiude al suo interno i componenti architetturali, coprocessore crittografico, motore SHA-1, motore HMAC, RNG, generatore di chiavi, previsti dalla specifica TCG per il TPM e descritti brevemente nella 6.1.

6.4.6 I/O

È il compomente responsabile della comunicazione tra TPM e Platform e componenti interni del TPM. In particolare, la comunicazione TPM ↔ Platform è possibile grazie all'utilizzo di un canale bidirezionale, descritto nella sottosezione 5.2.3, creato dalla Platform nella fase di inizializzazione dell'emulatore, mentre la comunicazione tra i componenti del TPM, parte dal TPM Main Coordinator e da lí viene demandata ai vari elementi chiamati in causa.

6.4.7 Volatile e NV Memory

Questi componenti rappresentano la memoria Volatile e Non-Volatile del TPM; tra le varie strutture dati memorizzate all'interno di queste aree, è possibile individuare i PCR e i DIR che, rappresentano, comunque, delle shielded location, accessibili soltanto tramite protected capability.

6.5 Inizializzazione del TPM

La fase di inizializzazione del TPM è semplice ed è illustrata nella figura 6.5. Il thread del TPM Main Coordinator viene creato in fase di inizializzazione dell'emulatore, dal componente Platform. Una volta creato, il TPM Main Coordinator, dopo aver inizializzato alcuni dati interni, si preoccupa di creare i thread del Command Decoder, HW Setup ed Execution Engine Coordinator e si mette in uno stato di *sleep*. Rimane attivo con il solo scopo di eseguire eventuali transizioni di stato dell'emulatore da *live* a *sleep* e viceversa.

6.6 Flusso di esecuzione all'interno del TPM

Il diagramma di sequenza che descrive le interazioni tra le componenti del TPM a fronte dell'esecuzione di un comando può essere diviso in due parti fondamentali: una relativa al flusso di esecuzione relativo ad un comando del TPM e l'altra relativa

alla gestione delle operazioni di basso livello legate al funzionamento fisico del TPM, accennate nelle sezioni precedenti.

6.6.1 Esecuzione di un comando TPM generico

I passi e le interazioni tra i componenti del TPM necessari per l'esecuzione di un comando possono essere sintetizzati nella tabella sottostante e illustrati dalle figure 6.6 e 6.7. Per semplificare la sequenza di operazioni coinvolte, il diagramma è stato diviso in due parti. La figura 6.6 si riferisce all'elenco seguente, comprensivo del punto 1a ma non del 1b, mentre la figura 6.7 si riferisce solamente ai punti 1 e 1b.

1. Il Command Decoder ha il compito, come precedentemente spiegato e descritto dall'automa a stati finiti delle figure 6.2 e 6.3, di decodificare il tipo di operazione richiesta e la porta di I/O coinvolta. Per far questo è necessario recuperare dal canale di I/O almeno un certo numero di byte[7]. Questi byte rappresentano le informazioni necessarie per determinare il tipo di comando che si vuole eseguire e quindi quanti byte rimangono da recuperare dal canale di I/O. Per quanto segue, è necessario dividere la trattazione a seconda che l'operazione determinata sia una outb, operazione di scrittura dati verso il TPM, o sia una inb, operazione di lettura dati dal TPM.

   (a) Se l'operazione decodificata è una outb, una volta recuperate tutte le infomazioni necessarie per rappresentare il comando che si vuole eseguire, il Command Decoder avvisa l'Execution Engine Coordinator e ritorna ad aspettare sul canale di I/O in attesa di altri dati.

   (b) Se l'operazione decodificata è una inb, una volta recuperate tutte le informazini necessarie per rappresentare il comando che si deve gestire, il Command Decoder, a seconda della porta coinvolta, interagisce con il canale di I/O restituendo il dato richiesto (porta di I/O BASE) o interagisce

[7] attualmente, nella v1.1 e v1.2 della specifica TCG, ammontano a sei; questi byte contengono anche l'informazione sulla lunghezza del pacchetto che descrive il comando da eseguire.

con il thread HW Setup, lasciandogli il compito della gestione del comando. In entrambi i casi, il Command Decoder si rimette in attesa di leggere dati dal canale di I/O. I punti rimanenti dell'elenco si riferiscono, pertanto, alla gestione di un comando pervenuto tramite operazione di scrittura verso il TPM, continuazione logica dei punti 1 e 1a.

2. L'Execution Engine Coordinator, dopo aver controllato che il comando ricevuto possa essere eseguito nella modalità operazionale corrente, attraverso l'ausilio del componente Operational Mode Checker, imposta lo stato del TPM a *busy* e sveglia il *thread* Core Execution Engine.

3. Il Core Execution Engine eseguirà il comando che coinvolgerà componenti architetturali del TPM, racchiusi nel componente software TPM Components e ritornerà al chiamante, l'Execution Engine Coordinator, una volta terminato il suo compito.

4. L'Execution Engine Coordinator cambia lo stato del TPM da *busy* ad *idle* e ritorna in attesa di essere svegliato nuovamente dal Command Decoder.

6.6.2 Esecuzione di un'operazione hardware

I passi e le interazioni tra i componenti del TPM necessari per l'esecuzione di un'operazione hardware legata al funzionamento fisico del TPM, condividono sicuramente il punto 1 dell'elenco descritto precedentemente. Le interazioni seguenti possono essere schematizzate nell'elenco sottostante e illustrate dalla figura 6.8:

1. dopo aver determinato il tipo di operazione e le porte di I/O coinvolte in questi comandi[8], il Command Decoder sveglia il thread HW Setup, ritornando poi in attesa sul canale di I/O.

2. l'HW Setup, venuto a conoscenza dell'operazione e della porta di I/O coinvolta, è in grado di eseguire il suo compito: restituire, scrivendo sul canale di

[8] outb su `BASE+1`, `ADDR` e `DATA`, inb su `BASE+1` e `DATA`.

I/O, lo stato del TPM o eseguire funzioni di basso livello, quali, ad esempio, disabilitazione degli interrupt per il TPM.

