POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Protezione e attestazione remota del software con Intel SGX

Relatori
Prof. Antonio Lioy
Dott. Ing. Daniele Canavese
Dott. Ing. Leonardo Regano

Candidato
Gabriele Sergio

Anno accademico 2022-2023

Ma non tocca a noi scegliere. Tutto ciò che possiamo decidere è come disporre del tempo che ci è dato.

Ringraziamenti

In questo lavoro che sento molto personale, compagno di un turbine di svolte nella mia vita, mi piacerebbe lasciare alcune parole di ringraziamento che diventeranno ad ogni rilettura sempre più cariche di significato, come lo è stato il mio percorso al Politecnico negli anni, iniziato da una partenza ingenuamente liberatoria e finito in vari imprevisti incontrati e eventi di cause maggiori che hanno cambiato alcuni piani, portando a scegliere deviazioni imprevedibili.
Ringrazio i familiari che hanno compiuto sacrifici nei miei confronti e chi mi ha lasciato valori in cui credo, che ammiro e non potrei mai rinnegare. In special modo, l'affetto per mia sorella *Ilaria* mi ha permesso di resistere in tanti momenti.
Ringrazio chi mi ha dato di più di quanto abbia ricevuto da me durante tanti anni, insegnandomi cosa sia davvero un amico speciale, come *Federico*, il cui rapporto fraterno è andato oltre negli anni, rendendo notti di lockdown piene di senso, e la stima, la comprensione e l'ammirazione per quello che realizza oggi non si è ancora spenta. E ugualmente, ringrazio chi mi ha lasciato sensazioni simili in molto meno tempo, sorprendendomi e dandomi prova di come non è la presenza nel tempo a fare la differenza ma è la qualità del tempo speso, e sento di voler dedicare ad *Alessandro* un segno di affetto e gratitudine per quel poco che penserà di aver fatto.
Ringrazio chi non ricambia la mia gentilezza con inutili parole di esaltazione ma con altrettanta pura gentilezza, pazienza e protezione, scegliendo di condividere fragilità importanti: *Alessio*, *Chiara* e *Valeria*, presenti negli anni di lezioni universitarie, sono alcuni dei nomi appartenenti a questo gruppo di persone e di amici che non ho timori a definire tali.
Ringrazio con affetto persone come *Ale* che mi hanno mostrato la bellezza nel voler bene ad una persona cara facendo più del dovuto, *Silvia* per aver contribuito ad avere ricordi che porterò sempre nel mio cuore e chi ha a mia sorpresa portato rispetto, chi è stato paziente sopportando i miei antipatici bassi e chi mi ha offerto bontà e disponibilità confortanti per la purezza dimostrata. E ringrazierò sempre persone così.

Indice

Capitolo 1

Introduzione

Il mondo informatico si è evoluto in maniera esponenziale dall'inizio del XXI secolo: i cambiamenti storici hanno accelerato il processo di digitalizzazione della società e di integrazione dell'informatica nel reale, e la recente pandemia di COVID-19 è stata ancora più determinante per l'implementazione di tecnologie digitali in ogni settore lavorativo e nell'uso quotidiano (basti pensare all'introduzione della Certificazione COVID digitale dell'UE[1], e ai relativi dibattiti su quali dati conservare e come conservarli, in rispetto alle normative europee che si occupano della tutela dei dati personali, come il GDPR[2]). Anche il modo di consumare digitale è cambiato: l'uso di servizi cloud, che spaziano da piattaforme di backup di dati anche potenzialmente molto sensibili a spazi di lavoro e testing senza che ci sia un investimento fisico, o l'aumento di dispositivi appartenenti alla gamma di Internet of Things sono prove di come il mondo sia diventato molto più connesso rispetto al passato. Come conseguenza al cambiamento del contesto tecnologico, anche il numero di attacchi informatici è aumentato: solo nel 2021, il numero di cyberattacchi effettuati e di casi di fuga di dati (o data breach) è aumentato del 15,1% rispetto all'anno precedente secondo Forbes[3], ed è stato calcolato da IBM che il costo medio per aver subito un data breach è arrivato nel 2022 a costare il massimo storico di 4,35 miliardi di dollari, circa il 2,6% rispetto al 2021 e il 12,7% rispetto al 2020[4]. Le informazioni e gli asset con un certo valore sono cresciuti numericamente, sono trattate informazioni molto più sensibili e preziose come quelle legate all'ambito medico e all'ambito finanziario, è diventato quindi più redditizio puntare all'ottenimento di queste risorse, anche grazie ad una combinazione di facilità nell'esecuzione degli attacchi e gli sforzi insufficienti nell'usare prodotti sicuri e adottare comportamenti responsabili e coscienti delle varie insidie informatiche. Per questo motivo, è diventato più importante trovare tecnologie e meccanismi di protezione delle risorse. Avere ambienti di lavoro ed esecuzione di applicazioni sicuri e affidabili è una soluzione che aiuta al prevenire tentativi di manomissione del codice, garantendo

[1] https://www.dgc.gov.it/web/

[2] https://gdpr-info.eu/

[3] https://www.forbes.com/sites/chuckbrooks/2022/06/03/alarming-cyber-statistics-for-mid-year-2022-that-you-need-to-know/

[4] https://www.ibm.com/security/data-breach

confidenzialità e integrità: queste tecniche sono conosciute come Trusted Execution Environment (TEE), e molti produttori di hardware e software hanno investito su questo settore.
Intel appartiene al gruppo dei produttori che hanno lavorato ad una soluzione proprietaria di questo tipo, e il loro prodotto più recente è Secure Guard Extensions (SGX)[5], che aiuta ad isolare l'applicazione e i dati che elabora dal resto del sistema, in aree che sono protette da attacchi grazie ad una combinazione di componenti hardware e funzionalità software. I programmatori possono così adottare la tecnologia tramite i processori abilitati e con il Software Development Kit[6] che permette la creazione di applicazioni conformi al modello di protezione offerto. Per le caratteristiche che offre, SGX è stato implementato per la gestione dei Digital Rights Management nella riproduzione di dischi Blu-Ray[7], dai Service Provider di piattaforme cloud come Microsoft Azure[8], phoenixNAP[9] e G-Core Labs[10], in servizi che implementano Intelligenze Artificiali come BeeKeeperAI[11] in campo medico e Fortanix[12]. Sebbene i vantaggi di SGX, e delle tecnologie Trusted Execution Environment in generale, siano numerosi, applicare queste tecnologie richiede da parte dei programmatori un lavoro maggiore, affinché le interfacce che usano le funzionalità offerte siano integrate dall'applicazione, ma anche per quanto riguarda il design dell'applicazione che deve essere costruito attorno al modello di protezione offerto. Per ottimizzare il processo di costruzione di una applicazione, sono realizzati dei framework che si occupano dell'automatizzazione del processo di conversione di applicazioni, lavorando sul codice sorgente di una applicazione in maniera programmata per fare le modifiche che un programmatore farebbe manualmente. L'obiettivo della tesi è quello di estendere il framework di semi-automatizzazione sviluppato dal gruppo TORSEC del Politecnico di Torino, e le aggiunte principali sono state l'implementazione di funzionalità di attestazione remota, di applicazione di tecnica di divisione del codice tra client e server e la rimozione del limite di una singola enclave per applicazione convertita.
Il framework SGX inoltre introduce un approccio al mondo dell'attestazione remota, offrendo la possibilità di poter implementare questo processo di verifica del software e della piattaforma, lavorando anche su tecniche di protezione del software che modificano l'architettura del software a favore di una che si affaccia maggiormente a curare l'aspetto della sicurezza che è ancora oggi considerato soltanto nelle

[5] https://www.intel.it/content/www/it/it/architecture-and-technology/software-guard-extensions.html

[6] https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/linux-overview.html

[7] https://www.intel.it/content/www/it/it/support/articles/000089271/intel-nuc.html

[8] https://docs.microsoft.com/en-us/azure/confidential-computing/confidential-containers-enclaves

[9] https://phoenixnap.com/security/confidential-computing

[10] https://gcore.com/blog/new-high-frequency-machines-and-virtual-machines-with-intel-sgx/

[11] https://www.beekeeperai.com/

[12] https://www.intel.com/content/www/us/en/architecture-and-technology/sgx-product-offerings/fortanix-confidential-ai.html

fasi finali del design di un prodotto. Tutte le funzionalità continuano ad essere disponibili per le applicazioni del programmatore scritte in C, usando anche alcune funzionalità del C++, e lo sviluppo del progetto è stato gestito su applicazioni che usassero questi linguaggi di programmazione, mentre il framework continua ad essere gestito da moduli Python. Un obiettivo ritenuto importante è stato anche lavorare su un sistema operativo Linux e il suo contesto di compilazione di sorgenti per generare un eseguibile finale: le librerie del Software Development Kit devono essere linkate con opzioni specifiche, e per questo è stato assicurato che non ci fosse soltanto una conversione semi-automatica del codice in un codice compatibile con le funzioni SGX, ma anche che venissero creati tutti i file necessari alla compilazione più semplificata possibile dei file che devono essere inclusi nel progetto.
Inizialmente la tesi illustrerà il background tecnico per poter capire il funzionamento del framework: il Capitolo 2 si occupa di illustrare SGX e come garantisce le proprietà di sicurezza dei dati, spiegando anche le vulnerabilità che possono essere sfruttate dagli attaccanti e quali tipi di attacco possono essere eseguiti, contestualizzando anche altre soluzioni proprietarie conosciute di TEE.Il Capitolo 3 invece descrive il processo di attestazione remota in generale e le opzioni che SGX implementa, introducendo i protocolli di comunicazione tra chi deve essere verificato e chi verifica, e anche quali sono le informazioni usate per verificare l'identità e lo stato di integrità di un sistema. Il Capitolo 4 introduce il framework SGX, spiegando quali sono le interazioni del programmatore con il framework, le fasi del processo di conversione, i tool esterni usati nel processo e le limitazioni attuali, mentre nel Capitolo 5 si entra nel dettaglio del funzionamento, illustrando l'architettura del framework e il workflow di conversione fase per fase. Il Capitolo 6 mostra i risultati sperimentali ottenuti e su cosa si è testato il processo di conversione. Infine, vengono illustrati nel Capitolo 7 altri framework di conversione semi-automatica di applicazioni aderenti a soluzioni TEE e applicazioni che implementano processi di attestazione remota.

Capitolo 2

Trusted execution environment in SGX

Nell'ambito della sicurezza informatica, uno degli obiettivi più comuni è la ricerca della garanzia di integrità dei dati che l'utente usa e conserva, evitando che siano danneggiati e ottenuti da altri utenti sprovvisti dei permessi di potervi accedere. Una perdita dei dati, una loro manomissione o qualsiasi azione che genera un disservizio del sistema può ripercuotersi sull'utente con danni di tipo economico o di tipo diverso, come un danno continuo se l'utente è impossibilitato ad usare quel servizio per un periodo di tempo non determinabile a priori, e impatti collaterali se i dati contengono informazioni sensibili la cui conoscenza potrebbe danneggiare la reputazione e l'identità di un individuo o gli interessi di compagnie di lavoro.
Per questo, si punta oltre alla sicurezza del prodotto con crittografia e altre tecniche, ad avere un ambiente di esecuzione attendibile, un Trusted Execution Environment (*TEE*): si tratta di un ambiente virtuale di qualsiasi dimensione, che sia un intero sistema informatico con hardware e software installato, o una singola applicazione, in cui l'esecuzione di operazioni e la conservazione di dati è svolta in maniera protetta e con la garanzia che le proprietà di sicurezza (confidenzialità, autenticità, integrità) siano rispettate.
Le proprietà di sicurezza non hanno una definizione univoca, ma ne esistono diverse, e ogni ente che si occupa della definizione di standard tecnologici e dei protocolli di sicurezza conia e aggiorna la definizione con il passare degli anni o in base al contesto di applicazione a cui si riferisce. Le seguenti definizioni sono prese dallo standard FIPS-200 [1]:

- autenticazione, è "il processo di verifica di un utente, un processo o un dispositivo, spesso da prerequisito per permettere l'accesso alle risorse di un sistema di informazioni" [1];

- confidenzialità, quando "si preservano restrizioni autorizzate sulle informazioni di accesso e di confidenzialità delle informazioni, inclusi i mezzi per proteggere la privacy di una persona e le informazioni personali" [1], o è "la confidenza che l'informazione non è svelata a persone, processi o dispositivi non autorizzati, e che copri i dati salvati, in fase di elaborazione e in trasferimento" [2];

- integrità, è "proteggere contro la modifica o la distruzione impropria di informazioni, inclusa la garanzia che sia rispettata il non-ripudio e l'autenticità dell'informazione" [1].

Il Trusted Execution Environment agisce sul sistema in cui viene eseguito creando partizioni dedicate: per partizione, si intende una area di memoria continua riservata ad una unità virtuale come un processo o un Sistema Operativo in modo tale che solo l'unità possa interagirvi. Inoltre, deve offrire sicurezza durante l'esecuzione delle funzionalità nello stesso modo in cui sono state progettate, e nel caso in cui il comportamento dell'ambiente fosse diverso da quello ideale a causa di un possibile attacco, riconoscere immediatamente che il sistema sia cambiato e reagire con eventuali risposte come il blocco dei dati sensibili. Le proprietà che ogni Trusted Execution Environment deve idealmente rispettare [3] sono:

1. separazione spaziale dei dati, deve essere impossibile l'interazione con i dati delle altre partizioni;
2. separazione temporale dei dati, facendo sanificazione delle risorse condivise per evitare leak di informazioni;
3. divieto di comunicazione tra le partizioni, se non in eccezioni specificate e controllate;
4. isolamento dei guasti, legati alle partizioni in cui avvengono e bloccando la propagazione dei danni al di fuori di esse.

Esistono varie tipologie di controllo dell'integrità dei componenti dell'ambiente, di natura hardware e di natura software, ma indipendentemente dall'architettura scelta, è necessario che a capo della gerarchia ci sia un componente altamente affidabile che gestisca tutte le operazioni di controllo e le operazioni più critiche. Questo componente è definito *Root Of Trust* [4], e deve necessariamente avere la fiducia assoluta di tutto il sistema poiché le funzioni di sicurezza che offre sono fondamentali per controllarne l'integrità: non è un componente che può essere analizzato da altri moduli che si trovano sotto nella gerarchia del sistema, dato che se fossero compromessi allora l'analisi sarebbe inutile. Per verificare lo stato di un componente, una Root Of Trust esegue una *misurazione*: per misurazione si intende un insieme di operazioni atte allo scopo di determinare il valore di un componente [5]. Ad esempio, la misurazione di un software è eseguita usando funzioni di hash sul binario e sui suoi contenuti, in modo tale da avere un valore finale che rappresenta lo stato del software al momento della misurazione. Un sistema deve avere tre tipi di Root Of Trust a seconda delle operazioni che ognuna deve fare:

- Root Of Trust for Measurement (RTM) per effettuare le operazioni di misura di un software;
- Root Of Trust for Storage (RTS) per la conservazione delle misure di un software effettuate;

- Root Of Trust for Reporting (RTS) per la comunicazione in maniera protetta dei contenuti della RTS.

Le soluzioni di tipo hardware solitamente usano un componente hardware economico installato sulla scheda madre come il *Trusted Platform Module*, che svolge le attività di Root of Trust for Storage e Root Of Trust for Reporting[1]. Il TPM è un chip resistente ad attacchi di manomissione prodotto secondo specifiche definite come standard tecnologico dalla Trusted Computing Group, ed è usato per conservare dati crittografici sensibili come password e chiavi di cifratura, permettendo di effettuare operazioni di attestazione usando l'algoritmo di hash SHA-1[6]. Il TPM infatti conserva in registri speciali chiamati Platform Configuration Register (PCR) le misurazioni hash effettuate sui software dalle unità RTM, e i valori contenuti all'interno sono aggiornati dalle misure prese dalle operazioni svolte, in modo tale che non si perdano i precedenti stati conservati ma che siano usati per calcolare la misura del software.[2]
Le soluzioni di tipo software invece possono convertire il componente Root of Trust in un processo software che gestisce tutti i sottoprocessi offrendo le operazioni delle tre unità Root Of Trust, vedesi il caso dell'ipervisore di una o più macchine virtuali [7].
I componenti che si occupano della misurazione della deviazione del dispositivo dal comportamento standard previsto, componenti necessari per misurare lo stato di affidabilità del sistema, appartengono al gruppo di risorse hardware e software che prende il nome di *Trusted Computing Base*(TCB). Diverse aziende hanno sviluppato modelli di TCB: oltre al già citato TPM, un altro esempio di TCB è z/OS di IBM, il sistema operativo proprietario delle workstation IBM (più approfondimenti nella Sezione 2.4.3) che possiede dei moduli software dedicati a proteggere i file che contengono le informazioni più cruciali (come i file di inizializzazione del sistema) da accessi e da modifiche di programmi reputati non fidati[3], usando ad esempio per ognuno di essi un Message Autentication Code (MAC)[4].