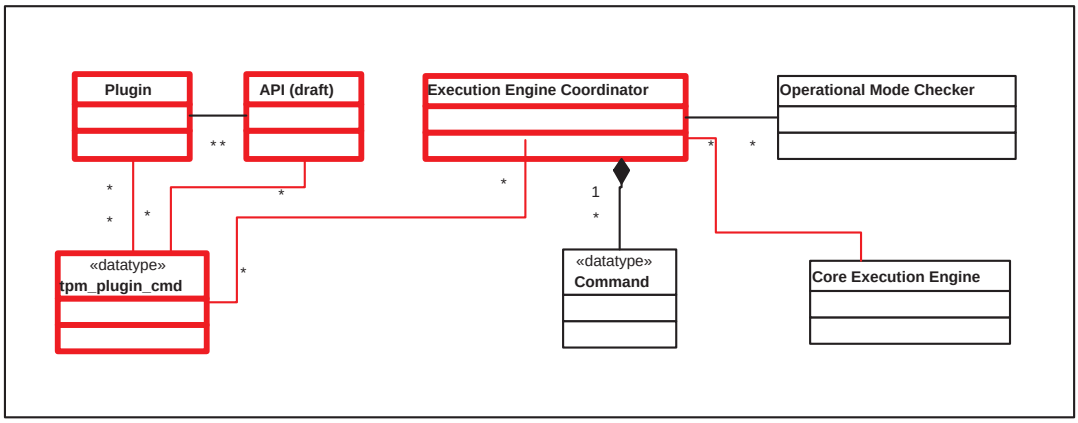

Figura 6.4: Diagramma delle classi del *package* Execution Engine

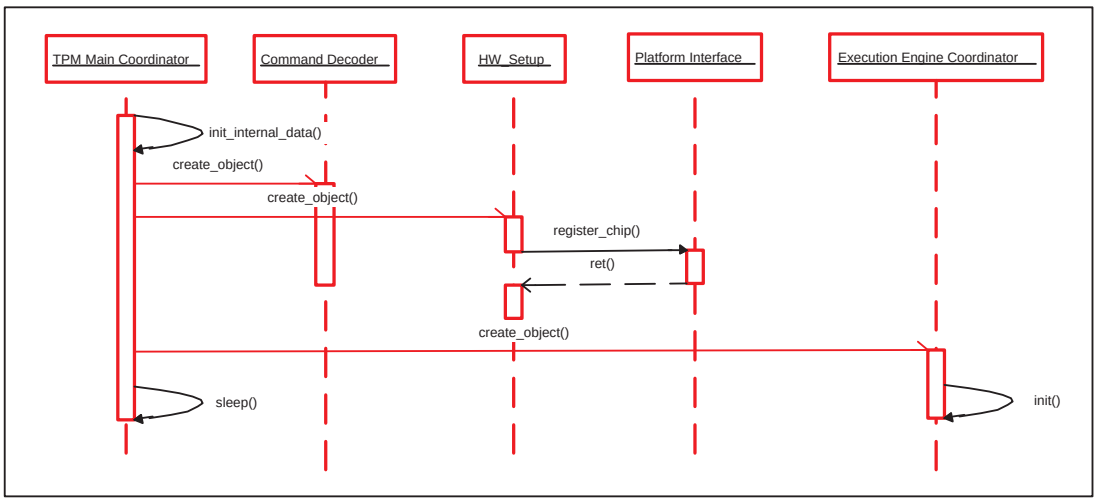

Figura 6.5: Inizializzazione del TPM

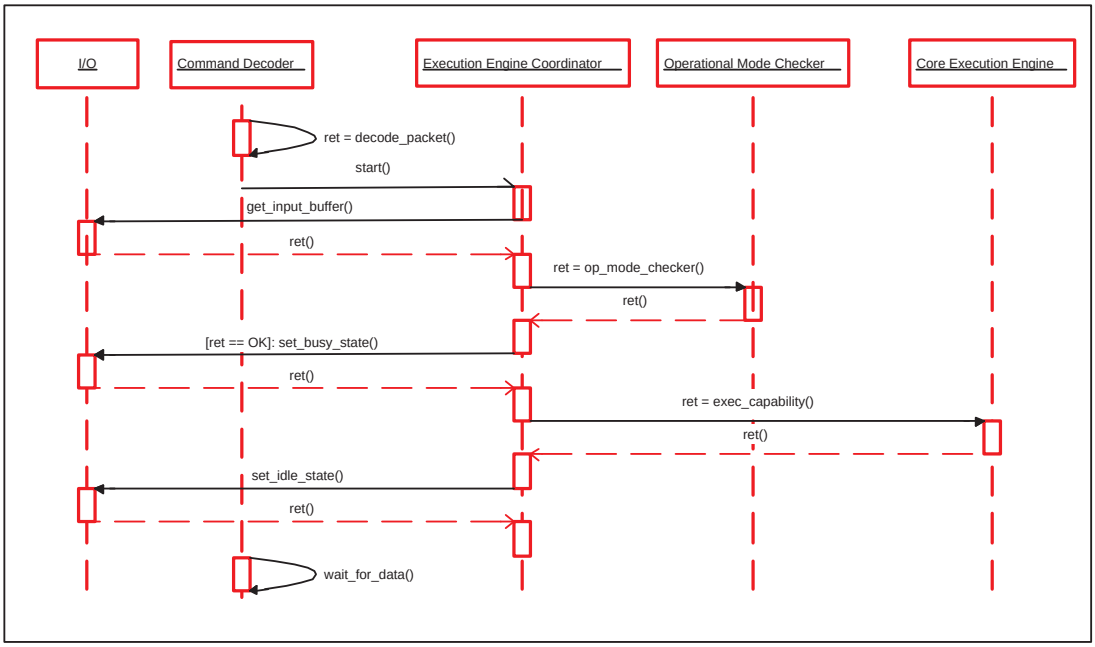

Figura 6.6: Esecuzione di un comando all'interno del TPM: outb

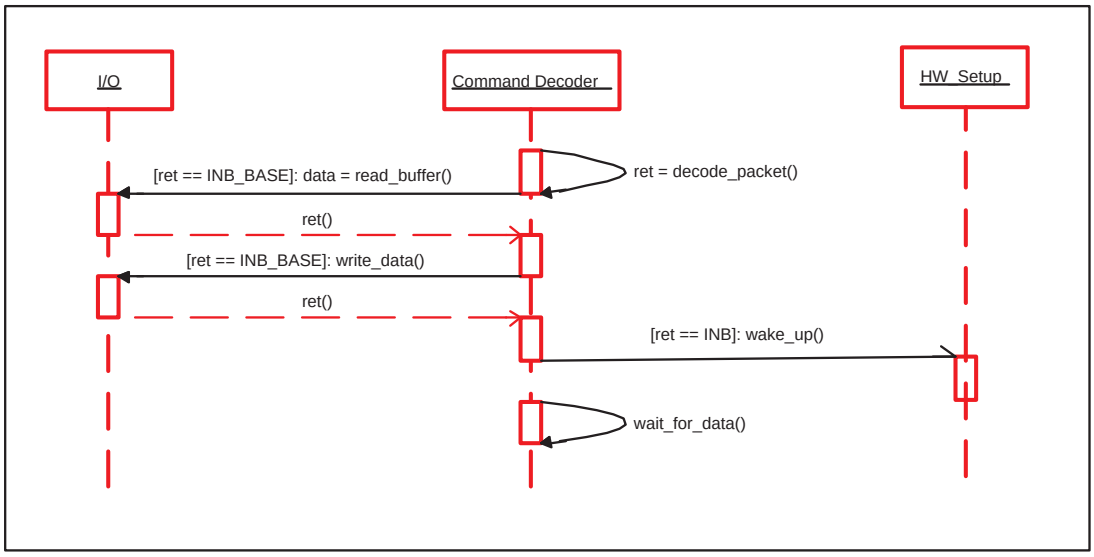

Figura 6.7: Esecuzione di un comando all'interno del TPM: inb su porta DATA

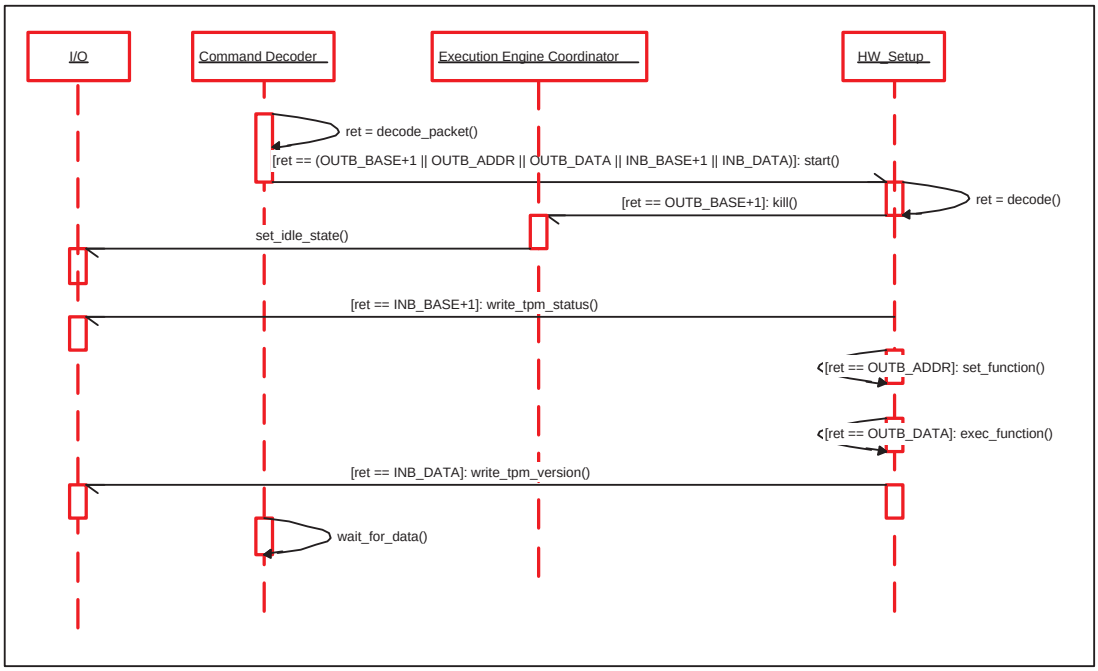

Figura 6.8: Esecuzione di un'operazione di gestione chip TPM

Parte III

La sicurezza dei protocolli di autorizzazione

Capitolo 7

Analisi di sicurezza dei protocolli di autorizzazione

In questo capitolo, vengono descritti brevemente i due protocolli di autorizzazione basilari previsti dalla specifica, OIAP e OSAP e, dopo aver descritto le minacce che li possono intaccare, si presenta uno studio approfondito del protocollo OIAP, descrivendo la proprietà che lo caratterizza e portando a termine un attacco su di esso, attacco dimostrato formalmente attraverso l'impiego di un software automatico di verifica formale.

7.1 Protocolli di autorizzazione: OIAP, OSAP

I protocolli di autorizzazione supportati dalla specifica TCG vengono usati per permettere l'utilizzo di comandi autorizzati (protected capability, vedere sezione 2.5) e:

- gestiscono la creazione confidenziale, cifrata, di dati di autorizzazione,
- forniscono prova della conoscenza dei dati di autorizzazione,
- sono usati per cambiare in modo sicuro dati di autorizzazione.

Dimostrare la conoscenza di un dato di autorizzazione di un oggetto TPM è richiesto prima di usare qualsiasi protected capability che utilizza tale oggetto.

Esistono due protocolli di autorizzazione principali che offrono la base sulla quale si appoggiano altri tre protocolli di autorizzazione descritti nella sottosezione 7.1.2, e sono:

Object Independent Authorization Protocol, OIAP. Permette l'apertura di una sessione di autorizzazione nella quale è possibile l'esecuzione di comandi autorizzati che operano su oggetti anche diversi.

Object Specific Authorization Protocol, OSAP. Permette l'apertura di una sessione di autorizzazione nella quale è possibile eseguire diversi comandi autorizzati che devono, necessariamente, riferirsi tutti ad un oggetto particolare.

7.1.1 Minacce: attacchi di tipo *reply* e *man in the middle*

Data la criticità di questi protocolli, i ricercatori TCG hanno prestato particolare attenzione ai requisiti di sicurezza che tali meccanismi devono offrire; in particolare i protocolli OIAP e OSAP, necessari per la creazione di sessioni all'interno delle quali possono essere eseguiti comandi autorizzati, sono stati progettati per fornire protezioni ai seguenti attacchi[1]:

Reply attack. Questo tipo di attacco dà la possibilità all'attaccante di poter riutilizzare, in modo illecito, informazioni valide scambiate tra le due parti, intercettate su un canale. Nel nostro caso, questo tipo di attacco darebbe la possibilità ad un attaccante di riutilizzare pacchetti, intercettati in una precedente sessione, rappresentanti comandi autorizzati al fine di desincronizzare lo stato della sessione tra TPM e esecutore del comando senza che nessuna delle due entità sia al corrente dell'inconsistenza creata nel protocollo.

Man in the Middle attack. Rappresenta un attacco nel quale un attaccante è in grado di leggere e modificare a proprio piacimento, i messaggi scambiati tra due o

[1] pag. 59 di [10], 48 of 150 cartacea.