2.1 Introduzione a SGX

Software Guard eXtension (SGX) è una soluzione hardware proposta da Intel che permette ad una applicazione di poter operare in condizioni sicure: ogni piattaforma SGX supporta un nuovo set di istruzioni che opera sulla memoria, bloccando accessi diretti da parte del sistema e riservando solo al processore i permessi di scrittura e lettura delle pagine contenute all'interno. Queste aree sono definite anche

[1] https://www.intel.co.uk/content/www/uk/en/business/enterprise-computers/resources/trusted-platform-module.html

[2] Il processo di misurazione è svolto tipicamente al boot del sistema, in cui il precedente valore conservato viene azzerato, poi si misura il bootloader, il kernel del Sistema Operativo con il valore conservato, i moduli del kernel con il valore conservato, e così via.

[3] https://www.ibm.com/docs/en/aix/7.1?topic=conventions-protection-trusted-computing-base-tcb

[4] Hash crittografico calcolato su dati per verificare l'autenticità dei dati trasferiti o memorizzati.

enclavi, ed è garantito l'isolamento dal resto del sistema. Il processore compatibile con SGX inoltre è dotato di un chip unico, il *Memory Encryption Engine* (MEE), posizionato sul controller della memoria che si occupa della cifratura dei contenuti dell'enclave per fornire confidenzialità ai dati che sono conservati all'interno e controllare che la proprietà di integrità dei dati sia soddisfatta (vedesi il meccanismo di swap pagine 2.2).
La divisione in memoria riservata all'enclave e memoria esterna all'enclave permette di avere una parte *esposta*, o untrusted, dell'applicazione, e una parte *protetta*, o trusted, dove è garantito che non ci sia stata nessuna modifica esterna. Il processore considera tutto ciò che è esterno all'enclave come una 'Root Of Untrust', e l'accesso nella zona protetta che è contenuta all'interno dell'enclave non è lasciata libera come per il resto degli accessi in memoria: l'accesso deve essere bloccato, che sia fatto dal kernel del sistema operativo, da un ipervisore o da una periferica che chiede l'accesso in DMA. Si può pensare che la protezione arrivi dal MEE, ma questa ipotesi non è possibile perché il chip si trova lontano dal punto in cui il bus verifica che l'indirizzo a cui vuole fare accesso appartenga ad una pagina presente nelle cache, quindi i controlli sono svolti modificando il meccanismo di traduzione degli indirizzi del Page Miss Handler: il software fa accessi sfruttando sempre indirizzi virtuali, e le traduzioni di indirizzi che non rispettano i vincoli di SGX sono annullati prima che raggiungano la Translation Look-aside Buffer[5] [8].
L'isolamento del flusso di controllo del programma in due piani permette di gestire i dati in maniera sicura quando sono usati nelle enclavi, anche se il resto del sistema potrebbe essere stato modificato da un attacco informatico, per il principio di isolamento della memoria dal sistema operativo, dagli ipervisori delle VM e dal kernel.
L'uso di Intel SGX è recentemente aumentato in ambito Cloud Computing e nelle applicazioni in microservizi eseguiti in container sicuri: diventa facile in questo modo distribuire applicazioni isolate dal resto del sistema su cui girano, e fornire e elaborare dati in maniera sicura. Un ulteriore strumento per garantire l'affidabilità delle enclavi è quello dell'*attestazione*, realizzata generando strutture dette *Quote*, ossia raccolte di informazioni sui componenti e sui valori dell'enclave con operazioni di hashing e cifratura, che possono essere attestate da terze parti ritenute affidabili (attestazione remota) oppure possono essere giudicate da altre enclavi che si trovano sulla stessa macchina (attestazione locale). SGX è supportato dai processori Skylake appartenenti alle generazioni successive alla sesta, compatibili con il set di istruzioni che permette la gestione dell'enclave.

2.2 Descrizione dell'architettura SGX

L'architettura comune di un processore x86 divide la modalità di esecuzione in più livelli di privilegio, che prendono il nome di *ring* [9]. Per ogni livello, è possibile

[5]La TLB è una memoria cache che conserva gli indirizzi virtuali tradotti in indirizzi fisici in modo tale da velocizzare le operazioni e usare risultati già elaborati. Per memoria cache, si riferisce ad una memoria usata per conservare dati di un'altra memoria a cui si può fare accesso in tempi minori per velocizzare le operazioni.

accedere a più risorse del sistema, associando ad un valore più elevato nella scala gerarchica permessi più elevati che prima non erano disponibili, in modo tale da assicurarsi che sia implementato un meccanismo di sicurezza che limiti il numero di interazioni che possono portare accidentalmente o intenzionalmente ad un danno. Un sistema operativo installato su questa architettura genera così quattro ring di privilegio: il ring 3 è il livello più restrittivo, dove vengono eseguite le applicazioni che hanno accesso ad un gruppo ristretto di risorse hardware essenziali (una parte della memoria, alcuni dispositivi Input/Output), mentre il ring 0 è riservato al kernel che deve gestire la coesistenza dei thread, i driver software per abilitare i dispositivi esterni, la gestione delle eccezioni, e il resto dei servizi che compongono l'intero sistema operativo. Il livello di privilegio più alto quindi permette che il meccanismo di monitoraggio dei permessi sia più solido verso i tentativi di bypass dei permessi o di manomissione, e se l'architettura ad anelli è ben implementata, permette di rafforzare la separazione effettiva dei domini di esecuzione[6] e la separazione dei confini dei domini [9].
Il processore riserva una porzione esclusiva della memoria DRAM per le enclavi, sotto il nome di *Processor Reserved Memory* (PRM), isolata anche dal ring di protezione 0. L'attivazione della protezione della memoria delle enclavi deve essere attivata dal proprietario della macchina accedendo all'Unified Extensible Firmware Interface (UEFI). All'interno delle enclavi è contenuto il codice sensibile delle applicazioni a cui appartengono le enclavi e i dati privati che bisogna proteggere.
La memoria PRM è una memoria di tipo continua, con blocchi allocati consecutivamente uno dopo l'altro, per evitare frammentazione tra gli indirizzi di memoria riservati alle enclavi e quelli riservati al sistema. Ogni blocco salva le informazioni riguardanti le modalità di occupazione della memoria da parte dell'enclave in una struttura dedicata, l'*Enclave Page Cache* (EPC): essa viene divisa in pagine da 4 KB ciascuna, e le pagine sono assegnate alle diverse enclavi generate, secondo un meccanismo di mapping simile a quello delle pagine della memoria fisica del sistema.
Questa parte della memoria viene protetta con più meccanismi: i permessi di scrittura non sono assegnati al gestore della memoria ordinaria, che non potrà usarla in maniera indistinta, e le pagine vengono cifrate dal Memory Encryption Engine (MEE). Il MEE ha come obiettivi:

1. la confidenzialità per i dati che vengono scritti nella regione protetta della memoria DRAM;

2. l'integrità dei dati con prevenzione agli attacchi di tipo replay[7], per assicurarsi che i dati che sono letti dalla regione protetta della DRAM siano gli stessi dati scritti poc'anzi sulla stessa memoria dalla CPU.

[6]Per dominio di esecuzione, si intende il massimo insieme di dati e risorse, astratti nella forma di uno spazio virtuale di esecuzione, che un processo può potenzialmente usare.

[7]Attacco che viene fatto intercettando in un canale di comunicazione alcuni pacchetti di dati trasferiti da un utente all'altro, e inviandoli ripetitivamente, ritardando l'invio o inviarli in una sessione di comunicazione futura per generare disservizi nella comunicazione.

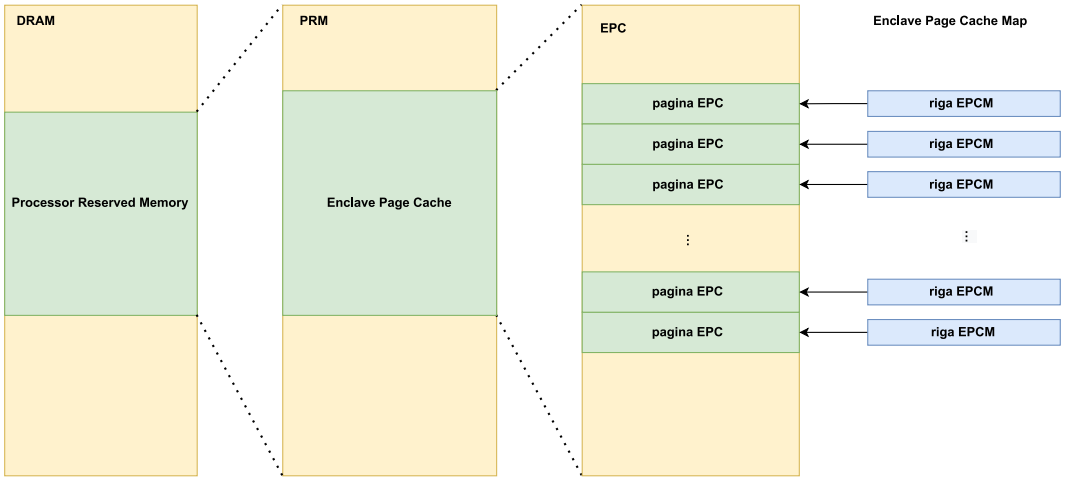


Figura 2.1: Struttura della memoria riservata per l'enclave

L'allocazione scelta per le pagine viene salvata dal processore in una Enclave Page Cache Map (EPCM) che possiede una riga per ogni pagina e i dati delle informazioni necessarie per risalire all'identità dell'enclave relativa a quella pagina.
Nella Tabella 2.1 viene mostrato il contenuto di una riga della EPCM. Oltre al meccanismo di validità della pagina e l'identificativo per riconoscere quali tipi di dati saranno contenuti, la struttura più importante per il processore è quella che permette di riconoscere l'enclave leggendo i metadati che essa ha memorizzato all'interno. Questa struttura dati prende il nome di *SGX Enclave Control Structure*

attributo	*dimensione (b)*	*descrizione*
validità pagina	1	indica se si può operare su quella pagina
tipo pagina	8	pt_reg per codice e dati, pt_secs se la pagina è una secs
sgx enclave control structure	4K	metadati usati dalla cpu per riconoscere l'enclave

Tabella 2.1: Schema della struttura interna di una riga della Enclave Page Cache Map

(SECS), ed è popolata solo quando l'attributo di Tipo di Pagina indica il valore PT_SECS. Il processore vieta categoricamente al codice di una enclave di accedere a questi dati, ed è l'unico che accede in modo tale da poter consultare le politiche di allocazione scelte per quella determinata pagina. Alcuni degli attributi sono:

- SIZE, la dimensione in Byte dell'enclave (necessariamente una potenza di 2);
- SSAFRAMESIZE, la dimensione in pagine di un frame SSA;
- BASEADDR, l'indirizzo virtuale di base dell'enclave;
- MRENCLAVE, misura del contenuto dell'enclave, viene aggiornata durante la sua generazione;
- MRSIGNER, registro del rispettivo valore che usa la chiave pubblica dell'enclave;

- ATTRIBUTES, attributi vari dell'enclave.

Quest'ultimo campo di 16 Bytes contiene altri metadati sull'enclave associata: informazioni come la possibilità dell'enclave di essere debuggata quando è caricata in memoria, un byte XFRM (eXtended Features Request Mask) per indicare il set di istruzioni usato dall'applicazione e le estensioni architetturali attivate dal compilatore allo scopo di generare il codice dell'enclave, e altre informazioni utili per strutturare la memoria virtuale dell'enclave.

L'enclave dispone di una memoria virtuale che deve essere divisa in una parte protetta e una non protetta o esposta; il sistema operativo, che non può accedere normalmente alle pagine SECS che sono nella PRM non essendo ritenuto affidabile dall'architettura SGX, ed essendo vulnerabile ad attacchi di tipo memory mapping (vedesi Sezione 2.3), viene indicato solo per fare la traduzione degli indirizzi fisici in virtuali protetti e non. Gli indirizzi virtuali protetti apparterranno ad un insieme di indirizzi contigui chiamato Enclave Linear Address Range (ELRANGE). In questa area, saranno contenuti i dati privati dell'enclave presenti nelle Enclave Page Cache e il codice sensibile. ELRANGE viene calcolato con un indirizzo base BASEADDR a cui è sommato un parametro SIZE che è preso dalla struttura di controllo SECS dell'enclave e segue la gestione della memoria dell'architettura x86 [8].
L'enclave viene caricata normalmente dall'applicazione sotto forma di libreria dinamica (formato ".dll" per sistemi Windows e ".so" per sistemi Unix). SGX permette sia il multithreading sia che all'interno di un enclave più thread possano concorrere, anche se solo chiamando le librerie appositamente scritte per il contesto di SGX, e bisogna abilitare l'enclave al multithreading assegnando ad ogni processore logico della CPU una Thread Control Structure (TCS): nella fase di definizione della configurazione dell'enclave, quando si assegnano valori agli attributi che definiscono le sue caratteristiche, bisognerà assegnare per ogni thread che si desidera avere una Thread Control Structure. Queste strutture sono salvate nelle pagine EPC, per renderle accessibili all'applicazione che girerà nel contesto di esecuzione SGX.
Con i TCS, nelle pagine EPC vengono anche salvate sequenze di State Save Area (SSA) per garantire una certa continuità a livello spaziale nella memoria: in caso di interruzioni per segnali e eccezioni che portano il sistema ad effettuare un cambio di contesto dall'enclave all'esterno del confine sicuro di esecuzione, il contesto della memoria in quel punto di esecuzione dell'applicazione è salvato rapidamente e in maniera sicura, e al ritorno la SSA associata all'enclave viene caricata ripristinando così il workflow dell'applicazione.

La Figura 2.2 indica il diagramma di stati di un enclave dal momento in cui è generata dal compilatore a quando l'applicazione termina, compiuto usando le istruzioni del set esclusivo a SGX. Ognuna di queste istruzioni non può operare contemporaneamente sulla stessa enclave, quindi è un processo sequenziale di stati. Quando una applicazione deve usare una enclave, chiede al Sistema Operativo di crearla con l'istruzione ECREATE che sceglie arbitrariamente una pagina EPC libera e la riserva per la nuova enclave, crea al di fuori della memoria protetta una struttura SECS con i valori iniziali e la copia all'interno della Enclave Page Cache. ECREATE inizializza anche l'attributo MRENCLAVE, che funge da log crittografico durante il processo di costruzione dell'enclave, e verifica che i valori

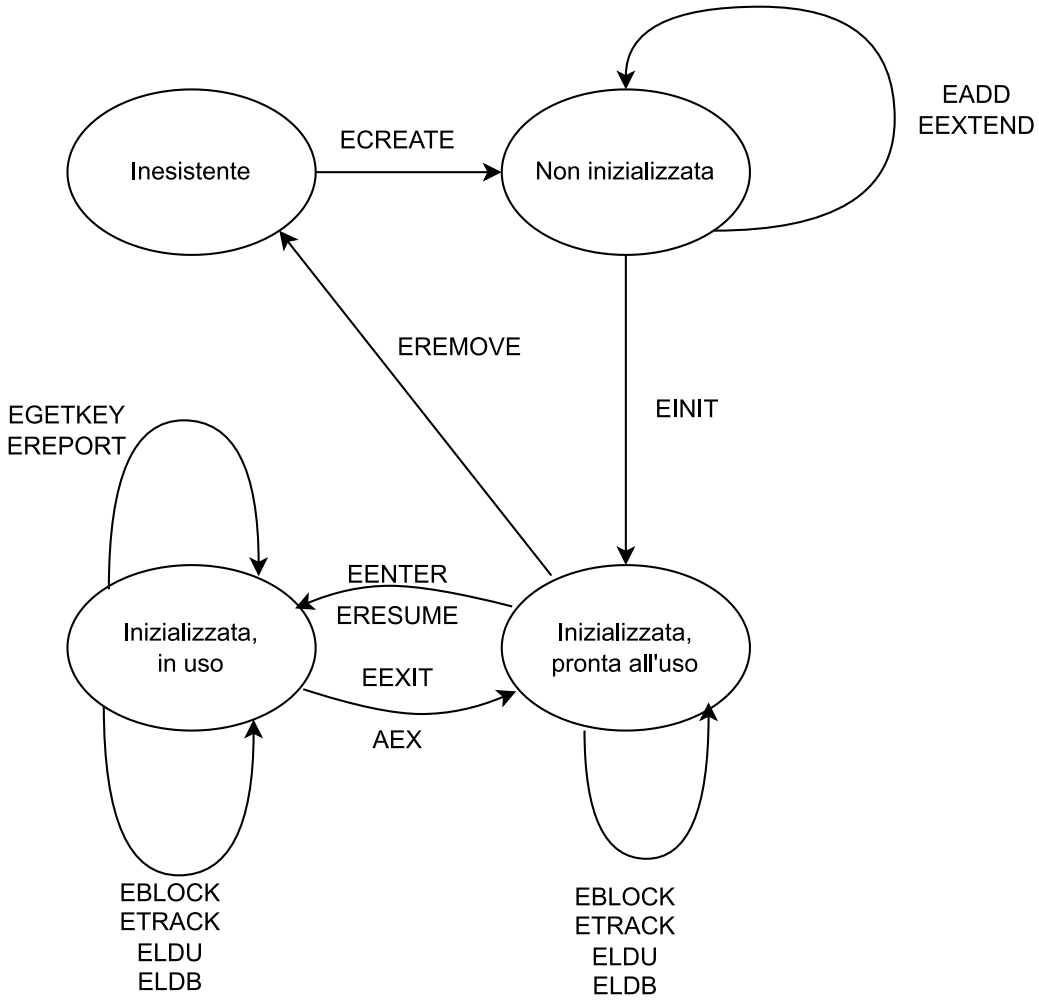


Figura 2.2: Ciclo di vita di una enclave generata.