più parti, senza che queste sappiano che il collegamento tra di loro sia stato compromesso. Nel nostro caso, questo tipo di attacco darebbe la possibilità ad un attaccante di modificare eventuali comandi autorizzati, in modo tale da far eseguire al TPM operazioni "privilegiate", autorizzate, senza che l'attaccante abbia, in realtà, nessun privilegio.

Come vedremo in seguito, nella sezione 7.2, la specifica TCG usa due protezioni, *rolling nonce* e *HMAC*, al fine di proteggere i protocolli di autorizzazione da questi attacchi.

7.1.2 ADIP, ADCP, AACP

Gli altri tre protocolli di autorizzazione basati su OIAP o OSAP vengono usati per permettere la creazione di un oggetto protetto dal TPM e la modifica di un dato di autorizzazione riferito ad un oggetto già esistente e protetto dal TPM, con o senza garanzia di privacy riferita al dato di autorizzazione modificato. I tre protocolli sono:

Authorization-Data Insertion Protocol, ADIP. È il protocollo usato nel momento della creazione di un oggetto protetto, per inserire un nuovo dato di autorizzazione necessario per poter usare l'oggetto.

Con riferimento a quanto detto nella sezione 3.3, valgono le seguenti assunzioni:

1. un nuovo oggetto protetto dal TPM è creato sempre sotto un oggetto TPM parent, nella gerarchia del protected storage,
2. è necessario che l'utente possa fornire prova della conoscenza del dato di autorizzazione dell'oggetto parent sotto il quale il nuovo oggetto verrà creato,
3. prima di eseguire qualsiasi comando che crei l'oggetto, l'utente deve aprire una sessione OSAP con l'oggetto parent.

L'accesso all'oggetto creato è garantito previa dimostrazione di conoscenza del dato di autorizzazione inserito. Questo significa che anche il proprietario dell'oggetto parent può accedere all'oggetto creato e quindi utilizzarlo; ci sono

situazioni in cui il proprietario dell'oggetto parent è fidato, altre invece dove si vuole garantire maggiormente la privacy del proprietario dell'oggetto appena creato. A tal fine, la specifica TCG ha previsto un protocollo di autorizzazione a due passi, AACP descritto successivamente, che garantisce questa proprietà di privacy.

Authorization Data Change Protocol, ADCP. È progettato per permettere al proprietario di un oggetto TPM di cambiare il suo dato di autorizzazione in modo sicuro con la cooperazione del proprietario dell'oggetto TPM parent. In particolare viene aperta una sessione OSAP con l'oggetto parent che permette la "sprotezione" dell'oggetto child. Viene successivamente aperta una sessione OSAP o OIAP necessaria per poter eseguire comandi autorizzati sull'oggetto child decifrato precedentemente.

Asymmetric Authorization Change Protocol, AACP. È simile al protocollo ADCP, ma permette al proprietario di un oggetto di averne l'utilizzo esclusivo, proteggendo la divulgazione del dato di autorizzazione e impedendo, quindi, che il proprietario dell'oggetto parent, che deve comunque collaborare nella fase iniziale del protocollo, possa accederci.

Il proprietario dell'oggetto parent esegue un comando che permette al TPM di generare una coppia di chiavi asimmetrica temporanea. La parte pubblica viene restituita al chiamante (insieme alla prova che la parte privata è nota solo al TPM), il proprietario dell'oggetto parent, che la inoltra all'utente dell'oggetto child che vuole cambiare il suo dato di autorizzazione.

A questo punto il proprietario dell'oggetto child, esegue un'altro comando che permette di spedire il nuovo dato di autorizzazione, cifrato con la chiave pubblica ottenuta precendentemente, al TPM. In questo modo solo il TPM sarà in grado di decifrare questo segreto e di aggiornare così il dato di autorizzazione relativo all'oggetto in questione.

7.2 Case study: OIAP

Ora che sono state delineate le caratteristiche principali dei protocolli di autorizzazione previsti dalla specifica TCG[2], concentriamo l'analisi su di uno dei due protocolli usati per creare sessioni autorizzate all'interno delle quali eseguire protected capability.

Lo scopo di questa sezione è quindi di descrivere le azioni di "autorizzazione" di un TPM quando questi riceve un comando che è stato autorizzato tramite il protocollo OIAP, che crea una sessione mediante il comando TPM non autorizzato `TPM_OIAP`[3].

Prima di procedere con un esempio di sessione OIAP e di comando autorizzato usato all'interno di questa, è bene accennare a due importanti meccanismi usati dal protocollo al fine di proteggere dalle due minacce descritte nella sottosezione 7.1.1: *HMAC* e il paradigma dei *rolling nonce*.

7.2.1 HMAC

L'HMAC, descritto dall'RFC 2104 [23], fornisce un modo per controllare l'integrità dell'informazione trasmessa o memorizzata su un canale insicuro, attraverso l'uso di funzioni di hash crittografiche. Nella descrizione seguente, *H* indica la funziona di hash crittografica generica[4], sebbene HMAC possa essere usato con funzioni hash quali MD5, SHA-1 e RIPEMD-128/160.

Per calcolare l'HMAC sul dato `payload`[5] è sufficiente eseguire[6]:

$$H(K\ XOR\ opad\ ||\ H(K\ XOR\ ipad\ ||\ payload))$$

dove *K* rappresenta la chiave dell'HMAC, *H* la funzione di hash crittografica usata e *ipad* e *opad* stringhe lunghe quanto un blocco *B*, contenenti, rispettivamente, `0x5c` e `0x36`.

[2] in realtà la versione 1.2 della specifica prevede un'altro protocollo di autorizzazione, Delegate-Specific Authorization Protocol (DSAP), che però non è stato preso in considerazione in questo lavoro.

[3] sessione che rimane aperta indefinitamente e che può essere chiusa da entrambe le parti, TPM o utente (pag. 60 di [10], 49 of 150 cartacea).

[4] La funzione di hash crittografica scelta dalla specifica TCG è SHA-1.

[5] la funzione lavora su input di dimensione multipla di blocci di dati *B*.

[6] per una trattazione più completa dell'HMAC, vedere [23].

Siccome il dato di autorizzazione da presentare per usare l'oggetto protetto dal TPM rappresenta la chiave usata da HMAC, allora la prova di conoscenza del dato di autorizzazione viene fornita al TPM in modo implicito, tramite l'HMAC, senza rivelare il dato stesso, sia che si tratti di trasmissione locale che remota. Data la sua proprietà di "controllore" di integrità, l'HMAC, inoltre, fornisce una garanzia contro eventuali attacchi di tipo *Man in the Middle* volti a manipolare pacchetti intercettati da parte di un attaccante perchè, come verrà spiegato nella sottosezione 7.2.3, l'HMAC viene calcolato su determinati campi dell'header del pacchetto rappresentante il comando autorizzato e, a meno che l'attaccante non sia a conoscenza del dato di autorizzazione, tale integrità non può essere sovvertita senza che il TPM se ne accorga.

7.2.2 Rolling nonce

Usando il paradigma dei *rolling*[7] *nonce*, che altro non sono che numeri pseudo-casuali di 160 bit, i ricercatori TCG proteggono i protocolli di autorizzazione da eventuali attacchi di tipo *reply*.

Infatti un tipico utilizzo di questo paradigma può essere schematizzato dal seguente elenco, dove *Bob* e *Alice* sono le entità comunicanti, msg_{bob} e msg_{alice} sono i messaggi inviati, rispettivamente, da Bob e Alice e la freccia $\rightarrow$ indica la direzione della comunicazione.

Bob $\rightarrow$ **Alice** Bob genera un nonce, $nonce_{bob}$, e lo spedisce insieme a msg_{bob} ad Alice.

Alice $\rightarrow$ **Bob** Alice riceve il messaggio, esegue una qualche computazione e spedisce msg_{alice}, insieme a $nonce_{bob}$ ricevuto precedentemente, a Bob.

Bob controlla che il messaggio ricevuto da Alice contenga $nonce_{bob}$ inviato da lui al primo passo. Se non lo contiene, suppone che il messaggio ricevuto non sia coerente da un punto di vista temporale, perchè non è la risposta al comando inviato da lui nel primo passo. Nel caso Bob volesse rispedire un nuovo messaggio, si preoccuperà di generare un nuovo nonce, prima di dar inizio allo scambio.

[7] rolling nel senso che questi numeri pseudo-casuali vengono scambiati in continuazione tra le parti coinvolte, durante il completamento di un comando.

Questo tipo di protezione non protegge da eventuali *modifiche* del pacchetto ricevuto, ma l'accoppiata di questo paradigma, insieme all'HMAC, dovrebbe garantire, a detta della specifica TCG, protezione da attacchi di tipo reply e man in the middle. Vedremo, tuttavia, che, sfruttando una situazione dove un man in the middle può operare in modo parziale con successo, è possibile far cadere una delle due protezioni usate dalla specifica (come meglio spiegato nella sezione 7.3 e in particolare nella sottosezione 7.3.2).

7.2.3 TPM_Example

Molti comandi usano sessioni create tramite OIAP, quindi è necessario astrarre la trattazione introducendo un comando fittizio, `TPM_Example`, usato per rappresentare un comando autorizzato del TPM. Questo comando usa la chiave denotata da `keyHandle` e l'utente deve conoscere il dato di autorizzazione (vedere sezione 2.5) per `keyHandle`, `key.usageAuth`, poiché questo segreto è usato come dimostrazione della conoscenza e del calcolo dell'autorizzazione. Il parametro `inAuth` fornisce l'autorizzazione per eseguire il comando. La figura 7.1 che segue, mostra il flusso dei comandi eseguiti e cosa passa in chiaro sul canale di comunicazione, mentre la tabella 7.1 mostra l'header associato con il comando `TPM_Example`, evidenziando i campi coinvolti nel calcolo dell'HMAC[8].

In particolare, `<inParamDigest>` è il risultato del calcolo *SHA1(ordinal, inArgOne, inArgTwo)*, `<outParamDigest>` è invece il risultato di *SHA1(returnCode, ordinale, outArgOne)*, `<inAuthSetupParams>` si riferisce ai parametri, in ordine, *authHandle, authLastNonceEven, nonceOdd, continueAuthSession* mentre l'ultimo aggregato, `<OutAuthSetupParams>`, si riferisce ai parametri, sempre in ordine *authHandle, nonceEven, nonceOdd, continueAuthSession*.

Esistono due *nonce* pari usati per eseguire il comando `TPM_Example`, quello generato come parte del comando `TPM_OIAP`, *authLastNonceEven*, e *nonceEven*, generato con l'output del comando.

[8] la doppia linea di separazione in tabella 7.1, distingue i parametri della sessione di autorizzazione dai parametri veri e propri del comando.

Supponiamo che l'entità coinvolta nell'esecuzione del comando, oltre al TPM, sia chiamata *caller*. Allora, come illustrato in figura 7.1 i passi per poter eseguire il comando autorizzato `TPM_Example` sono descritti qui di seguito:

1. Il caller apre una sessione OIAP tramite il comando `TPM_OIAP`. Il TPM, una volta ricevuto il comando ed eseguito alcuni sanity check,
   - crea una sessione e un handle, *authHandle*, da associare a tale sessione,
   - genera *authLastNonceEven* e lo associa alla sessione creata,
   - restituisce al caller *authHandle* e *authLastNonceEven*.
2. il caller
   - genera *nonceOdd*,
   - calcola *inAuth = HMAC(key.usageAuth, inParamDigest, inAuthSetupParams)*,
   - salva *nonceOdd* insieme alle informazioni ricevute dal TPM sulla sessione da lui creata.
3. ...e infine spedisce il comando `TPM_Example` avendo cura di riempire in modo appropriato tutti i campi dell'intestazione del comando (tabella 7.1).