dell'indirizzo di partenza (BASE) e di dimensione dell'enclave (SIZE) siano corretti[8]. L'istruzione segnala la presenza di errori in caso di informazioni delle proprietà di sicurezza dell'enclave non corretti (ad esempio, la dimensione non è una potenza di 2) e imposta l'attributo di INIT nella SECS dell'enclave per indicare che la memoria è stata assegnata ma è ancora vuota.
L'enclave è pronta per salvare informazioni all'interno della memoria dedicata, quindi vengono effettuate più operazioni di EADD per caricare le pagine di memoria che contengono codice e dati: le informazioni sono prese da pagine PAGEINFO, strutture che contengono informazioni come l'indirizzo lineare della pagina EPC, l'indirizzo virtuale della pagina non-EPC che contiene i dati da inserire, i permessi della pagina EPC allocata come i permessi di accesso (i bit R, W, X) e il tipo di pagina. Insieme al caricamento delle informazioni, verrà usata l'istruzione EEXTEND per aggiornare il valore della misura dell'enclave per le operazioni di attestazione.
A questo punto, il software incaricato della gestione dell'enclave usa una enclave speciale, la Launch Enclave (LE), che è autorizzata da Intel per lanciare tutte le

[8]SIZE dovrà essere una potenza di 2, BASE deve appartenere all'insieme di indirizzi costituito da ELRANGE.

enclavi che sono state create da autori diversi da lei e firmata con una chiave di Intel unica hardcodata all'interno del software SGX, per ottenere un EINIT Token Structure. Questo token è passato all'istruzione EINIT per segnalare la fine del processo, bloccando ogni futura operazione di EADD di aggiornamento di nuove pagine e permettendo all'enclave di essere eseguita con lo stesso livello di priorità di una applicazione, ossia il livello 3. Quando si vuole terminare l'enclave, si chiama l'istruzione EREMOVE per deallocare tutta la memoria composta dalle pagine che erano state caricate con EADD: si controlla che nessun processore logico stia eseguendo il codice che è contenuto nelle pagine dell'enclave, e vengono rimosse tutte le pagine terminando con quella che conteneva la struttura SECS dell'enclave. L'enclave dovrà quindi essere nuovamente generata seguendo il processo iniziale di allocazione e riempimento di pagine EPC in memoria.

Meccanismo di swap pagine nella Processor Reserved Memory

Se la memoria dell'enclave è piena e c'è bisogno di inserire nuove pagine, si attiva il processo di swap delle pagine EPC su disco fisso che le sposta sul supporto e le cifra prima di salvarle. Questo compito viene svolto dal Memory Encryption Engine (MEE), un chip dedicato a questa operazione incluso nella CPU, usando AES128 [10] CTR per cifrarle. Questo meccanismo fa parte del paging, una operazione svolta dal Sistema Operativo per liberare la memoria sfruttando la capacità più ampia del disco fisso per memorizzare le pagine che contengono informazioni non più utili per l'applicazione, a favore di nuove pagine.
SGX deve garantire la confidenzialità dei contenuti anche per queste pagine spostate sulla memoria non-PRM e offre la sua versione modificata di processo di paging: la pagina che deve essere rimossa viene cifrata dal Memory Encryption Engine con l'operazione EWB, che controlla non ci siano riferimenti nei processori logici a quella pagina e genera, oltre alla pagina EPC cifrata e salvata fuori dalla PRM in modo tale che il Sistema Operativo possa spostarla, anche un sottoinsieme dei suoi attributi presenti nella rispettiva struttura EPCM, più un MAC e un nonce[9].
I metadati non sono cifrati ma questo non è un problema a livello di sicurezza perché sono informazioni trascurabili e già specificate durante la chiamata di ECREATE. Dopo che la pagina è stata spostata, se l'enclave prova l'accesso al contenuto sarà generato un Page Fault e si uscirà dal contesto di esecuzione dell'enclave con l'istruzione Asynchronous Enclave EXit (AEX), che controlla che l'eccezione hardware lanciata sia dovuta ad un Page Fault e invoca quindi il gestore di eccezioni page fault del kernel.
Per ripristinare le pagine rimosse dal disco alla PRM, si invocherà ELDU/ELDB. Come prima cosa, verrà verificato il tag MAC[10] prodotto da EWB per assicurarsi l'autenticità della pagina, del nonce e dei metadati prodotti ed evitare attacchi attivi che usano page swapping e traduzione di indirizzi (vedere Sezione 2.3). Se non sono segnalati errori in questa verifica, allora i contenuti della pagina saranno

[9]Un numero casuale dal valore diverso per ogni generazione, può essere pseudo-randomico o un True Random Number Generator.

[10]Message Authentication Code, è un digest di lunghezza fissa a seconda dell'algoritmo usato che si usa per verificare se i dati associati sono stati modificati o se sono autentici.

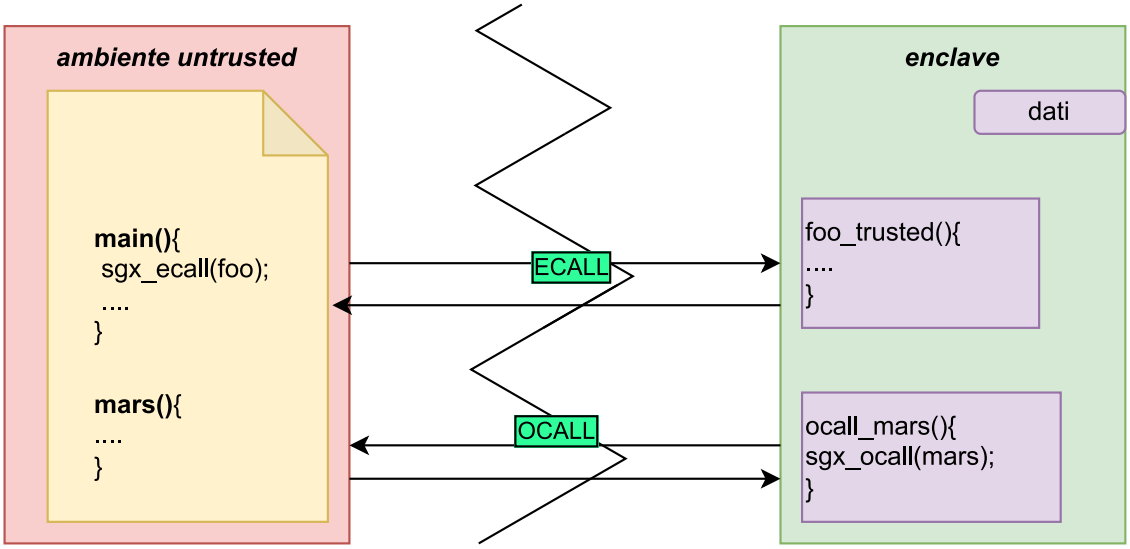


Figura 2.3: Schema di chiamate ECall e OCall in una applicazione con enclave

decifrati in una pagina EPC libera, ripristinando la rispettiva pagina che apparteneva alla EPCM. Ciò che differisce tra queste due istruzioni è la modifica di un bit all'interno della riga EPCM che indica se la pagina è bloccata e inaccessibile o no.

Flusso di esecuzione SGX

La Figura 2.3 mostra come l'applicazione è divisa dal modello di isolamento della memoria di SGX. L'enclave è completamente isolata dal resto del sistema dell'applicazione, e all'interno contiene i dati privati e il codice che sono stati inseriti durante la fase di compilazione dell'applicazione. La barriera di separazione dei contesti che si trova in mezzo è innalzata dal processore che blocca ogni accesso diretto.
L'unico modo per poter effettuare il cambio di ambiente di esecuzione è con chiamate specifiche: quando si deve invocare una funzione dentro una enclave, si parla di Enclave Call, o "ECall"; quando si è all'interno dell'enclave e si vogliono chiamare funzioni che sono invece localizzate all'esterno, si parla di Outside Call o "OCall". Le ECall sono funzioni che possono essere scritte in C o in C++, e usano una versione delle librerie standard riscritte da Intel per essere compatibili con l'enclave e che siano reputate sicure. All'interno, potranno solo essere eseguite le funzioni che sono state staticamente linkate con l'enclave, oppure funzioni che si trovano nella memoria non protetta. Una volta terminata l'esecuzione, si esce dal contesto e si ripristina lo stato relativo al momento prima che l'applicazione invocasse l'ECall, ritornando un valore che indica se le operazioni compiute con SGX sono state completate con successo o se ci sono stati determinati errori, codificati in interi.
L'invocazione delle funzioni fuori dalla memoria quando si è dentro l'enclave è possibile solo tramite l'uso di una OCall. Queste funzioni devono poter passare i parametri tramite un meccanismo di copia, dato che l'applicazione non potrà fare accessi nella memoria dell'enclave. Queste funzioni sono scritte in C o in C++ e linkate con un compilatore di codice C.

Sealing dei dati

SGX permette di effettuare una operazione di cifratura su dati usando una chiave personale, la *Chiave di Sealing* (o Sealing Key), una chiave segreta conosciuta solo dall'enclave del sistema. L'enclave ha un ciclo di vita che è limitato dalla durata dell'applicazione: quando l'applicazione termina l'esecuzione, o richiede la rimozione dell'enclave, tutte le risorse usate vengono cancellate dalla memoria, compresi i dati usati nel contesto protetto. L'operazione di cifratura, o Sealing, permette di preservare sul disco le informazioni presenti nell'enclave in un blob di dati cifrati, in modo tale che una nuova enclave possa caricarli e ripristinarli nella condizione in cui si trovavano al momento del salvataggio (l'operazione inversa è definita come *Unsealing*)[11].
Quando viene eseguita l'operazione di cifratura sui dati, SGX deve anche conoscere la politica di cifratura del software: devono essere definite delle condizioni in modo tale che quando queste siano rispettate, i dati possono essere decifrati e ripristinati. Le politiche implementate, e descritte nel manuale degli sviluppatori, sono le seguenti:

1. sealing sulla misura dell'enclave, richiederà una enclave che avrà lo stesso valore di misura MRENCLAVE;

2. sealing sulla misura dell'autore dell'enclave, la cui identità è memorizzata su uno dei registri della CPU e usa come fattore anche il Product ID dell'enclave.

La seconda politica permette a diverse enclavi definite dallo stesso autore di condividere dati, e anche di aggiornare una enclave senza problemi, dato che avrebbe un valore di misura MRENCLAVE diverso da quello precedente.
La chiave di Sealing segue lo standard NIST SP 800-108 [11] per usare come funzione AES-CMAC come funzione pseudo-casuale (Pseudorandom Function, PRF)[12].

2.3 Attacchi all'architettura SGX

Esistono diversi tipi di attacchi all'architettura SGX, che non riesce a proteggersi completamente da tentativi di attacchi esterni nonostante crei con le enclavi un suo spazio di esecuzione protetto isolato dal sistema operativo. Il sistema operativo può essere usato come punto di partenza per iniziare l'attacco sui dati protetti, dato che i sistemi operativi moderni sono molto complessi ed é difficile garantire la sicurezza e la stabilità di un sistema complesso: devono gestire infatti servizi critici come la gestione dei processi, il sistema di I/O e il controllo della memoria fisica e virtuale [12]. Qui verranno elencati e descritti brevemente alcuni tipi di attacchi principali e più diffusi, basandosi su studi più approfonditi [13] [12]:

[11] https://download.01.org/intel-sgx/linux-1.7/docs/Intel_SGX_Developer_Guide.pdf

[12] Una PRF è una funzione che genera un output usando un input di generazione casuale, o seed, e un dato come input, in modo tale che il risultato sia computazionalmente indistinguibile da un output veramente randomico [11].

- vulnerabilità delle enclavi;
- vulnerabilità hardware;
- attacchi di tipo fisico e di canali secondari;
- attacchi di timing sulle cache;
- attacchi di predizione di salti;

2.3.1 Vulnerabilità delle enclavi

Le enclavi sono vulnerabili agli attacchi definiti Return-oriented-Programming [14], exploit che permettono l'iniezione di codice malevole inserito in pacchetti di istruzioni macchina, chiamati *gadget*, tramite l'uso delle chiamate di ritorno da una funzione. In Figura 2.4 è presente un esempio di ROP: nel primo caso, assistiamo ad una sequenza di istruzioni che rimuove un valore dallo stack, mentre nel secondo si fa la somma tra due registri diversi, per ottenere un valore diverso come un nuovo indirizzo che si riferisce ad un nuovo dato o ad una nuova funzione. Ogni gadget deve terminare con l'istruzione *ret* per poter permettere di creare combinazioni di gadget che portano ad eseguire del codice non previsto dall'applicazione originale, e proprio per questo sono descritti come oggetti "*Return-oriented*". Durante le istru-

```
POP eax                    ADD eax,ebx
RET                        RET
```

Figura 2.4: Due gadget utilizzati per fare ROP

zioni del salto del codice infatti, si perde il controllo dello stack e possono crearsi situazioni anomale che possono essere sfruttate come base per attacchi.
Un esempio di attacco sviluppato sulle enclavi è il Dark-ROP [15], che disarma tutti i controlli di sicurezza SGX e sfrutta una vulnerabilità di corruzione della memoria nel software dell'enclave.
La vulnerabilità sfruttata dall'attacco è causata da una corruzione della memoria di tipo buffer overflow[13] che è possibile forzare con la giusta manipolazione dell'ambiente. Questa corruzione è prima di tutto ricercata usando programmi di fuzzing[14], e si nota che l'enclave, che viene eseguita al ring 3 dell'architettura, non

[13]Un buffer overflow è un errore che avviene quando si tenta di inserire una quantità di dati in un buffer che ha una dimensione massima inferiore alla quantità di dati trasferiti. Con questa vulnerabilità si possono inserire dati nello stack, corrompere la memoria e far crashare l'applicazione.

[14]Anche chiamati *fuzzer*, sono usati per studiare una applicazione: se i fuzzer più primitivi generano dati casuali, quelli più comunemente usati e complessi seguono strategie che ereditano la logica di American Fuzzy Lop, se non lo stesso AFL reperibile da https://lcamtuf.coredump.cx/afl/, e non generano input randomicamente ma partendo da un seed di partenza

può gestire le eccezioni che sono legate al processore per i permessi troppo bassi, e le restituisce al sistema operativo eseguendo una Asynchronous Enclave Exit. Il codice gira nonostante tutto in un ambiente protetto, ma attraverso l'uso di programmi oracolo[15] che leggono lo stato di esecuzione dell'enclave nonostante per la protezione dello schema TEE il codice e i dati siano nascosti. Dark-ROP riesce a lanciare una applicazione ombra per emulare una enclave configurata precedentemente e con un comportamento già deciso, e tramite le chiamate ROP usa l'enclave realmente esistente per compiere le operazioni di quella emulata, come la lettura delle chiavi crittografiche usate da SGX [15].
SGX-ROP [16] usa la programmazione orientata al ritorno per inserire dall'interno di una enclave modificata per inserire codice non autorizzato pericoloso. In particolare, i passi che segue sono i seguenti:

1. l'enclave chiama la primitiva *read* eseguendo un gadget per individuare potenziali gadget che possono essere presi dal codice e usati;

2. l'enclave trova zone di memoria su cui possono essere eseguite operazioni di scrittura, e con il codice di un gadget scrive all'interno di queste zone codice malevolo;

3. l'enclave costruisce una catena ROP dai gadget trovati per inserirli nello stack dell'applicazione ed esce dal contesto protetto, restituendo il flusso all'host;

4. l'applicazione arriva all'indirizzo in cui parte la catena ROP e esegue il codice malevolo iniettato, per poi ritornare al normale flusso di esecuzione e senza traccia del codice.

Un altro tipo di attacco sono gli Enclave Interface Attack: puntano alle interfacce di entrata e uscita nell'enclave (le ECall e le OCall) collezionando dati sulle chiamate effettuate (dimensioni dei dati di input ad esempio) e analizzano i risultati per svelare informazioni normalmente segrete sui dati usati [17]. Nella fase offline di questo attacco l'attaccante compie una ricerca, grazie anche a canali laterali che ascoltano le interfacce[16], sui pattern utilizzati dalle chiamate alle interfacce per ogni tipo di dato che viene usato in input, costruendo così un Dataset[17] che colleziona le informazioni raccolte. Viene quindi ideato, o scelto, un algoritmo che gestisca i dati raccolti. Nella fase online, si attacca l'applicazione SGX usando i dati elaborati dall'algoritmo per manipolare il comportamento dell'enclave.