Come è possibile osservare nel punto 2, il dato di autorizzazione necessario per utilizzare l'oggetto sul quale il comando `TPM_Example` opera, rappresenta la *chiave* dell'HMAC calcolato[9], che garantisce per l'integrità del pacchetto, impedendo eventuali manipolazioni/alterazioni da parte di attaccanti e man in the middle.

[9] tale funzione è calcolata anche su `inParamDigest` e `inAuthSetupParams`.

HMAC	Type	Name	Description
	TPM_TAG	tag	TPM_TAG_RQU_AUTH1 _COMMAND
	UINT32	paramSize	Dimensione del pacchetto, incluso tag e paramSize
sì	TPM_COMMAND_CODE	ordinal	Identificatore del comando: TPM_Example
	TPM_KEY_HANDLE	keyHandle	Handle della chiave caricata nel TPM
sì	BOOL	inArgOne	Primo parametro di input
sì	UINT32	inArgTwo	Secondo parametro di input
sì	TPM_AUTHHANDLE	authHandle	Handle dell'autorizzazione usata per keyHandle
sì	TPM_NONCE	authLastNonceEven	Nonce pari generato precedentemente dal TPM (contromisura per *reply* attack)
sì	TPM_NONCE	nonceOdd	Nonce dispari generato dal caller, associato con authHandle
sì	BOOL	continueAuthSession	Chiudi la sessione?
	TPM_AUTHDATA	inAuth	Il digest dei dati di autorizzazione di input per keyHandle. Chiave HMAC: key.usageAuth

Tabella 7.1: Header del comando autorizzato TPM_Example

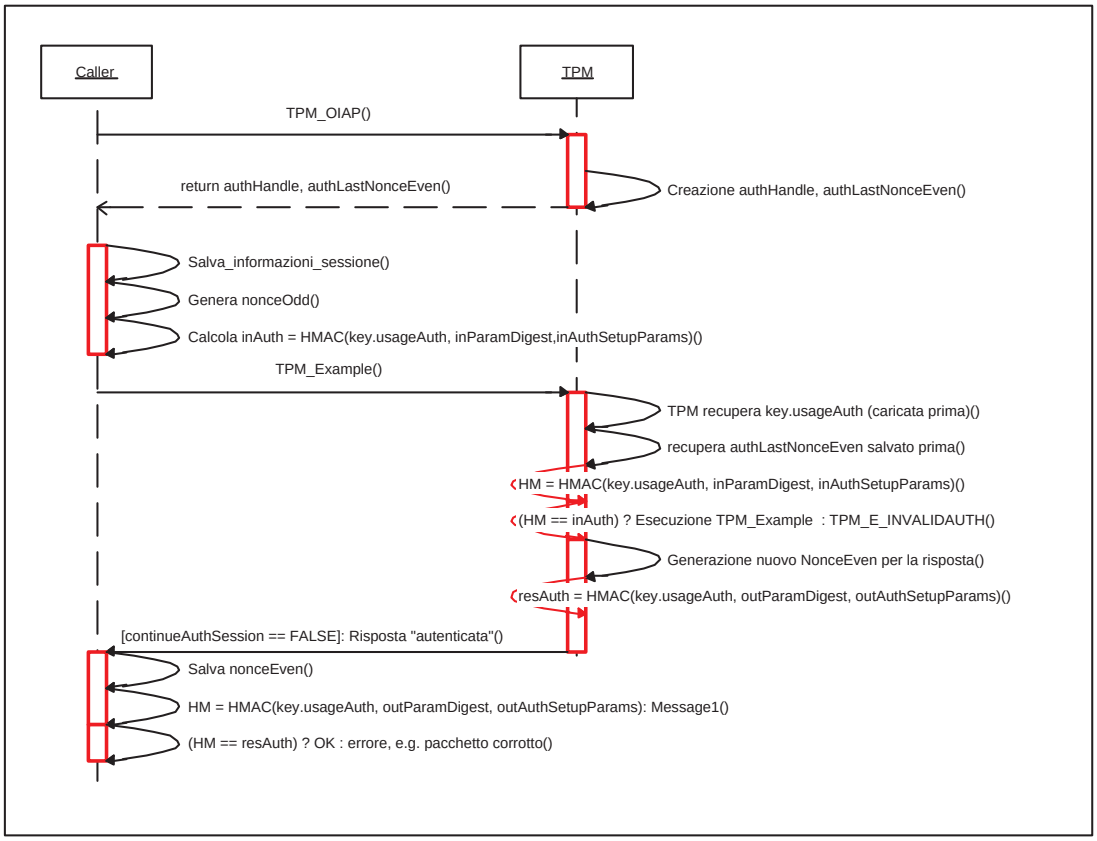

Figura 7.1: TPM_Example

7.3 Model checker, analisi di sicurezza

L'analisi del protocollo di autorizzazione OIAP è stata fatta attraverso l'impiego di uno strumento software [22] usato per la verifica formale in modo automatico di protocolli e sistemi software distribuiti.

La verifica formale di protocolli, ad esempio, può essere effettuata mediante l'individuazione di determinate proprietà espresse secondo una logica temporale del primo ordine (Linear Temporal Logic, LTL) o espresse come invarianti, usando delle asserzioni (tecnica usata nella dimostrazione dell'attacco). La correttezza della proprietà porta alla correttezza del protocollo esaminato, a meno che non sia fallace la fase di "modellazione"[10] del protocollo stesso.

7.3.1 Modellazione del protocollo OIAP

Il primo passo da eseguire per poter procedere alla verifica formale della correttezza del protocollo OIAP è quello di descriverlo, e quindi esprimerlo, con un linguaggio appropriato, capibile dal model checker. Il linguaggio usato dal model checker *spin* [22], si chiama *PROMELA (a PROcess MEta LAnguage)* e la descrizione completa del protocollo OIAP mediante questo linguaggio è presentata nell'appendice A.

La descrizione del protocollo ha portato all'individuazione di tre attori che interagiscono al fine di poter eseguire il comando d'esempio sopra descritto. I due attori *TPM* e *Caller* sono le parti che interagiscono per poter eseguire in modo corretto il comando autorizzato, mentre il terzo, *MITM*, rappresenta l'attaccante che, come vedremo in seguito, è in grado di sovvertire il protocollo, portando un attacco di tipo reply. In particolare, il MITM è stato descritto in modo generico e si è supposto che possa eseguire anche operazioni in modo "disordinato", senza seguire "letteralmente" il protocollo OIAP. Questa astrazione ha introdotto, tuttavia, un certo livello di *non determinismo* associato con le operazioni compiute da questo attore, facendo aumentare la complessità dell'automa generato dal model checker e non permettendo di analizzare, in modo esaustivo, tutte le strade determinate dal modello. Questo vincolo

[10] descrizione del protocollo in un linguaggio capibile dallo strumento utilizzato.

apparente, però, non rappresenta un limite alla dimostrazione dell'attacco effettuato, in quanto è stato sufficiente trovare un solo cammino, nell'automa generato dal model checker, che ha portato all'individuazione con successo dell'attacco.

Il MITM rappresenta un *Man in the Middle* e come tale è proprio in mezzo alla comunicazione tra caller e TPM[11]; infatti è possibile individuare due canali di comunicazione instaurati, ad insaputa degli altri due attori, tra Caller e MITM e tra MITM e TPM.