[15]Programmi che hanno come obiettivo l'analisi e la predizione del comportamento di un software studiando l'esecuzione, senza tutti i dati a disposizione.

[16]Ad esempio, un firewall che usa SGX quando deve scrivere su un log che sono stati bloccati dati compie una OCALL per fare una write. Il canale ascolta quando un IP viene inserito in una blacklist inviando un pacchetto e forzando questa scrittura del log.

[17]Un insieme di dati raccolti che vengono forniti in input da un algoritmo di elaborazione dati. L'algoritmo usa questi dati nella fase di training per poter migliorare l'efficacia delle sue operazioni (machine learning).

2.3.2 Vulnerabilità hardware

Gli attacchi di tipo fisico sono tra quelli più difficili da effettuare, grazie al sistema di protezione di cifratura del Memory Encryption Engine che protegge i dati quando vengono spostati sulle memorie non protette del sistema. Non sono noti infatti attacchi riusciti che sfruttano il bus tapping alla DRAM e al suo bus, così come gli attacchi basati sull'exploit della potenza e del consumo dei componenti del sistema [13]. Attacchi di tipo fisico, che provano a eseguire operazioni di manomissione sulla componentistica di un sistema attraverso strumenti specifici, hanno un costo troppo alto a fronte di un guadagno che di fronte alla spesa da sostenere non è sostenibile. Considerando come anche le porte di debug siano disattivate, non si hanno ancora tracce di exploit conosciuti e dimostrati validi. L'unico attacco di cui si hanno conferme effettive è quello di tipo Denial of Service, ossia di *interruzione del servizio* del sistema, che prende il nome di RowHammer[18]: questo attacco consiste nel cambiare il valore fisico di un transistor elettronicamente per generare corruzione tra i dati contenuti in quelle cella della DRAM e squilibri nella lettura dell'integrità dell'enclave, causando così il blocco del processore SGX.
Dal bug di RowHammer, è stato ideato un attacco che prende il nome di SgxBomb [18] per diffondere il Denial of Service su macchine server che lanciano enclavi ricevute dai loro clienti in tempi brevi (283s con una DRAM dal tasso di refresh di 64ms), e in questo modo è possibile portare allo spegnimento della macchina inviandogli una enclave che deve eseguire funzioni di bit flipping per portare allo shutdown del sistema. Il bit flipping dell'attacco, basato su RowHammer, consiste in numerosi tentativi di accesso ad indirizzi casuali della memoria DRAM, ottenendo il cambio del valore contenuto in un bit della memoria: in questo modo si porta intenzionalmente il processore ad attivare la sua politica di "*drop-and-lock*" che blocca il sistema che non risponderà più agli interrupt che riceve, dovendo così spegnere o riavviare la macchina, causando un danno al servizio.

2.3.3 Attacchi di tipo canale laterale e fisico

SGX è completamente vulnerabile agli attacchi di tipo *side channel*, o attacchi con canale laterale, e non ha implementato meccanismi di protezione da questi attacchi, basandosi esclusivamente sulla protezione e l'isolamento all'interno dell'enclave isolata dal resto del sistema. Un attaccante può quindi usare l'intero sistema per attaccare l'applicazione SGX, e sfruttare così tutte le risorse per trovare potenziali punti deboli, considerando che è impossibile per l'applicazione non lasciare una traccia di uso sui componenti hardware che la eseguono.
Bisogna spiegare cosa è innanzitutto un attacco di tipo canale laterale. Per questa tecnica, si intende un tipo di attacco passivo, e spesso non invasivo nei confronti del sistema elettronico, che osserva il comportamento del dispositivo e delle sue proprietà fisiche, per estorcere un segreto che il sistema ha elaborato e cerca di nascondere. Alcuni esempi possono essere [19]:

- *attacchi sul timing* per scoprire le tempistiche di una applicazione;

[18] https://syssec.kaist.ac.kr/~yongdaek/courses/ee515/2017/Slides/Row_hammer.pdf

- *attacchi sul consumo di energia* per analizzare visivamente la differenza tra operazioni svolte e provare a capire quali operazioni sono state effettuate dai pattern[19];
- *attacchi di analisi elettromagnetica*, o EMA, in cui viene generata appositamente una interferenza elettromagnetica per disturbare e studiare un sistema che è in stato di esecuzione autonomamente.

L'attacco avviene parallelamente al sistema vittima, che è spesso ignaro di essere sotto attacco. Alcuni di questi attacchi sono anche facilmente replicabili dato il costo basso per compierli, anche se gli attacchi di tipo EMA sono sicuramente i più complessi perché necessitano di grandi consumi di energia per poter creare disturbi che abbiano un impatto sul sistema attaccato. Al contrario, gli attacchi sul consumo di energia di un sistema sono più facili, hanno un costo più basso e sono più efficaci (è dimostrato da [19] come è possibile scoprire che un sistema sta compiendo una operazione di cifraggio con DES).
I ricercatori di Microsoft Research, hanno scoperto un sottotipo di attacco con canale laterale che prende il nome di *attacco con canale controllato*: sfruttando un sistema operativo compromesso, vengono bloccati gli accessi a pagine di memoria che contengono variabili usate da una applicazione. Quando l'applicazione farà accesso ad un dato contenuto in una di queste pagine bloccate, verrà generato un errore di Page Fault (PF) che potrà essere intercettato da un attaccante in ascolto. Con diversi input ricevuti, cambieranno il numero di accessi sulle pagine, e sfruttando anche il principio di località delle pagine, è possibile ottenere silhouette fedeli di immagini JPEG decompresse da librerie che vengono eseguite dentro una enclave [20], ma anche ricostruire documenti testuali. La ricerca di queste informazioni deve scendere ad un livello di granularità della grandezza dei byte, dato che i pixel di una immagine e il carattere di una parola sono informazioni rappresentate su quell'ordine di grandezza, inferiore alla grandezza tipica usata da un Sistema Operativo nella gestione delle pagine di memoria di 4 KB [20], e si registrano tramite canali laterali tutti i page fault avvenuti durante le applicazioni che gestiscono i dati che vogliamo recuperare (ad esempio, per un documento di testo si può usare il correttore ortografico *Hunspell* mentre per una immagine la libreria standard di compressione di immagini *JPEG*): si identificano prima i PF relativi all'accesso a dati, poi si cerca una sequenza minima di PF relativi al codice (che avvengono prima dei dati che vengono cercati dalla memoria per essere usati) tipicamente composta da 2-3 elementi. Combinando le sequenze identificate, si può associare ad un PF di dati i PF del codice, e si analizzano così i byte. Maggiori saranno le iterazioni di ricerca dei PF eseguite, maggiore sarà la qualità delle informazioni ottenute.
In tutto ciò, SGX continua a garantire la confidenzialità dei dati, ma non riesce a nascondere l'uso della memoria e delle strutture usate all'infuori del contenuto memorizzato dalle intenzioni degli attaccanti malevoli. Un altro esempio di possibile attacco con canali laterali è SGX-Step, un framework Linux Open Source che permette ad un processo untrusted di configurare interrupt di timer Advanced Programmable Interrupt Controller (APIC) e tabelle di tracciamento pagine direttamente in user space [21]: il timer, che basa la sua frequenza su quella del

[19]Si dividono in Simple Power Analysis (SPA) e Differential Power Analsysis (DPA).

processore e su parametri di configurazione inseriti dall'utente, e configurabile anche tramite librerie di SGX-Step, parte quando l'applicazione esegue una istruzione dell'enclave che lo chiama e il processore esegue l'AEX per cambiare contesto. Il modulo kernel dell'attacco viene configurato sul sistema in modo tale che sia caricato ogni volta che il timer APIC sia attivato, e con il cambio di contesto viene caricata anche la libreria di SGX-Step che permette l'esecuzione di codice inserito dall'attaccante con i permessi del kernel. Prima che sia eseguita una ERESUME per ritornare nel contesto protetto, viene configurato nuovamente il timer per il successivo interrupt.

2.3.4 Attacchi sul timing delle cache

La memoria cache viene usata per accedere molto più velocemente a dati della memoria, salvati all'interno della cache che si trova molto più vicino al processore rispetto alla DRAM e evitando così inutili operazioni di accesso che rallenterebbero l'esecuzione. Per ogni accesso in memoria, il controller della cache controlla se all'interno vi è il dato richiesto, e per velocizzare ciò controlla solo gli ultimi bit dell'indirizzo del dato, per determinare la riga della tabella cache in cui si trova il dato (la cache è divisa in linee, e ad ogni linea possono corrispondere diversi indirizzi): considerando che le applicazioni SGX condividono la stessa memoria cache, si sfruttano le differenze temporali di accesso alla cache per lanciare attacchi [13]. Questo tipo di attacchi si divide in tre categorie: la prima categoria è nota come *Evict+Time*, la seconda *Flush+Reload* mentre l'ultima è conosciuta come *Prime+Probe*.
Evict+Time [22] è un tipo di attacco che manipola lo stato della cache (fase di Prime) prima di ogni operazione di cifratura e misura il tempo di esecuzione (fase Time), riuscendo grazie alla giusta configurazione della cache su cui si esegue l'operazione crittografica ad ottenere chiavi AES.
La tecnica Flush+Reload è stata proposta inizialmente da Gullasch[23] per attaccare AES su sistemi Linux sfruttando attacchi sulle cache: il funzionamento di questa tecnica si basa sulla rimozione di linee cache di livello 3 nel sistema tra un processo vittima e un processo malevolo. Le condizioni necessarie devono essere l'uso di memoria cache condivisa nel sistema, e l'uso dell'istruzione *clflush* per svuotare la cache. Nella prima fase dell'attacco, il processo malevolo esegue l'operazione di svuotamento su una linea dell'intera cache condivisa anche dal processo sano. Nella seconda fase di attesa, attende che il processo sano esegua una operazione di scrittura nella memoria. Nella terza fase, di *Reload*, il processo malevolo esegue nuovamente l'operazione di reflush sulla linea di memoria e misura la latenza di operazione: se il tempo trascorso è breve, il processo sano aveva fatto accesso alla riga della cache (cache hit), viceversa si ha un cache miss [24].
Gli attacchi che puntano a sfruttare il timing delle memorie cache sono disponibili per il livello di esecuzione applicativo equivalente al ring 3 secondo l'architettura Intel x86. La tecnica di attacco Prime+Probe utilizza un canale laterale per ritrovare informazioni sensibili dalle memorie cache condivise nel sistema, riempiendo prima la cache con dati casualmente generati che sono ignorati dall'algoritmo (fase di Priming) e dopo aver eseguito l'algoritmo per un intervallo di tempo sufficientemente

lungo, vengono analizzati gli accessi alle cache effettuati, provando a indovinare quali sono state le linee di cache caricate che comprendono i veri dati usati dall'applicazione (fase di Probe). Molti attacchi si basano su Prime+Probe, come CacheZoom [25] e CacheQuote [26], che dimostra come la garanzia di anonimato di EPID (descritto in Sezione 3.2.2) possa essere bucata tramite canali laterali che leggono la cache, superando così la protezione fornita dai meccanismi di attestazione remota di Intel SGX (affrontati nel Capitolo 3.2).
Usando la debolezza agli attacchi di tipo canale laterale di SGX, fonti hanno dimostrato come sia possibile estrarre una intera chiave ottenuta con l'algoritmo crittografico *Rivest-Shamir-Adleman* (RSA) da 2048 bit durante l'operazione di decifrazione usando una enclave modificata [27], ma la lista di algoritmi attaccabili con questa tipologia di attacco include anche altri nomi molto conosciuti come AES [28].
Un altro attacco basato su questo tipo di vulnerabilità è CacheOut [29]: studiato dall'Università di Adelaide e dall'Università del Michigan, permette all'attaccante di selezionare quali dati e quali linee di cache (di livello 1) possono subire involontariamente un leak che li copia nei buffer della CPU, fuori dai confini di separazione che SGX costruisce, dove sarà possibile recuperarli facilmente. Nonostante Intel abbia pubblicato dei fix per mitigare al problema, l'attacco non è stato completamente mitigato, ed è anche possibile estrarre dall'enclave chiavi AES e chiavi RSA private, nonché invalidare totalmente l'integrità e l'autenticità del processo di attestazione remota svolto con EPID (per approfondimenti, Capitolo 3.4)

2.3.5 Attacchi sulla predizione dei salti

Gli attacchi di predizione dei salti (o *branch*) si concentrano sulle unità di predizione dei salti Branch Target Buffer (BTB), Pattern History Table (PHT) e Return Stack Buffer (RSB). Queste unità servono per la previsione dei salti di un programma: per non consumare colpi di clock, l'unità di predizione dei salti (composta da una BTB e da un predittore che è nella sua forma più semplice composto da un solo bit) esegue speculativamente le istruzioni. Questo significa che non analizzerà le istruzioni e i dati per prevedere esattamente quando dovrà eseguire un salto, ma le eseguirà e deciderà in tempo di esecuzione se ci sarà un salto da compiere o no: in caso questa predizione sia sbagliata, avviene una *misprediction* che porta a cancellare le istruzioni su cui si è speculato fino a quel momento e a ripetere il branch eseguito, perdendo così tempo di esecuzione.
La BTB è una memoria cache per memorizzare per ogni indirizzo le rispettive informazioni di predizione di salto[20], la PHT è una tabella usata dai predittori a 2 bit che ha lo scopo di indicare quali sono gli ultimi salti avvenuti nel programma e il RSB è un buffer hardware dal numero di elementi variabile in cui sono conservati gli indirizzi di ritorno usati dalla istruzione di basso livello *call*: quando la keyword viene usata, l'indirizzo viene inserito nello stack dal processore con una operazione di push, e verrà ottenuto con una operazione di pop quando ci sarà una istruzione RET [24].

[20]Come il Program Counter relativo all'istruzione e il valore del Program Counter target che corrisponde all'indirizzo su cui il processore dovrà saltare.

Un programma può eseguire due tipi di salti: quelli condizionati, determinati da costrutti condizionali come l'if-then-else, e quelli non condizionali, indicando direttamente la posizione della memoria in cui il puntatore all'istruzione corrente deve puntare. Gli algoritmi crittografici come RSA e i suoi derivati usano spesso istruzioni condizionali che in base agli operandi determinano di eseguire una istruzione che non si trova in una posizione sequenziale ma in un altro punto della memoria. Nonostante si possano usare predittori più potenti che usano due bit o un approccio misto, effettuare attacchi può essere molto efficace per rallentare l'applicazione e manipolare il suo flusso di esecuzione. Predicendo un salto, o forzando l'applicazione a fare un salto, è possibile scoprire informazioni sulle variabili su cui sono fatti test per condizionare il flusso: per questo sono disponibili attacchi che riescono, conoscendo l'architettura del predittore o scoprendola mediante reverse engineering, a scoprire i parametri usati in un algoritmo crittografico. [30]
Alcuni attacchi degni di nota contro una applicazione TEE, e compatibili anche per Intel SGX, sono BranchScope, che usa al posto del branch target buffer un predittore di branch direzionale [31] e Bluethunder, che usa un predittore direzionale a due livelli, con prestazioni migliorate di 52 volte in confronto a BranchScope [32].

2.4 Tecniche alternative di creazione TEE

L'interesse nello sviluppo di ambienti di esecuzione il più possibile sicuri è stato sempre forte e sufficiente affinché venissero prodotte nuove soluzioni più sicure e aggiornate in modo per soddisfare le richieste. Verranno presentate in questa sezione più da vicino alcune delle tecnologie più famose di Intel, ARM e IBM.