Le operazioni che il MITM può eseguire sono indicate nell'elenco sottostante:

- aprire sessioni OIAP, tramite il comando `TPM_OIAP` e lasciarle aperte indefinitamente o chiuderle a proprio piacimento,
- modificare e inviare (o re-inviare) pacchetti recuperati dalla comunicazione tra Caller $\leftrightarrow$ MITM e MITM $\leftrightarrow$ TPM,
- inviare o reinviare i pacchetti *registrati* in ordine casuale.

Proprietà del protocollo

La descrizione del protocollo OIAP ha portato all'individuazione di una proprietà che *non deve mai verificarsi* affinché il protocollo possa operare in modo corretto.

Tale proprietà si riferisce alla conoscenza dello stato della sessione da parte del Caller e del TPM ed è espressa dal seguente frammento di codice, dove gli indici 0 e 1 si riferiscono al numero di sessione gestita dal TPM.

```
#define PROPERTY do \
{ \
   (  \
      (tpm_session_understanding[0] == FAIL &&              \
         caller_session_understanding[0] == SUCCESS) ||    \
      \
      (tpm_session_understanding[1] ==  FAIL && \
```

[11] si considera un contesto in cui il caller vuole comunicare con il TPM per via remota. Questa ipotesi non è restrittiva ed è prevista dalla specifica TCG [10].

```
         caller_session_understanding[1] ==  SUCCESS) ||   \
      \
      (tpm_session_understanding[0] ==  SUCCESS && \
         caller_session_understanding[0] ==  FAIL) ||      \
      \
      (tpm_session_understanding[1] ==  SUCCESS  &&        \
         caller_session_understanding[1] ==  FAIL)         \
   ) \
} while (0)
```

In pratica, se la proprietà sopra enunciata si verifica, il MITM è stato in grado di portare il protocollo in uno stato inconsistente, nel quale le due parti non hanno la stessa visione del risultato del comando autorizzato eseguito nella sessione OIAP, come descritto nella seguente sottosezione.

7.3.2 Attacco al protocollo OIAP

La verifica della proprietà sopra enunciata, permette all'attaccante che può controllare la comunicazione tra le parti, di portare a termine un attacco di tipo *reply*, come mostrato in figura 7.2. Il protocollo OIAP, comunque, offre protezione sufficiente da attacchi di tipo MITM veri e propri, dove l'attaccante può modificare, eliminare o introdurre pacchetti nella comunicazione. L'HMAC usato dal protocollo, infatti, offre la protezione necessaria in quanto tramite il suo utilizzo il TPM può controllare l'integrità del comando ricevuto e, implicitamente, il dato di autorizzazione inviato (coinvolto, come spiegato nella sottosezione 7.2.1 nel calcolo dell'HMAC) che, a meno che non sia conosciuto, non può essere specificato e falsificato dal MITM.

L'attacco mostrato in figura 7.2, portato avanti dal MITM, si articola nei passi[12] che seguono. Per semplicità e perché la specifica offre protezione da attacchi MITM, il calcolo dell'HMAC non è coinvolto nella descrizione del protocollo.

1. Il Caller esegue il comando `TPM_OIAP` che viene intercettato dal MITM e quindi non raggiunge in realtà il TPM. `caller_session_understanding[0]`

[12] esistono, comunque, altre sequenze che portano a termine l'attacco con successo.

e `tpm_session_understaning[0]` contengono il valore `UNKNOWN` in quanto non è stato ancora eseguito nessun comando TPM autorizzato; lo stato della sessione è, pertanto, sconosciuto.

2. Il MITM apre una sessione OIAP con il TPM che gli restituisce, come risposta, i nonce relativi. Il MITM restituisce tali dati anche al Caller. Questa operazione può essere effettuata in quanto il comando `TPM_OIAP` è un comando non autorizzato e le informazioni restituite non sono autenticate.

3. Il Caller prosegue con il protocollo OIAP, eseguendo il comando `TPM_Example`, usando però i valori restituiti dal MITM, valori legati alla sessione aperta tra MITM e TPM e non tra Caller e TPM.

4. Il MITM intercetta il pacchetto contenente il comando autorizzato e lo conserva. Nello stesso tempo restituisce al Caller una risposta, fingendosi il TPM. Tale pacchetto di risposta, non può essere corretto, perchè il calcolo dell'HMAC coinvolto nella risposta deve tenere conto del dato di autorizzazione (oltre ad altre informazioni) conosciuto dal TPM e dal Caller, ma non dal MITM. Questo non è un problema per il MITM, ma porta lo stato della sessione del Caller da `UNKNOWN` a `FAIL`.

5. Il MITM rigira il pacchetto registrato nel punto precedente al TPM che, dopo averlo verificato, lo esegue[13]. Questa operazione porta lo stato della sessione del TPM da `UNKNOWN` a `SUCCESS` di fatto rendendo inconsistente la conoscenza della sessione tra TPM e Caller.

[13] sempre se i dati di autorizzazione usati dal Caller sono corretti.

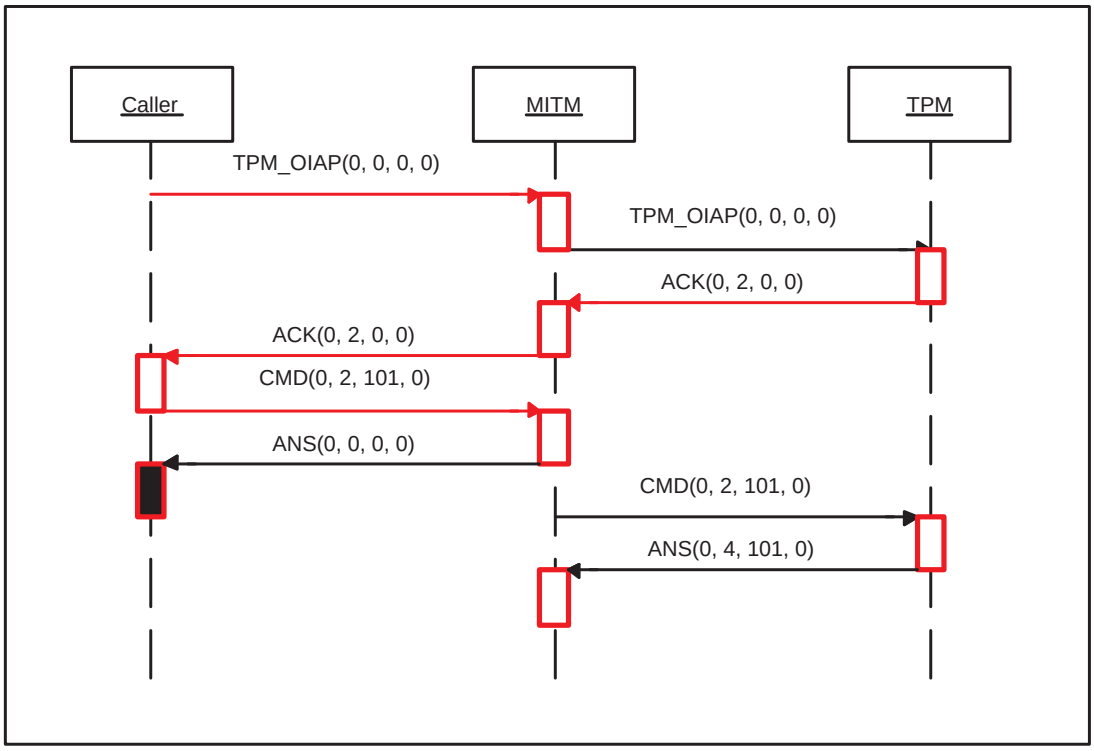

Figura 7.2: Attacco di tipo *reply* sfruttando *mitm* al protocollo OIAP

Capitolo 8

Conclusioni e sviluppi futuri

Questo lavoro di tesi ha permesso di analizzare e studiare le *piattaforme di calcolo fidate* (Trusted Computing Platform, TP), tecnologia proposta dal Trusted Computing Group.

Dopo un primo studio dell'architettura TP che ha permesso di individuare le componenti e le funzionalità principali introdotte con lo scopo di coadiuvare i meccanismi di sicurezza esistenti, si è passati alla progettazione e realizzazione di un emulatore software dell'intera architettura proposta da TCG.
Questa strada permette un continuo adattamento alla specifica, senza esser vincolati alle politiche aziendali di rilascio specifiche e componenti hardware o software, così come facilita l'aggiunta o la rimozione di nuove funzionalità all'emulatore, proponendo e sperimentando situazioni e scenari difficilmente realizzabili via hardware.

La realizzazione di un'emulatore permette, inoltre, di comprendere maggiormente l'archiettura proposta, dovendo obbligatoriamente considerare tutti i dettagli della specifica che possono venire, a volte, superficialmente letti, ma che rappresentano fondamenta importanti per il corretto funzionamento dell'architettura. Questi "dettagli" hanno permesso di affrontare una fase di analisi di sicurezza dei meccanismi di sicurezza sui cui si basa l'intera architettura.

Due sono infatti i protocolli di autorizzazione, OIAP e OSAP, che rappresentano le fondamenta di questi meccanismi e l'analisi di uno di essi, l'OIAP, ha permesso di determinare una sua falla progettuale, attraverso la quale è possibile sovvertire la

funzionalità più importante introdotta dalle Trusted Platform.

Nonostante la problematica[1] sorta, le TP rappresentano, comunque, un importante sostegno ai meccanismi di sicurezza esistenti ed è possibile pensare ad un loro utilizzo, ad esempio, al fine di contrastare le più diffuse minacce alla sicurezza delle informazioni, quali buffer overflow e attacchi diretti in memoria, al fine di avere garanzie di integrità sull'ambiente rappresentato dai processi (in esecuzione) e non solo per i programmi di una piattaforma di calcolo.

[1] la specifica è *aperta* ed è questo che ha reso possibile la scoperta della falla e che renderà possibile la sua correzione.

Bibliografia

[1] *Common Criteria*. More info at http://www.commoncriteria.org/cc/cc.html.

[2] *IBM T. J. Watson Research Center – Global Security Analysis Lab*. More info at http://www.research.ibm.com/gsal.

[3] *Intel LPC Specification*. More info at http://www.intel.com/design/chipsets/industry/25128901.pdf.

[4] *Non-exec page protection, ASLR and so on – The PaX Team*. Available at http://pax.grsecurity.net.

[5] *Non-exec page protection for Windows*. Available at http://www.idefense.com.

[6] *Protection Profile*. More info at http://csrc.nist.gov/cc/Protection_Profiles.html.

[7] *RSA Lab, PKCS1 – RSA Cryptography Standard*. Available at http://www.rsasecurity.com/rsalabs/pkcs/pcks-1/index.html.

[8] *RSA Lab, PKCS1 v2.0 Amendment 1: Multi-Prima RSA*. Available at http://www.logi.org/documentation/crypto/standards/pkcs-1v2-0a1.pdf.

[9] *RSA Lab, PKCS1 v2.1: RSA Cryptography Standard (Draft 2)*. Available at http://www.logi.org/documentation/crypto/standards/pkcs-1v2-1d2.pdf.

[10] *TCG v1.2 Design Principles*. Available at https://www.trustedcomputinggroup.org/downloads/tpmwg-mainrev62_Part1_Design_Principles.pdf.

[11] *TCG v1.2 TPM Commands*. Available at https://www.trustedcomputinggroup.org/downloads/tpmwg-mainrev62_Part3_Commands.pdf.

[12] *TCG v1.2 TPM Structures*. Available at https://www.trustedcomputinggroup.org/downloads/tpmwg-mainrev62_Part2_TPM_Structures.pdf.

[13] *Trusted Computing Group*. More info at http://www.trustedcomputinggroup.org.

[14] W. Arbaugh, D. Farber e J. Smith. A secure and reliable bootstrap architecture. In *Proceedings 1997 IEEE Symposium on Security and Privacy, pages 65–71*. May, 1997.

[15] Atmel. *Trusted Platform Module*. For info http://www.atmel.com/dyn/products/product_card.asp?part_id=2604.

[16] B. Balacheff, L. Chen, S. Pearson, D. Plaquin e G. Proudler. *Trusted Computing Platforms, tcpa technology in context*. Prentice Hall, 2003.

[17] J. Dicke. *User-Mode Linux Kernel Patch*. Available at http://user-mode-linux.sourceforge.net.

[18] J. Dyer, M. Lindemann, R. Perez, R. Sailer, S. W. Smith, L. van Doorn e S. Weingart. Building the ibm 4758 secure coprocessor. In *IEEE Computer 34, pages 57-66*. October, 2001.

[19] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd edition*. Addison-Wesley, 2004.

[20] S. Gafinkel, G. Spafford e A. Schwartz. *Practical Unix & Internet Security, 3rd edition*. O'Reilly.

[21] R. A. Grimes. *Malicious Mobile Code: Virus protection for Windows*. O'Reilly, 2001.

[22] G. J. Holzmann. *The Spin Model Checker*. For info http://www.spinroot.com.

[23] H. Krawczyk, M. Bellare e R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. For info ftp://ftp.rfc-editor.org/in-notes/rfc2104.txt.

[24] E. Matricciani. *Fondamenti di comunicazione tecnico-scientifica*. Apogeo, 2003.

[25] A. Rubini e J. Corbet. *Linux Device Drivers, 2nd edition*. O'Reilly, June 2001.

[26] R. Saiter, X. Zhang, T. Jaeger e L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *13th Usenix Security Symposium, pages 223-238*. August, 2004.

[27] W. Stallings. *Cryptography and Network Security. Principles and Practice, 2nd edition*. Prentice-Hall, 1999.

[28] E. D. Zwicky, S. Cooper e D. B. Chapman. *Building Internet Firewalls, 2nd edition*. O'Reilly.