2.4.1 Intel Trusted eXecution Technology

Intel SGX è l'erede della precedente soluzione conosciuta come Trusted eXecution Technology (TXT)[21], molto più vicina alla soluzione di TPM. L'obiettivo di Intel TXT è fornire protezione contro gli attacchi sul sistema operativo, sul firmware BIOS e qualsiasi componente che è compreso nella misura del software della macchina usando un chip come nuovo Root Of Trust nell'hardware per formare una catena di misure Root Of Trust. Il confronto con le misure previste dal TPM è fatto con la creazione di un Measured Launch Environment (MLE) che associa un identificatore unico per ogni componente e possiede metodi per bloccare il lancio del codice del componente che non supera il confronto con il valore previsto. In questo modo ogni software che verrà eseguito sarà stato sicuramente controllato di essere autentico e non modificato, anche se le modifiche durante l'esecuzione del software non vengono più rilevate fino al nuovo lancio.
Quando il sistema esegue il bootstrap la prima misura è eseguita su un modulo autenticato e firmato digitalmente dal produttore del chip Intel chiamato *Authenticated Code Module* (ACM). L'ACM calcola successivamente le misure hash usando

[21] https://www.intel.it/content/www/it/it/support/articles/000025873/processors.html

SHA-1, o SHA-2 [33] se il TPM installato lo supporta, sul BIOS e sui contenuti presenti nella sequenza di PCR. I valori riportati sono poi confrontati con i valori contenuti nei Platform Configuration Registers inseriti dal TPM, che svolge anche il ruolo di Root Of Trust for Measurement. Se viene misurato un valore diverso da quello del TPM, l'hardware TXT blocca i contenuti che erano in procinto di essere lanciati, assicurandosi così che il sistema rispettasse le condizioni di integrità durante l'inizio dell'esecuzione. Esiste anche un meccanismo di misura di alcuni componenti del sistema che parte lanciando un comando dedicato e svolto su PCR dinamici che vengono resettati per poter compiere nuove misure.
Questa verifica verrà fatta partendo da un modulo hardware ACM diverso dal precedente, ACM SINIT, conosciuto anche come Dynamic Root Of Trust for Measurement: la differenza da una normale RTM, conosciuta come *statica*, è che invece di misurare tutti i moduli del sistema partendo dal boot, si misurano solo i componenti quando c'è il bisogno di lanciare un evento sicuro che richieda l'affidabilità del sistema. Il vantaggio principale di questa soluzione è la dimensione della TCB ridotta rispetto alla dimensione di una TCB usata da una RTM statica. Questo meccanismo è anche chiamato Late Launch, perché può essere lanciato dopo che sono state svolte le misure iniziali fondamentali[22].
Il processore che supporta Intel TXT deve anche essere abilitato all'uso della tecnologia Intel Virtualization Technology (VT-X): serve per creare un hardware virtuale, come una macchina virtuale, che sia condiviso dai vari processi di un sistema, e che possa dinamicamente scalare le risorse allocate e concesse in base alle operazioni eseguite[23]. Uno svantaggio di TXT è rappresentato quindi dall'aumento di hardware che appartiene alla TCB. L'hardware che deve essere incluso infatti comprende:

- il TPM;
- i chip di supporto Intel TXT e VT-X;
- il processore Intel e i chip;
- i moduli ACM che contengono le sezioni di codice dedicato al preboot di TXT.

2.4.2 ARM TrustZone

La principale soluzione TEE proposta da ARM è ARM TrustZone[24]: come con SGX, il protocollo di Trustzone divide il sistema in due parti isolate, in cui le risorse da proteggere, il firmware e le funzionalità di sicurezza sono inserite in un ambiente sicuro su cui gira un container, *Rich Execution Environment* (REE) o Normal World, e il resto del sistema, che possiede uno spazio di memoria non protetto, con il nome *Trusted Execution Environment* (TEE) o Trusted World, su cui

[22] https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf

[23] https://www.intel.it/content/www/it/it/virtualization/virtualization-technology/intel-virtualization-technology.html

[24] https://www.arm.com/technologies/trustzone-for-cortex-m

gireranno le applicazioni che non necessitano di permessi speciali[25].

La Figura 2.5 mostra l'implementazione dei due ambienti di esecuzione. Ogni

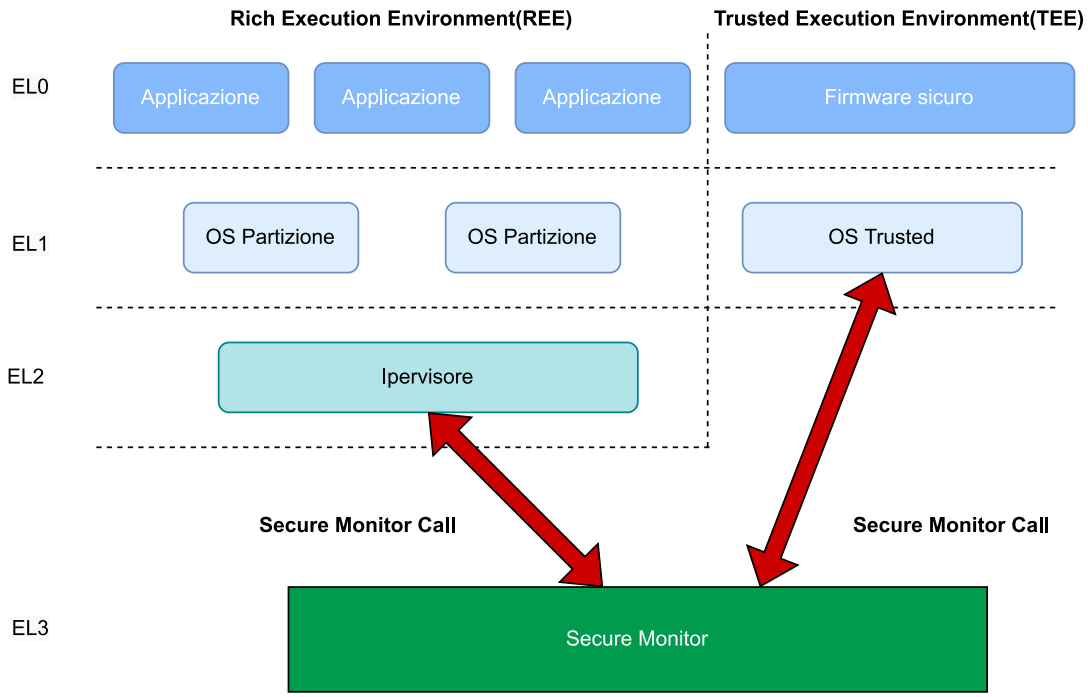


Figura 2.5: Schema del sistema di isolamento dei contesti in TrustZone

ambiente è diviso in una struttura di gerarchia e distribuzione dei privilegi all'interno del sistema basato su *livelli di Eccezioni* (Exception Level, EL). Ogni livello è associato ad un numero che parte dal valore minimo di 0 fino al massimo di 3: più si scende di numero, minori saranno i privilegi concessi al processo eseguito in quel livello. Al livello più alto, EL3, si trovano le routine di sicurezza del sistema, il firmware più vicino al linguaggio macchina e il Secure Monitor. Mentre è possibile che un processo che si trova ad un EL più alto possa accedere a risorse di livelli di privilegio inferiori, non è possibile l'opposto. Si può notare che il Trusted World normalmente non ha un livello di privilegio EL2, essendo opzionale per molti dispositivi come ARM-v8A e supportato solo da alcune versioni in poi[26].

Il processore permette di fare un cambio del contesto di esecuzione con "Security Extensions" per entrare nel codice protetto e uscire da esso. L'istruzione chiamata per effettuare questo passaggio è Secure Monitor Call (SMC): essa invoca il Secure Monitor, un agente ponte tra le due partizioni che si occupa della transizione da una parte all'altra del sistema, della gestione delle eccezioni e dello stato dell'applicazione precedente al cambio che deve essere salvato momentaneamente per essere poi ripristinato.

Una differenza rispetto a SGX è che se la soluzione Intel opera sulla stessa memoria riservandone una zona per l'esecuzione sensibile, TrustZone cerca di usare idealmente due hardware diversi, ma l'implementazione invece comprende una modalità di

[25] https://developer.arm.com/documentation/102418/0101/What-is-TrustZone-

[26] https://developer.arm.com/documentation/102418/0101/TrustZone-in-the-processor

operazione del processo che è definita da un Secure Bit. Il Secure Bit indicherà qual è l'ambiente in cui il processo è in esecuzione, sebbene rimanga a monte la limitazione che solo i processi critici e affidabili potranno essere eseguiti nell'hardware sicuro, mentre nel REE potranno invece trovarvi posto le applicazioni, i manager di VM, senza doversi preoccupare che l'uno abbia visibilità sull'altro.
TrustZone è disponibile per un ampia gamma di dispositivi, specialmente per i dispositivi mobile su cui i processori ARM sono diffusi, e per i dispositivi Internet of Things (IoT).

2.4.3 IBM Secure Service Container

IBM Secure Service Container è una tecnologia di container che sfrutta hardware, software e firmware disponibile per famiglie di sistemi server IBM Z[27] e IBM LinuxONE[28] che permette di lanciare in maniera rapida e protetta software eseguibile in maniera sicura, sfruttando una combinazione di hardware, software e firmware. All'interno di questi mainframe, il principale Sistema Operativo installato è z/OS, mentre la tecnologia di virtualizzazione, nonchè ipervisore delle macchine virtuali, usata è tipicamente z/VM.
Al suo interno sfrutta un hardware integrato specializzato nelle operazioni crittografiche, che permette di eseguire operazioni in maniera rapida, con un True Random Number Generator[29] a disposizione e con gli standard FIPS 140-2 [35] rispettati dal modulo PCIe di Sicurezza IBM Crypto Express 6S: in caso di scoperta di tampering dei dati conservati o di fluttuazione dei valori, il modulo di sicurezza che rispetta il livello 4 di FIPS 140-2 [35] sovrascrive i dati memorizzati con dati dal valore nullo (equivalenti a byte consecutivi dal valore zero) per proteggere gli altri dati e le chiavi.
Lo standard FIPS 140-2 ha come tema la definizione dei requisiti di sicurezza di un modulo crittografico utilizzato in un sistema di sicurezza. Sono definiti quattro livelli, in base alla maggior protezione fornita dal prodotto:

1. il primo livello è il livello di sicurezza più basso, dai requisiti semplici e senza moduli di crittografia hardware aggiuntivi, e usano almeno un algoritmo o funzione presente nella lista di quelli approvati;

2. il secondo livello aumenta la sicurezza con il requisito di meccanismi antimanomissione sui moduli crittografici, meccanismi che devono essere fisicamente danneggiati per poter accedere ai dati crittografici come chiavi e password conservate, denominati *Parametri Critici di Sicurezza* (CSP, Critical Security Parameter), e include un minimo sistema di autenticazione dei ruoli.

[27] https://www.ibm.com/it-infrastructure/z

[28] https://www.ibm.com/it-infrastructure/linuxone

[29] Un TRNG garantisce una vera randomicità nella generazione del numero usando dati appartenenti al mondo fisico, e non tramite un modello matematico che potrebbe essere prevedibile tramite uno studio (in questo caso si parla di Pseudo RNG). Alcuni dati fisici presi come seed possono essere l'intensità della luce rilevata, la carica elettrica contenuta in un transistor dell'hardware TRNG e la frequenza di oscillazione in un circuito [34, Cap. 3].

Il software, firmware compreso, richiede l'approvazione dei requisiti di Common Criteria (CC) con un livello di Profilo di Protezione almeno pari a EAL2 (Evaluation Assurance Level);

3. il terzo livello, oltre ai requisiti precedenti, richiede meccanismi di rilevazione di accesso fisico ai moduli crittografici protetti, un modulo che verifichi l'autorizzazione di un utente e del suo ruolo e che tutti i dati, gestiti su porte e interfacce separate da quelle normali, siano trasferiti in forma cifrata. Gli standard da rispettare sono CC con EAL3 e in aggiunta una Informal Target of Evaluation[30] (TOE) Security Policy Model;

4. l'ultimo livello fornisce la protezione più completa, proteggendo completamente il modulo crittografico al punto tale da rilevare con alta fiducia ogni tipo di intromissione nel modulo e azzerare tutti i CSP conservati. Rilevano anche manomissioni ambientali come cambi di corrente consumata e di temperatura, anticipando potenziali attacchi di tipo fisico (descritti in Sezione 2.3.3). Il software deve rispettare un il livello CC con Profilo di Protezione EAL4.

Opzionalmente può anche sfruttare una workstation di tipo Trusted Key Entry[31], una feature di supporto che combina specifici hardware e soluzioni software per conservare localmente e remotamente i dati crittografici sensibili come chiavi e password, rafforzando la sicurezza con moduli crittografici abilitati ad operazioni di crittografia hardware, e aiutando la gestione dei domini di più sistemi con messaggi broadcast, ossia messaggi diretti a tutti gli host che appartengono a quel dominio.
La soluzione IBM permette di usare, all'interno della memoria del dispositivo su cui l'utente ha la possibilità di installare diverse partizioni (dal contenuto rappresentato in Figura 2.6), una nuova partizione di tipo Secure Service Container che include al suo interno il sistema operativo, i meccanismi di sicurezza, le interfacce e le funzionalità necessarie per installare dispositivi virtuali che siano eseguiti in maniera isolata dalle altre partizioni. La politica di isolamento scelta comprende anche il gruppo di amministratori del sistema, che possono compiere scelte sui dati e sulle risorse non sicure che possono portare a criticità di sicurezza o malfunzionamenti, e si cerca dunque di limitare la loro operatività e padronanza sul sistema. La connessione alla macchina virtuale tramite SSH è una operazione vietata per la pericolosità di permettere un controllo ulteriore sulle risorse da proteggere, e le applicazioni vengono lanciate solo dopo che sono state misurate per controllarne l'integrità e giudicate sicure, e non operano direttamente sui dati. Infatti, sia le applicazioni sia gli amministratori non hanno accesso diretto sulle risorse cifrate, ma devono chiamare API REST a loro volta cifrate ogni volta per poter trasferire dati da una zona di memoria all'altra, cifrandoli ulteriormente con la tecnologia Transport Layer Session (TLS).
TLS [36] è un protocollo crittografico di comunicazione dati ideato usando come

[30]Secondo CC, è un sistema informatico o una parte del sistema e la relativa documentazione soggetta a valutazione di sicurezza.

[31]https://www.ibm.com/docs/en/zos/2.4.0?topic=services-trusted-key-entry-tke-support

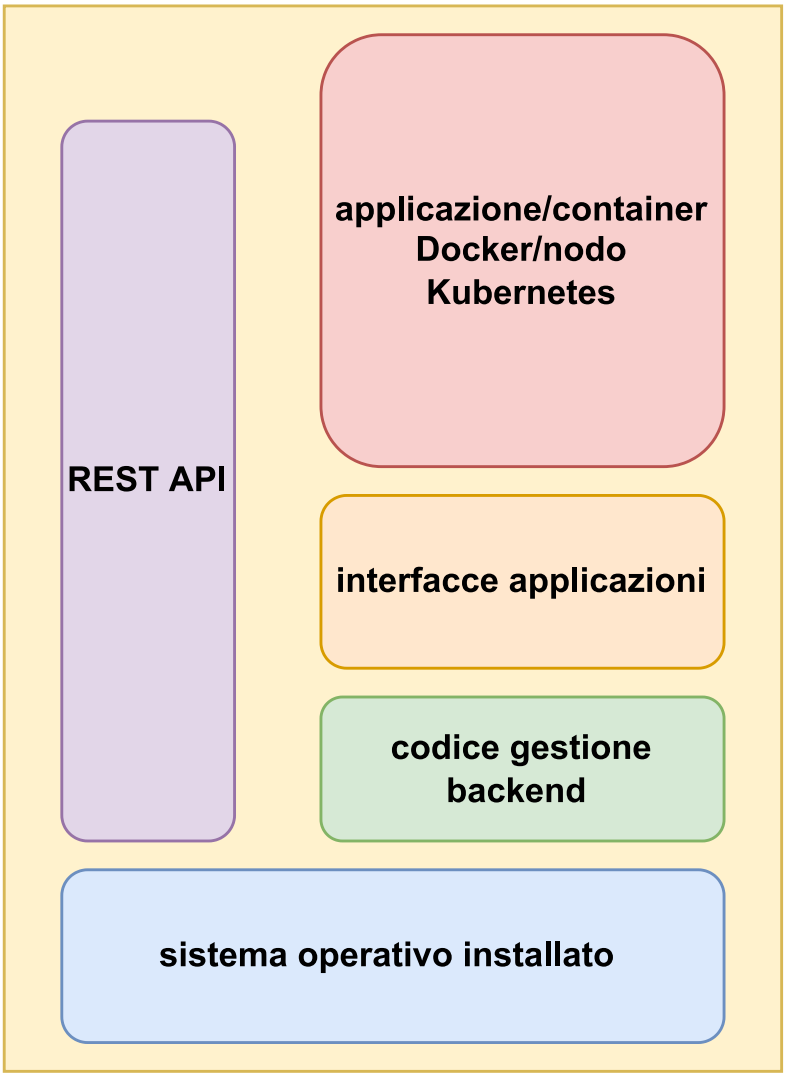


Figura 2.6: Schema della struttura interna di una partizione Secure Service Container

base Secure Sockets Layer (SSL) [37] , ed è composto da un protocollo di creazione del canale di comunicazione, o *handshake*, seguito da un protocollo di trasporto dei dati, o *record*: è implementabile con il protocollo TCP ed è comunemente usato per molte applicazioni web come mail e la navigazione Internet con HTTP.
Il dispositivo potrà dunque compiere in questo ambiente protetto che garantisce alle operazioni più importanti la protezione dal tampering dei dati: al boot del sistema solo le applicazioni autorizzate potranno essere lanciate, la misurazione dei componenti è stata effettuata per controllarne l'integrità.
L'hardware usato è resistente ad attacchi di tipo Row Hammer (introdotto nella Sezione 2.3.2), funzionanti su macchine SGX, grazie all'architettura della memoria secondo lo stile Redundant Array of Independent Memory, in cui si usano tecniche di striping e mirroring della memoria in maniera simile all'architettura Redundant

Array of Independent Disks (RAID)[32]. Le tecniche di striping e di mirroring sono usate, anche in combinazione per avere risultati più efficienti e robusti, nei meccanismi di recupero dati in caso di danni fisici al disco rigido usato, e consistono nella suddivisione dei dati in blocchi della stessa dimensione ripetutamente inseriti in uno o più dischi, a seconda della tecnica usata: nel livello 0 i blocchi dei dati sono presenti solo su un disco, divisi esattamente come se fossero "strisce" di memoria, mentre nel livello 1 si usano due dischi per avere copie ridondanti degli stessi blocchi di dati, in modo tale che se è fisicamente impossibile accedere al blocco presente in un disco si recupera l'altra copia presente nel supporto non guasto.
Gli algoritmi supportati dai server cloud che supportano SSC sono numerosi, e tra quelli disponibili della suite OpenSSL ci sono ECDH [10]/RSA [38] per le funzioni di Key Exchange e Autenticazione, AES-GCM-128/256 [10] per la cifratura, SHA-256/SHA-384 [39] per il controllo di integrità con un Message Authentication Code. Inoltre, sfrutta la versione GNU Private Guard di OpenPGP secondo lo standard RFC4880 [40] per quanto riguarda la cifratura dei dati.[33]

[32]Tra le differenze principali che occorrono tra l'architettura RAIM e quella RAID, la prima è integrata su celle di memoria DRAM e ha tecnologie di ripristino di dati da errori hardware (faults) più specifiche e efficienti, al contrario del RAID che ha algoritmi di riparazione più semplici il controllo di parità dei bit e le memorie Error Correction Code (ECC).

[33]https://www.ibm.com/docs/en/sscfcp/1.1.0?topic=overview-security-secure-service-container-cloud-private

Capitolo 3

Attestazione Remota in SGX

L'attestazione è una funzionalità che verifica l'identità della macchina su cui gira il software e il suo stato di integrità per confermare l'affidabilità dal punto di vista della sicurezza. L'hardware o il software che sfrutta questa funzionalità guadagna la fiducia di un produttore o di una entità remota, inviando prove che dimostrano che il software sia in esecuzione all'interno di una enclave SGX e su un sistema aggiornato all'ultima versione di sicurezza[1]. Il processo di attestazione avviene secondo un protocollo, implementabile in più schemi (Sezione 3.2.2), che formalizza lo scambio di messaggi tra un agente, o *sfidante*, e un *verificatore* assunto sicuro per confermare l'identità dello sfidante. L'utente che deve provare la sua identità deve fornire al verificatore una credenziale da analizzare, sotto la forma di firma digitale operata su un hash delle misure dei dati del dispositivo.
Secure Guard eXtensions offre la possibilità di effettuare due tipi di attestazione[2]: l'attestazione *locale* e l'attestazione *remota*. Quando si effettua l'attestazione locale, due o più enclavi possono attestarsi tra di loro, tramite una enclave speciale che Intel predispone all'interno del sistema; nell'attestazione remota la verifica della prova crittografica, generata dall'enclave, può essere inviata tramite un canale di trasporto dati sicuro che permette una comunicazione ad un verificatore remoto che garantisca affidabilità. Con questi meccanismi di attestazione, è possibile generare un gruppo di enclavi che sono eseguite simultaneamente e che scambiano i propri segreti, in maniera indipendente e slegata, con i fornitori di servizi remoti, dimostrando di rispettare le condizioni poste dalle politiche di sicurezza scelte per usufruire dei loro servizi[3].

---

[1] https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html

[2] https://courses.cs.ut.ee/MTAT.07.022/2017_spring/uploads/Main/hiie-report-s16-17.pdf

[3] https://www.intel.com/content/www/us/en/developer/articles/technical/innovative-technology-for-cpu-based-attestation-and-sealing.html

3.1 Descrizione

Il processo di attestazione è fondato sulla generazione della credenziale crittografica della macchina, una prova di integrità che è verificata localmente o remotamente. È importante scegliere nel design del processo cosa si vuole attestare, ossia quali sono i valori da misurare per poter confermare che il software venga eseguito correttamente, e quali algoritmi di hash usare per effettuare le misurazioni su di essi. All'attestazione, può essere aggiunto un processo di "Shared Key Agreement" come Diffie-Hellman Key Exchange (DHKE) [41] per proteggere le comunicazioni tra le due entità con una chiave segreta condivisa, e aggiungere proprietà di sicurezza utili per la protezione dei dati.

Il protocollo Diffie-Hellman tra due parti è implementato in questo modo:

1. i due utenti, Alice e Bob, scelgono un numero generatore g e un numero primo p tale che $g < p$. "p" e "g" saranno parametri pubblici poiché noti a tutti e due gli interlocutori.

2. Alice sceglie un numero segreto casuale "a" e manda a Bob il prodotto A = $g^a mod p$.

3. specularmente, Bob sceglie un numero segreto casuale "b" e manda ad Alice il prodotto B = $g^b mod p$.

4. Alice può quindi calcolare la chiave simmetrica $K_a = B^a = (g^b mod p)^a = g^{ab} mod p$

5. Bob otterrà la stessa chiave simmetrica $K_b = A^b = (g^a mod p)^b = g^{ab} mod p$

La Figura 3.1 mostra il modello generico di una firma di attestazione prodotta dal dispositivo che vuole essere attestato. Viene generata una firma di attestazione, composta da un messaggio che comprende al suo interno i valori da leggere dal verificatore. Insieme ai dati di attestazione, verrà eseguita una operazione di firma digitale con la chiave di attestazione del processore della macchina sul software eseguito.
Il verificatore sarà in possesso della chiave di attestazione grazie al processo di Key Exchange che è stato completato con l'enclave: richiede al software eseguito dentro l'enclave di iniziare un processo di generazione di chiavi con un messaggio iniziale come un numero casuale g^a scelto secondo le linee del protocollo adottato e riceve dal software con cui sta comunicando un secondo messaggio con un secondo numero casuale g^b e la richiesta di concatenare i due segreti generati e calcolare l'hash H = h(g^a $||g^b$)[4].
Il verificatore ha a disposizione la chiave pubblica necessaria per verificare i dati firmati dall'enclave con la chiave privata del processore della macchina secondo lo schema di un protocollo crittografico asimmetrico: la chiave pubblica è presente nel certificato rilasciato dal produttore del processore, e permette di verificare tutti i dati che sono stati firmati con la chiave privata di attestazione della macchina,

[4]I caratteri "||" indicano la concatenazione di due o più dati.

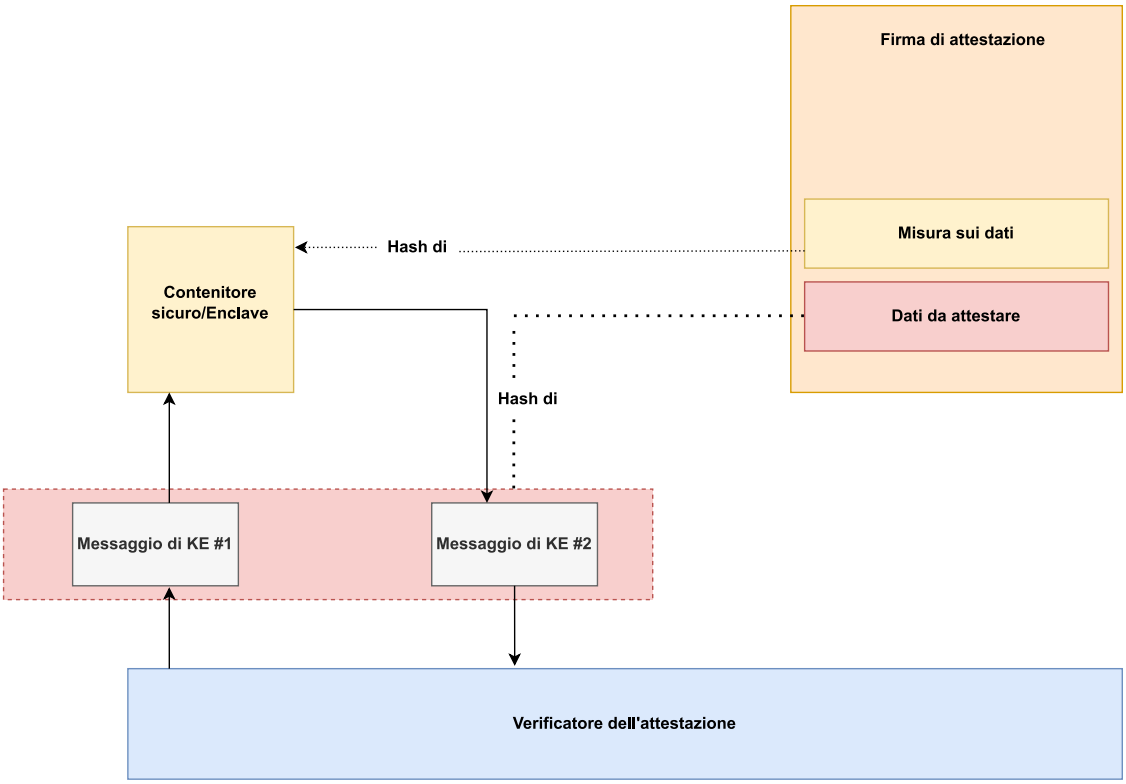


Figura 3.1: Schema di creazione di una credenziale di attestazione

salvata non in un certificato ma all'interno di un hardware resistente ai tentativi di manomissione e usata soltanto per attestare.
Con questo design, sia il verificatore che la macchina dell'enclave non devono temere attacchi, dato che il certificato del produttore non è disponibile pubblicamente ma deve essere consegnato appena la macchina è stata prodotta e rilasciata sul mercato. Il verificatore deve essere quindi un gestore fidato e che sia autorizzato dal produttore del sistema ad avere i certificati che permettono la verifica dei dati. Deve inoltre conoscere il formato dei dati che riceve, e deve poter eseguire le stesse operazioni che l'enclave ha svolto per poter ottenere gli stessi risultati. Si vedrà nella Sezione 3.2.4 che nel processo di attestazione remota di SGX è Intel a fare da verificatore.

3.2 Attestazione all'interno dell'ambiente SGX

Prima di essere attestata, l'enclave deve svolgere la sua fase di misurazione iniziale. Ogni enclave ha al suo interno una struttura SIGNATURE STRUCTURE abbreviata in *SIGSTRUCT*. La SIGSTRUCT è un certificato che viene firmato dall'ambiente nella fase di inizializzazione, e contiene tutte le informazioni necessarie alla misura del suo stato in quel momento. La sua esistenza è fondamentale visto che l'istruzione EINIT controlla che ci sia una SIGSTRUCT all'interno dell'enclave che chiede di essere lanciata, e deve leggerne i contenuti per poter caricare i dati corretti nella SGX Enclave Control Structure.
Alcuni dei suoi attributi sono i seguenti:

- ENCLAVEHASH, un hash SHA-256 (rispettando le norme del FIPS PUB 180-4 [33]) del codice e dei dati caricati nell'enclave secondo l'ordine corretto, comprendendo nella misura anche le proprietà delle pagine di memoria associate[5];
- VENDOR, per indicare se l'enclave è stata definita da Intel (0x00008086H) o se è stata creata da una applicazione dello sviluppatore (0x00000000H);
- ISVSVN, il Security Version Number assegnato all'enclave dal suo sviluppatore per indicare una specifica versione dell'enclave con rispettive proprietà di sicurezza ad un identificativo numerico;
- ISVPRODID, è un altro identificativo numerico del prodotto che può essere usato per l'attestazione associandolo allo stesso valore di identità dell'autore dell'enclave;
- la SIGNATURE, una firma RSA eseguita sull'header e sul corpo del certificato SIGSTRUCT.

I parametri scelti per la signature, salvati all'interno del certificato, sono quattro interi di 3072 bit, e devono essere: un modulo intero da 3072 bit, un esponente pubblico scelto pari a 3, e la firma deve seguire lo schema EMSA-PKCS1-v1.5 [42] e PKCS#1 v2.1 [42] per il formato del valore "DigestInfo"[6]. Il motivo per cui l'esponente scelto è pari a 3, seguendo il design di SGX, è una ottimizzazione della velocità dell'operazione di verifica della firma RSA con quel valore invece di altri: più è piccolo l'esponente, più è possibile usare uno schema alternativo per la verifica firme che compie una pre-computazione dei quozienti, e scegliere il valore "3" permette infatti di poter evitare di svolgere operazioni di divisione e di evitare totalmente numeri negativi nell'algoritmo di verifica [8] [43]. SGX segue lo schema della sezione precedente come base per il processo di attestazione modificando alcuni passaggi. Il sistema non può accedere direttamente alla chiave di attestazione privata, ma può farlo soltanto attraverso una enclave con permessi speciali autorizzata da Intel. Questa enclave prende il nome di *Quoting Enclave* ed è incaricata di trasformare i report di attestazione prodotti dalle enclavi locali in report che possono essere attestabili remotamente da altre piattaforme. Intel offre due implementazioni diverse per l'attestazione remota dell'enclave: una implementazione che usa l'algoritmo crittografico a chiave pubblica Elliptic Curve Digital Signature Algorithm (ECDSA) e Intel Enhanced Privacy ID (EPID) (descritto nella Sezione 3.2.2).

3.2.1 Fase di Attestazione Locale

Con questo processo, l'obiettivo è quello di generare un report contenente le informazioni di una enclave che sia verificato da un'altra enclave del sistema con la

[5] Proprietà definite nella SGX Enclave Control Structure.

[6] https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3d-part-4-manual.html

stessa CPU, e quindi con gli stessi segreti necessari alla generazione delle chiavi. Il modello usato da Intel per generare le chiavi e dei segreti non è illustrato in questa tesi, ma è possibile esaminarlo parzialmente fino a dove i segreti industriali di Intel non proteggono i dettagli [8]
La generazione del report è ottenuta usando una istruzione EREPORT, che è chiamabile solo all'interno dell'enclave e del suo contesto di esecuzione protetto, e immette nella memoria un report crittografico sui contenuti dell'enclave in cui è stata chiamata e una misura hash di essa.

istruzione	*EAX*	*RBX*	*RCX*	*RDX*
EREPORT	In	TARGETINFO (In, EA)	REPORTDATA (In, EA)	OUTPUTDATA (In, EA)

Tabella 3.1: Schema dell'istruzione di EREPORT dal Manuale Sviluppatore Intel

In Tabella 3.1 viene rappresentato l'encoding scelto per l'istruzione EREPORT. L'istruzione usa tre parametri input, a cui accede tramite il rispettivo Effective Address della pagina salvata in cache:

- TARGETINFO, la struttura contenente le informazioni dell'enclave target che chiama l'istruzione;
- REPORTDATA, un campo del report che deve essere popolato con i dati dell'enclave che si vuole misurare, e che popolerà l'output finale;
- OUTPUTDATA, la prova crittografica completa che sarà salvata nell'indirizzo indicato.

Il processore come primo passo controlla che questi parametri siano presenti all'interno dell'enclave, necessariamente allineati, altrimenti l'istruzione deve lanciare un errore. A questo punto genera la chiave di report conoscendo l'identità dell'enclave che dovrà verificare il report, leggendo il valore della sua misura MRENCLAVE dall'indirizzo contenuto in RBX, con l'istruzione EGETKEY. L'istruzione quindi usa l'indirizzo in RCX per accedere ad una struttura di report appartenente all'enclave in cui l'istruzione è stata chiamata: vengono aggiunti dati appartenenti alla SECS dell'enclave per completare la struttura, e ne effettua subito una misura hash usando AES-CMAC[7] e una chiave di report da 16 Byte ottenuta con l'istruzione EGETKEY.
Il risultato dell'hash è inserito nei contenuti del report, e il suo indirizzo è salvato in RDX. L'applicazione copia la nuova credenziale crittografica trasferendola all'enclave destinataria incaricata dell'attestazione. Essa usa l'istruzione EGETKEY per ottenere la stessa chiave simmetrica usata per il calcolo del MAC del report: con la chiave verifica l'autenticità del MAC e controlla l'identità dell'enclave iniziale. Eventualmente, può anche ripetere lo stesso processo di generazione della credenziale crittografica per avere una mutua attestazione con l'enclave sorgente.

[7] https://www.intel.com/content/www/us/en/developer/articles/technical/innovative-technology-for-cpu-based-attestation-and-sealing.html

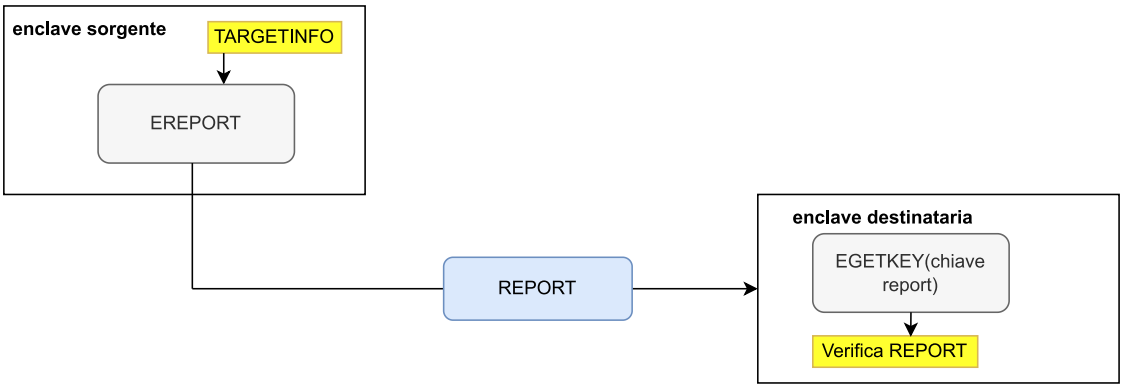


Figura 3.2: Schema del flusso dell'attestazione locale

In questo modo, è possibile che più di due enclavi possano attestarsi tra di loro, e poter iniziare uno scambio di dati da una zona di memoria riservata ad un altra dovendo soltanto proteggere i dati durante il trasferimento.