Elenco delle figure

Elenco delle tabelle

Appendice A

PROMELA: descrizione del protocollo OIAP

```
mtype = {TPM_OIAP, ACK, CMD, ANS, FINISH}

typedef payload {
   /* cmd */
   /* int id; */
   /* args */
   bit session; /* only 2 sessions permitted */
   short ne;
   short no;
   bool cont;
   /* int hmac; */
}

chan caller_mitm_wire = [0] of { mtype,  payload };
chan mitm_tpm_wire = [0] of { mtype,  payload };
byte no[2];
byte ne[2];
bool opened_oiap[2];


/* #define HMAC(a, b, c)  ((b)*10000 + (c)*10 + (a)) */
```

```
#define CONSISTENT(s) (no[s] == r_no && ne[s] == r_ne)

#define FAIL 2
#define SUCCESS 1
#define UNKNOWN 0

byte tpm_session_understanding[2];
byte caller_session_understanding[2];

#define MAX_CALLER_ACTIVITY 100

proctype
caller(bool cont)
{

/* cont: continue session with another command? */
byte r_no, r_ne;  /* received even nonce and odd nonce */
bit r_session;    /* received session */

   do
      :: no[0] + no[1] > MAX_CALLER_ACTIVITY -> break
      :: else ->
         /*
          * reset session understanding to unknown before
          * sending a new OIAP command
         */
         caller_session_understanding[0] = UNKNOWN;
         caller_session_understanding[1] = UNKNOWN;
         caller_mitm_wire!TPM_OIAP(0,0,0,0);
         caller_mitm_wire?ACK(r_session,r_ne,_,_);

         do
            :: no[0] + no[1] > MAX_CALLER_ACTIVITY -> goto END
            :: else ->
               atomic {
                  no[r_session] = no[r_session] + 2;
```

```
                caller_session_understanding[r_session] = UNKNOWN;
                caller_mitm_wire!CMD(r_session, r_ne, \
                   no[r_session], cont);
             }
             atomic {
                caller_mitm_wire?ANS(eval(r_session), r_ne, \
                   r_no, _);
                if
                   :: !CONSISTENT(r_session) ->
                      /* Exit without retrying */
                      caller_session_understanding[r_session] = \
                         FAIL; break
                   :: !CONSISTENT(r_session) ->
                      caller_session_understanding[r_session] = \
                         FAIL;
                   :: CONSISTENT(r_session) && cont ->
                      caller_session_understanding[r_session] = \
                         SUCCESS;
                   :: CONSISTENT(r_session) && !cont ->
                      caller_session_understanding[r_session] = \
                         SUCCESS;
                      break /* Exit without retrying */
                fi;
             }
          od;
   od;
END:
   printf("End Caller")
}

proctype
tpm( bit session )
{

byte r_ne, r_no;     /* received even and odd nonce */
bool r_cont;         /* session continued */
bool consistent;
```

```
do
   :: mitm_tpm_wire?FINISH(_,_,_,_) -> break
   :: mitm_tpm_wire?TPM_OIAP(_,_,_,_) ->
      atomic {
         if
            :: !opened_oiap[session] ->
               opened_oiap[session] = true;
               ne[session] = ne[session] + 2;
               tpm_session_understanding[session] = UNKNOWN;
               mitm_tpm_wire!ACK(session, ne[session], 0, 0);
            :: else -> mitm_tpm_wire!ANS(session,0,0,0);
         fi
      }
      do
         :: mitm_tpm_wire?FINISH(_,_,_,_) -> goto END
         :: atomic {
            mitm_tpm_wire?CMD(eval(session), r_ne, r_no, \
               r_cont);
            consistent = CONSISTENT(session) };
            if
               :: atomic { consistent ->
                  if
                     :: (tpm_session_understanding[session] == \
                        UNKNOWN) ->
                        tpm_session_understanding[session] = \
                           SUCCESS;
                        ne[session] = ne[session] + 2;
                        atomic {
                           mitm_tpm_wire!ANS(session, ne[session], \
                              r_no, r_cont);
                           opened_oiap[session] = false;
                        }
                        if
                           :: r_cont -> skip
                           :: else -> break
                        fi;
```

```
                        :: else -> atomic {
                           mitm_tpm_wire!ANS(session, 0, 0, 0);
                           opened_oiap[session] = false;
                        }
                     fi;
                     }
                  :: else -> atomic {
                     tpm_session_understanding[session] = FAIL;
                     mitm_tpm_wire!ANS(session, 0,0,0);
                     opened_oiap[session] = false;
                  }
               fi;
         od;
   od;
END:
   printf("End TPM")
}

proctype
mitm()
{

bit r_oiap_session, r_ack_session, r_cmd_session, r_ans_session;
short r_oiap_ne, r_ack_ne, r_cmd_ne, r_ans_ne;
short r_oiap_no, r_ack_no, r_cmd_no, r_ans_no;
bool r_oiap_cont, r_ack_cont, r_cmd_cont, r_ans_cont;
int mitm_activity;

#define MAX_MITM_ACTIVITY (MAX_CALLER_ACTIVITY * 4)

mitm_activity = 0;

do
   :: ( mitm_activity > MAX_MITM_ACTIVITY ) -> break;
   :: else ->
      :: caller_mitm_wire?FINISH(_,_,_,_);
         mitm_activity++;
```

```
:: caller_mitm_wire?TPM_OIAP(r_oiap_session,r_oiap_ne,\
      r_oiap_no,r_oiap_cont);
   mitm_activity++;
:: caller_mitm_wire?ACK(r_ack_session,r_ack_ne,r_ack_no,\
   r_ack_cont);
   mitm_activity++;
:: caller_mitm_wire?CMD(r_cmd_session,r_cmd_ne,r_cmd_no,\
   r_cmd_cont);
   mitm_activity++;
:: caller_mitm_wire?ANS(r_ans_session,r_ans_ne,r_ans_no,\
   r_ans_cont);
   mitm_activity++;

:: caller_mitm_wire!TPM_OIAP(r_oiap_session,r_oiap_ne,\
   r_oiap_no,r_oiap_cont);
   mitm_activity++;
:: caller_mitm_wire!ACK(r_ack_session,r_ack_ne,r_ack_no,\
   r_ack_cont);
   mitm_activity++;
:: caller_mitm_wire!CMD(r_cmd_session,r_cmd_ne,r_cmd_no,\
   r_cmd_cont);
   mitm_activity++;
:: caller_mitm_wire!ANS(r_ans_session,r_ans_ne,r_ans_no,\
   r_ans_cont);
   mitm_activity++;
:: caller_mitm_wire!ANS(0,0,0,0);
   mitm_activity++;

:: mitm_tpm_wire?TPM_OIAP(r_oiap_session,r_oiap_ne,r_oiap_no,\
   r_oiap_cont);
   mitm_activity++;
:: mitm_tpm_wire?ACK(r_ack_session,r_ack_ne,r_ack_no,\
   r_ack_cont);
   mitm_activity++;
:: mitm_tpm_wire?CMD(r_cmd_session,r_cmd_ne,r_cmd_no,\
   r_cmd_cont);
   mitm_activity++;
```

```
      :: mitm_tpm_wire?ANS(r_ans_session,r_ans_ne,r_ans_no,\
         r_ans_cont);
         mitm_activity++;

      :: mitm_tpm_wire!TPM_OIAP(r_oiap_session,r_oiap_ne,r_oiap_no,\
         r_oiap_cont);
         mitm_activity++;
      :: mitm_tpm_wire!ACK(r_ack_session,r_ack_ne,r_ack_no,\
         r_ack_cont);
         mitm_activity++;
      :: mitm_tpm_wire!CMD(r_cmd_session,r_cmd_ne,r_cmd_no,\
         r_cmd_cont);
         mitm_activity++;
      :: mitm_tpm_wire!ANS(r_ans_session,r_ans_ne,r_ans_no,\
         r_ans_cont);
         mitm_activity++;
      :: mitm_tpm_wire!ANS(0,0,0,0);
         mitm_activity++;
od
}


#define PROPERTY ( (tpm_session_understanding[0] == FAIL \
&& caller_session_understanding[0] == SUCCESS) || \
(tpm_session_understanding[1] == FAIL \
&& caller_session_understanding[1] == SUCCESS) || \
(tpm_session_understanding[0] == SUCCESS \
&& caller_session_understanding[0] == FAIL) || \
(tpm_session_understanding[1] == SUCCESS \
&& caller_session_understanding[1] == FAIL) )

active proctype monitor(){
atomic{
   PROPERTY -> assert(!PROPERTY);
}
}
```

```
init{

   atomic { no[0] = 1; no[1] = 1; ne[0]=0; ne[1]=0;
      caller_session_understanding[0] = UNKNOWN;
      caller_session_understanding[1] = UNKNOWN;
      tpm_session_understanding[0] = UNKNOWN;
      tpm_session_understanding[1] = UNKNOWN;
      opened_oiap[0] = false;
      opened_oiap[1] = false;
      run tpm(0); /* run tpm(1); */
      run mitm();
      run caller(0); /* run caller(0);   */
      timeout -> mitm_tpm_wire!FINISH(0,0,0,0);
                 caller_mitm_wire!FINISH(0,0,0,0);
      }
}
```

Finito di stampare il 12 dicembre 2004 utilizzando LaTeX 2ε