3.2.2 Opzioni di Attestazione Remota in SGX

Il report di identità dell'enclave dell'applicazione a questo punto deve essere trasformato in una struttura che possa essere verificata da una macchina diversa da quella in cui è stato generato.
La piattaforma di SGX offre due opzioni di attestazione remota:

- attestazione con Intel Enhanced Privacy ID;
- attestazione con Intel Elliptic Curve Digital Signature Algorithm.

Le due implementazioni sono diverse, e la scelta è lasciata allo sviluppatore che conosce anche l'ambiente di sviluppo e applicazione in cui dovrà aggiungere il processo di attestazione. Per lo sviluppo di questo progetto, è stato scelto di usare Enhanced Privacy ID come meccanismo di base per l'attestazione dell'enclave, mentre ECDSA sarà descritto brevemente nelle Sezione 3.2.3.

Intel Enhanced Privacy ID

Lo schema di Enhanced Privacy ID è stato implementato dal 2008 su dispositivi Intel per oltre 2,4 Miliardi di dispositivi. Nasce dallo schema Direct Anonymous Attestation [44], appartenente alla famiglia dei protocolli Sigma [45][8], ed usano il protocollo di "Key Agreement" Diffie-Hellman per offrire "perfect forward secrecy" [46], una proprietà crittografica che garantisce, grazie ad un continuo processo

[8]Un protocollo appartenente alla famiglia dei protocolli Sigma, detto anche Protocollo Sigma, richiede uno sfidante e un verificatore che si scambiano tre messaggi, il primo di impegno alla sfida, il secondo con la sfida inviata e il terzo di risposta. L'obiettivo da raggiungere è quello di ottenere una chiave condivisa per proteggere i segreti scambiati.

di generazione e ricambio dei materiali crittografici usati, che il furto o la compromissione di una chiave crittografica non comporta nessun rischio di compromissione dei dati scambiati nel futuro dell'applicazione, ma che solo i dati che hanno usato quella specifica chiave compromessa siano a rischio. L'implementazione di SGX tramite EPID però evolve il modello tradizionale di Public Key Infrastructure e il concetto di chiavi crittografiche asimmetriche. La prima aggiunta è che cambia l'identità univoca che creava una corrispondenza 1:1 tra chiave privata e chiave pubblica. Da una chiave pubblica adesso possono essere generate milioni di chiavi private che possono essere assegnate a diversi dispositivi Intel EPID, e la chiave pubblica servirà a garantire che la transazione crittografica firmata con la chiave privata sia valida. In questo modo, non serve essere a conoscenza dell'identità del dispositivo che ha usato quella chiave o del sistema su cui la chiave è stata usata: questo garantisce la privacy e il totale anonimato del proprietario della chiave privata. Al verificatore EPID serve solo sapere che il dispositivo che ha attestato appartiene allo stesso gruppo di dispositivi, e che ognuno di questi sarà dotato di una personale chiave privata che è stata generata dalla stessa chiave pubblica del gruppo. Ci spostiamo così dal concetto di verifica di identità del sistema crittografico al concetto di autenticità del sistema crittografico. Questo permette la personalizzazione della politica di generazione delle chiavi: ad esempio, una serie di dispositivi IoT finalizzati allo scopo di raccogliere dati appartengono allo stesso gruppo con un unica chiave pubblica, e se si vorranno inserire nuovi dispositivi di raccolta dati, viene generata per ciascuno di essi una chiave privata partendo dalla stessa chiave pubblica appartenente al gruppo di quel tipo di dispositivi; se invece hanno una finalità diversa, verranno generate le chiavi usando la chiave pubblica di un gruppo che include dispositivi dalla funzionalità diversa.
Un altro vantaggio è l'inserimento di meccanismi di protezione dal possibile sfruttamento della privacy dei dispositivi, sotto forma di *revoca delle chiavi*. La revoca delle chiavi è utile nei casi in cui è avvenuta la compromissione di dispositivi che eseguono firme di attestazione a causa di attacchi informatici, nel caso in cui un agente malevolo è entrato in possesso di una chiave privata o nel caso in cui si è giunti al termine della scadenza della licenza di uso di Intel di un dispositivo che compie un servizio commerciale.
Per questo motivo, sono stati pensati tre diversi sistemi di revoca EPID: revoca della chiave privata, revoca della firma digitale e revoca di un gruppo EPID. La revoca della chiave privata è effettuata sul singolo componente asimmetrico generato dalla chiave pubblica di quel gruppo di dispositivi EPID, ed è possibile quando si è a conoscenza della chiave privata da revocare. Il gruppo EPID di appartenenza della chiave rimane così ancora valido e gli altri dispositivi che hanno ricevuto una chiave del gruppo non dovranno preoccuparsi di refresh dei sistemi usati.
In caso non si fosse a conoscenza della chiave, si può revocare la firma digitale effettuata da essa, inserendola nel secondo tipo di gruppo di revoca riservato alle firme. Quando un dispositivo riceverà richiesta di attestazione con una firma digitale presente in questa blacklist di revoca, il verificatore EPID potrà immediatamente riconoscere che la transazione crittografica è stata richiesta da un dispositivo segnalato e bloccarne il normale uso.
Infine, la flessibilità dello schema di revoca EPID permette anche di includere un intero gruppo di dispositivi associati ad una chiave pubblica se il suo amministratore deve terminare l'uso dei dispositivi o deve sospendere temporaneamente

l'attività per aggiornamenti di sicurezza. Per scoprire più approfonditamente i ruoli dei gestori dei dispositivi EPID e i dati scambiati nelle inizializzazioni, è possibile consultare le documentazioni ufficiali Intel[9].

3.2.3 Attestazione con Elliptic Curve Digital Signature Algorithm

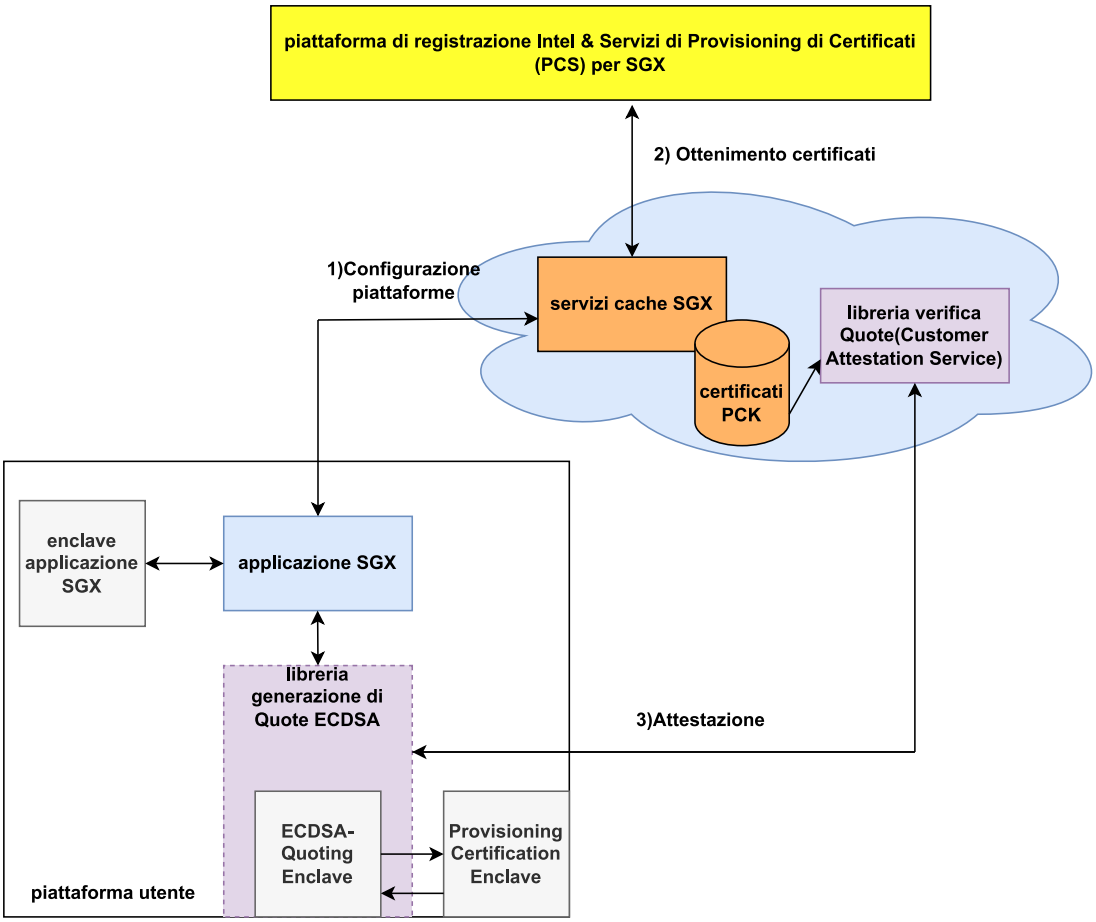


Figura 3.3: Schema ad alto livello dell'Attestazione con ECDSA in una struttura cloud

Elliptic Curve Digital Signature Algorithm (*ECDSA*) è un algoritmo crittografico alternativo all'algoritmo DSA ma basato sull'uso delle curve ellittiche, e viene usato dal servizio di SGX Data Center Attestation Primitives (conosciuto anche come SGX *DCAP*). DCAP è una estensione fornita da Intel che permette al fornitore di un servizio di gestire un proprio servizio di verificazione delle quote delle enclavi, senza che sia contattato l'Intel Attestation Service, ideato attorno alle metriche del protocollo di EPID. DCAP usa anche un modulo chiamato Flexible Launch Control per fare in modo che sia l'utente dell'applicazione a verificare direttamente

[9] https://www.intel.com/content/www/us/en/developer/articles/technical/intel-enhanced-privacy-id-epid-security-technology.html

quali sono le enclavi inizializzate e lanciate nel sistema, sostituendo la Launch Enclave autorizzata da Intel, e di controllare l'identità delle enclavi che hanno fatto accesso al Platform Provisioning Identifier, un valore unico della piattaforma e della Provisioning Enclave usato insieme al Security Version Number della TCB per identificare la piattaforma che richiede il certificato PCK da Intel[10]. La Figura 3.3 schematizza i passi svolti nell'attestazione ECDSA:

- configurazione della macchina, bisogna iscriversi alla Piattaforma di Registrazione Intel[11] per ottenere le chiavi necessarie per l'uso delle API che permettono di recuperare le informazioni della Trusted Computing Base, dei certificati PCK, delle liste di revoca e informazioni sulla Quoting Enclave e la Provisioning Certification Enclave;

- ottenimento dei certificati firmati da Intel, da salvare su una serie di memorie di caching a disposizione del servizio;

- attestazione, usando le librerie di generazione di quote per inviare la prova crittografica e le librerie di verifica delle quote per controllare che la firma sia corretta.

Il data center usa *PCK Cert ID Retrieval Tools* per recuperare le informazioni necessarie all'attestazione contattando l'Intel SGX Provisioning Certificate Server (PCS), e le salva nel suo sistema di caching. Il salvataggio sulla memoria del dispositivo è necessario dato che il sistema cloud resterà isolato dal resto della rete per motivi di sicurezza e integrità, e nessuna chiamata al PCS viene eseguita in runtime. La verifica è indipendente da un verificatore remoto come avveniva per lo schema EPID, e il gestore del data center può implementare le API messe a disposizione dalla Libreria di Verifica delle quote integrandole nel codice dei suoi server.
L'applicazione SGX dovrà usare la Quoting Enclave[12] basata su ECDSA e la Provisioning Certification Enclave.
La Provisioning Certification Enclave è l'enclave architetturale incaricata di firmare i report della QE, con la sua Chiave di Provisioning privata.
Il server del data center verifica la quote ricevuta usando le informazioni di attestazioni presenti nel servizio di Caching del Data Center, e se la firma risulta valida, esamina gli attributi contenuti all'interno della quote che compongono la Trusted Computing Base del sistema. In base alle politiche di sicurezza che sono scelte per il servizio, si deciderà se l'identità dell'enclave è ritenuta affidabile o no. La quote usata in questo schema di attestazione avrà una struttura diversa da quella usata per lo schema EPID.

[10] https://www.intel.com/content/dam/develop/external/us/en/documents/intel-sgx-support-for-third-party-attestation-801017.pdf

[11] Come l'Intel Attestation Service, è una portale web di Intel, la cui registrazione è eseguita tramite una richiesta REST di tipo POST.

[12] Essendo una enclave speciale, non può essere chiamata direttamente ma tramite una opzione In-Process e una Out-of-Process.

3.2.4 Fase di Attestazione Remota

L'enclave che inizia il processo di attestazione genera un report localmente attestabile. Terminata questa fase, il sistema deve lanciare la Quoting Enclave: riceve i report usati nella fase precedente e richiede le chiavi di Report usando l'istruzione EGETKEY. La Quoting Enclave verifica che i report ricevuti siano autentici, e si appresta a chiamare la precedente istruzione per poter ottenere una Provisioning Seal Key. Il suo scopo è la decrittazione della chiave di Attestazione ricevuta in forma cifrata dalla Provisioning Enclave, una enclave speciale creata da Intel, per assicurarsi che questo segreto fondamentale non venga scoperto da nessun ascoltatore.
La Quoting Enclave ottiene la chiave di Attestazione solo quando è lanciata per la prima volta dal sistema: nella fase di inizializzazione la Provisioning Enclave contatta l'Intel Provisioning Service (IPS), un servizio di Intel disponibile tramite Internet sviluppato per l'attestazione basata su schema EPID, in modo che possa ricevere la sua chiave EPID privata che corrisponderà alla chiave di Attestazione[13].

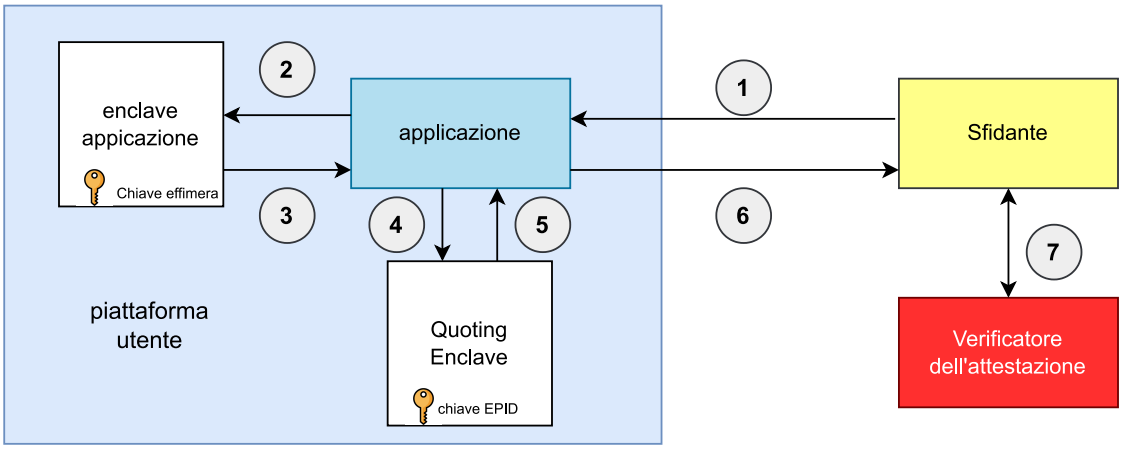


Figura 3.4: Schema ad alto livello del processo di Attestazione Remota in SGX

La Figura 3.4 mostra come SGX implementa l'attestazione remota di una enclave. Il meccanismo usato viene eseguito da tre agenti:

- la *Piattaforma Utente* SGX, reputata non sicura, autrice dell'enclave da attestare;
- lo *Sfidante*, una macchina remota che vuole conferma dell'identità dell'enclave;
- il *Verificatore*, la Root Of Trust di Intel, l'Intel Attestation Service.

Lo sfidante può essere un servizio offerto da un provider, in contatto con la macchina SGX che in un caso di applicazione reale può essere il client che scambia informazioni con la sua applicazione (ad esempio, un player di streaming che richiede

[13]Non tutti i dettagli della creazione delle chiavi sono noti pubblicamente, per scelta tecnica di Intel, come i segreti memorizzati tramite operazioni di scrittura non volatile (flashing) nei chip del processore usati nelle generazione delle chiavi chiamate con EGETKEY.

pacchetti di dati video e deve confermare ad ogni intervallo temporale l'identità del client alla piattaforma che fornisce il contenuto audiovisivo). Non è necessario che il fornitore del servizio remoto sia una macchina compatibile con SGX, perché come vedremo a breve le operazioni eseguite per la verifica dei dati non necessitano di un ambiente di esecuzione isolato e di segreti appartenenti ad altre macchine. Le fasi dell'attestazione sono le seguenti:

1. lo sfidante, durante l'esecuzione dell'applicazione, chiede una prova crittografica che confermi che l'enclave sia affidabile. Manda perciò all'utente una richiesta di attestazione, eventualmente come nonce per evitare attacchi di tipo replay;

2. l'applicazione chiede all'enclave quindi di creare la sua credenziale crittografica, entrando nel contesto di esecuzione protetto per generare il report e passare la chiave pubblica all'agente remoto per poter anche creare una sessione di Diffie-Hellman Key Exchange;

3. l'enclave genera un report usando l'istruzione di EREPORT, genera anche una chiave effimera che possa servire nello scambio di dati attraverso una sessione sicura, in caso l'attestazione riesca con successo, e restituisce il REPORT all'applicazione;

4. l'applicazione inoltra il report ottenuto alla Quoting Enclave per trasformarlo in una struttura attestabile remotamente da un'altra macchina, e ad un livello più basso manda la chiave effimera e i dati del protocollo DHKE all'agente remoto seguendo il protocollo di Key Exchange;

5. la Quoting Enclave, avendo ricevuto il report dell'enclave da attestare, chiama la chiave di Report attraverso EGETKEY per confermare la validità locale del report, e successivamente firma il corpo del report con la sua chiave di Attestazione che altri non è che la Chiave EPID Privata del dispositivo;

6. la quote è stata generata, ed è inviata indietro all'applicazione che è pronta a inoltrarla allo sfidante;

7. lo sfidante non si occupa di verificare il report, ma lascia il compito al verificatore, l'Intel Attestation Service. Inoltra la quote dell'enclave inserendola nel body di una richiesta API, e aspetta una risposta. L'Intel Attestation Service usa la chiave EPID Pubblica relativa al gruppo del dispositivo solo per *verificare* la firma generata con la chiave privata, controllando quindi che sia la chiave sia la firma che genera non appartengono a liste di revoca, e manda il risultato allo sfidante, *senza esaminare* il contenuto della quote. L'output assumerà la nuova forma di una struttura chiamata *Attestation Verification Report* che conterrà valori aggiuntivi sulla configurazione della piattaforma della chiave, come informazioni sul software del BIOS (CPUSVN) e la possibilità di presenza di vulnerabilità di sicurezza.

A questo punto, è il gestore del servizio che deve esaminare il contenuto del report, e, in base alle politiche di attestazione adottate, scegliere se reputare l'enclave affidabile o no. Ad esempio, in fase di registrazione della enclave dell'applicazione,

l'utente può aver fornito valori di MRENCLAVE e MRSIGNER in condizioni ideali di esecuzione protetta e senza modifiche: se ci saranno state manomissioni sull'enclave o sul sistema, i valori contenuti nel report saranno diversi, e l'enclave potrà essere ritenuta inaffidabile, interrompendo così l'applicazione (nel caso dello streaming di un video, il video viene così interrotto). Le quote generate possono essere Name Based per essere associate ad un dispositivo o Random Based, a seconda dell'opzione di privacy che il Service Provider sceglie nella fase di registrazione all'IAS: in base all'opzione scelta, l'IAS verificherà che la quote ricevuta sia dello stesso tipo scelto dal portale di registrazione, e se i dati non sono coerenti allora risponderà con un messaggio di tipo "*400 Bad Request*"[14]. Il servizio remoto di attestazione di Intel contatterà l'IPS per poter ottenere la chiave di Attestazione pubblica relativa al dispositivo e verificare il report e la firma di Attestazione allegata.

3.3 Tecnica di divisione del codice

Nonostante non sia una tecnica legata all'attestazione remota, è possibile combinarla con un meccanismo di attestazione remota, come quello di Intel SGX, per mitigare possibili attacchi sull'applicazione del client e per ridurre la superficie di attacco[15] dell'applicazione remota che avrà un ulteriore livello di protezione per i propri dati e offrire un meccanismo di autenticazione. Con la tecnica di protezione del software, si trasforma la logica di una applicazione in una nuova dipendente dall'esecuzione di una applicazione eseguita su una macchina remota (server), e che viene contattata dall'applicazione client per eseguire quella parte dell'applicazione che è stata trasferita. Sull'applicazione del server vengono eseguite delle funzioni selezionate appartenenti alla applicazione originale, da cui sono state rimosse. Questa applicazione quindi non sarà più autonoma, ma dipenderà da una connessione remota: a fronte di una applicazione più complessa, che gestisce la comunicazione e la gestione dei dati che devono essere trasferiti tra le due entità, si hanno proprietà di sicurezza ulteriori. Non solo è possibile proteggere dati e informazioni, ma anche segreti intellettuali come algoritmi e meccanismi proprietari, che non saranno più dipendenti dalla macchina su cui era eseguita originariamente l'applicazione. Se il dispositivo del client è infatti compromesso, tramite attacchi di un agente esterno o la modifica del software mediante reverse engineering [47][16] ottenuto con strumenti di analisi statica e dinamica , gli asset assegnati alla applicazione eseguita su un server remoto sono al sicuro e non compromessi direttamente. Per questo, si trasferiscono i dati e le funzioni più sensibili di una applicazione, in modo tale che le informazioni esposte non rappresentino un danno importante per l'utilizzatore.
Questa tecnica comprende anche svantaggi che devono essere citati: oltre ad una

[14] https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf

[15] Ci si riferisce ad un insieme di elementi dell'applicazione che è esposto ad attacchi, per vulnerabilità intrinseche del software o perché è la parte con cui è possibile l'interazione dall'esterno. Più è grande questa superficie virtuale, più un sistema è attaccabile, quindi meno sicuro.

[16] Attività che serve alla ricostruzione di un software dopo una analisi della sua esecuzione e senza la possibilità di accedere al suo codice sorgente, allo scopo di migliorare una applicazione o di ottenere informazioni segrete o copiarla, come può succedere in ambito militare.

dipendenza dovuta alla rete per quanto riguarda la comunicazione e il trasferimento di dati, il server non ha nessun tipo di controllo sul client che non può essere considerato un hardware o un software sicuro dato che non ha alcun tipo di controllo su di esso[47]. L'applicazione della tecnica è usata anche per implementare meccanismi di protezione di un software dalla pirateria [48].

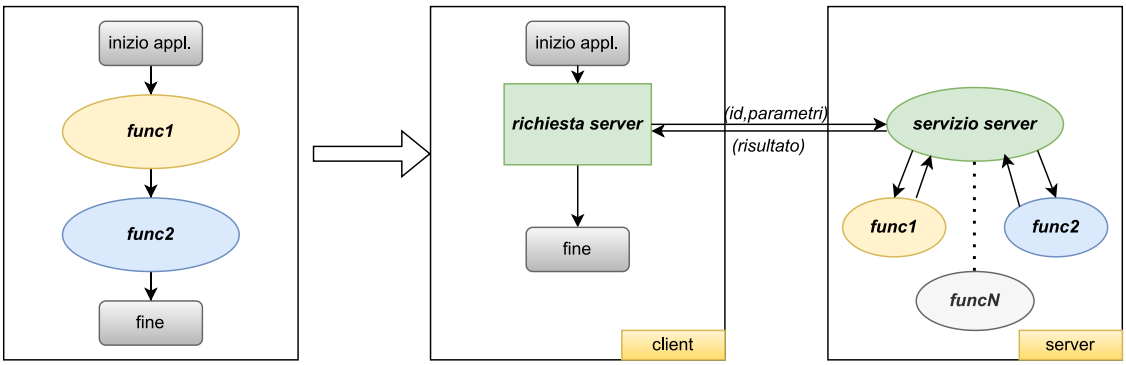


Figura 3.5: Modello di applicazione su cui è applicata la tecnica di divisione codice

Nella Figura 3.5, viene mostrato un esempio di come la tecnica sia implementata su una applicazione: la divisione separa una funzione e la sua logica applicando un taglio tra quello che viene eseguito sulla macchina originale e quello che viene eseguito su una macchina remota. Si crea quindi una applicazione client contenente un punto di partenza e al posto delle funzioni tagliate fuori dal contesto non protetto del client è inserito un meccanismo che ogni volta che deve eseguire una funzione trasferita si interfaccia ad una applicazione di un server: lo scopo del server è eseguire le funzioni su richiesta dell'applicazione, che manda i dati che servono all'esecuzione e le informazioni necessarie, come un identificativo, per indicare cosa deve essere eseguito. Questo meccanismo è implementato da una funzione che sostituisce quelle che sono state spostate remotamente. Il server deve ritornare il risultato di quel processo chiamato al client, in modo tale che possa continuare ad eseguire la logica della sua applicazione. Questo processo verrà ripetuto per ogni volta che l'applicazione dovrà eseguire una delle funzioni presenti sulla macchina remota fino al termine della sua esecuzione.

3.4 Attacchi disponibili sulla RA di SGX

Sono conosciute pubblicamente vulnerabilità che minano la sicurezza del protocollo di Attestazione Remota di SGX. Come riportato in [49], il protocollo completo sembra sicuro, ma l'esecuzione sequenziale e concorrente del protocollo ha dei limiti che possono essere sfruttati da un attaccante che finge di essere un service provider proponendosi autentico e sicuro.
L'enclave può essere plasmata secondo la volontà dell'attaccante forzandola a effettuare Asynchronous Enclave Exit (AEX), cambi di contesto di esecuzione che riportano il flusso dell'applicazione nel contesto non protetto che possiede una superficie d'attacco più grande. Inoltre, nel contesto non protetto ci sono più probabilità di trovare vulnerabilità sfruttabili grazie all'uso di software che può interagire

con il sistema operativo e con le risorse dell'ambiente. Si possono ottenere ulteriori vantaggi usando anche strumenti di estrazione di conoscenza, definiti in protocolli di Zero Knowledge Proof-of-Knowledge (ZKPK)[17] [50], per simulare l'esecuzione di una applicazione e riportarla ad un valore del suo stato specifico, ma non ad un valore che possa inavvertitamente rivelare una chiave segreta. Se una enclave non viene creata con un design che rispetta le linee guida di sicurezza e protezione da tecniche di controllo, è possibile indurla ad assumere un comportamento o a trovarsi in un ambiente creato ad-hoc per generare una AEX involontaria, come succede nell'attacco SGAxe.
Un'altra potenziale preoccupazione è che Intel sia in possesso di tutte le chiavi usate nel processo di firma secondo schema EPID (Sezione 3.2.2) : questo può potenzialmente portare a dubitare di Intel, che potrebbe lanciare attacchi di tipo Man In The Middle sulle enclavi. Con attacco Man In The Middle (MITM), si intende una tecnica di attacco nell'ambito della cybersecurity che tipicamente è composto da due utenti (vittime) e una terza parte esterna che ha accesso al canale di comunicazione usato dai due sistemi che scambiano dati: l'attacco MITM mira a compromettere la confidenzialità dei dati, ascoltando i dati in transito (eavesdropping); l'integrità dei dati, raccogliendoli e modificando i messaggi; la disponibilità, cancellando alcuni messaggi appartenenti alla sequenza o alterando il flusso per mutare la comunicazione [51]. Le quote che vengono mandate all'Intel Attestation Service, inoltre, contengono il Service Provider ID (SPID)[18] che gli è stato assegnato dopo la registrazione sul portale di Intel, e questa credenziale permette un tracciamento delle attività di attestazione richieste in un periodo di tempo, portando così ad una potenziale violazione di Intel della privacy.

SGAxe

SGAxe è un attacco sviluppato da una collaborazione tra l'Università del Michigan e l'Università di Adelaide[19] che sfrutta una vulnerabilità dei processori solo parzialmente mitigata dall'ultimo quadrimestre del 2018 in poi da Intel. Permette di rompere il protocollo di attestazione remota ottenendo le chiavi di attestazioni private della Quoting Enclave per creare quote non originali. L'attacco è possibile sfruttando CacheOut, già citato precedentemente nella Sezione 2.3.4, per leggere la chiave di Sealing (vedesi Sezione 2.2) ottenuta con l'istruzione EGETKEY. Questa chiave viene però cancellata dalle API di Intel quando le operazioni di Sealing e Unsealing vengono completate, quindi l'attacco deve essere fatto contemporaneamente ad una di queste operazioni. Bisogna combinare driver Linux modificati per

[17] Protocolli crittografici che consistono in un agente (*prover*) che deve dimostrare ad un verificatore (*verifier*) la prova della veridicità di una affermazione, non fornendo nessun dettaglio al di fuori della affermazione, dimostrando così che è in possesso di informazioni sensibili senza svelare quali siano.

[18] Un identificativo assegnato al Service Provider usato come dato di autenticazione della propria identità.

[19] https://sgaxeattack.com/

fare continui swapping di pagine di memoria contenenti la chiave, insieme ad attacchi controllati con canali laterali per forzare cambi di contesto lanciati da AEX, e con processi di speculazione[20] ottenere la chiave di Sealing. Ottenuta la chiave, si attacca lo spazio di memoria usato della Quoting Enclave: studiando gli indirizzi di memoria usati, e trovata la chiave privata EPID in forma cifrata, la si decritta con la chiave di Sealing, rompendo i meccanismi di protezione che SGX ha applicato sui suoi materiali crittografici segreti.
A questo punto, è possibile generare quote non originali con dati arbitrari garantendo una falsa veridicità: questa viene fornita compiendo sui dati una firma ottenuta con la chiave autentica. Il Servizio di Attestazione Intel confermerà che lo stato dell'enclave sia valido, dato che l'unica operazione che compie è soltanto la verifica della firma EPID, ed essendo stata ottenuta in una modalità illegale non potrà mai fornire prove di una riuscita compromissione dei segreti della macchina SGX. Con la generazione di false quote, è addirittura possibile superare con successo il controllo di attestazione di una applicazione che gira su una macchina non SGX.
Una parziale mitigazione da Intel è stata fornita nella versione 4 delle API che vengono usate per contattare l'IAS: nel body dell'Attestation Verification Report è presente in questi casi un parametro "SW_HARDENING_NEEDING" inserito quando riconosce che la chiave EPID Pubblica usata nella validazione della firma di attestazione appartiene ad un processore che è vulnerabile a questo exploit. In questo modo, segnala al server che la piattaforma usata dall'enclave non è completamente affidabile e lascia a lui la responsabilità di accettare i rischi legati alla sicurezza del servizio di interagire con un utente potenzialmente pericoloso.

SgxPectre

SgxPectre nasce da uno studio dell'Università dello Stato dell'Ohio nel 2018, e prima che l'attacco venisse svelato pubblicamente fortunatamente è stato risolto da Intel, che è stata contattata dai ricercatori privatamente per aiutarli nella pubblicazione di una patch risolutiva. Dimostrazioni provano la compatibilità di questa tecnica di attacco con il Software Development Kit di Intel per SGX, Gramine SGX[21] e il framework di Rust SGX [46][22]. Basato su canali laterali che influenzano le tempistiche delle operazioni di una memoria cache, compie con la manipolazione della memoria e del flusso di esecuzione dell'applicazione il furto dei segreti contenuti dentro una enclave firmata da Intel, ottenendo la chiave di attestazione e i segreti di provisioning e di sealing dei dati.
Si suppone che l'attaccante abbia il pieno controllo della macchina SGX, quindi del suo sistema operativo, dei permessi di uso delle risorse compreso gli argomenti passati all'enclave e gli argomenti che ritorna quando cambia contesto di esecuzione, che possa lanciare thread paralleli e altre enclavi parallele a quella da attaccare. Eseguendo prima un gadget (definito in Sezione 2.3.1) scritto appositamente, si riesce a modificare il contenuto del buffer di predizione del branch, una pipeline di

[20] Comprendono l'esecuzione di cinque round della QE in modalità produzione, interrompendo il processo nell'istante immediatamente successivo alla generazione le chiavi AES.

[21] https://gramine.readthedocs.io/en/stable/index.html

[22] https://edp.fortanix.com/docs/

valori consecutivi, in modo tale che quando il programma compierà il salto secondo il normale flusso del programma, arriverà all'indirizzo inserito dal codice del gadget che contiene una istruzione capace di causare un leak di informazioni. Successiva-

```
lea .Lflush, %r10
push %r10
lea .Lrsb, %r10
push %r10
...
push %r10
.Lrsb:
ret
.Lflush:
...
```

Figura 3.6: Gadget caricato per modificare lo stack

mente è il processore ad essere manipolato tramite operazioni di flush di linee della memoria cache e di svuotamento della RSB: queste operazioni aumentano le possibilità di eseguire specularmente le istruzioni che portano al leak dei segreti, prima che rilevi l'errore di salto e ripulisca il buffer, poichè forzano la CPU ad eseguire una lettura della BTB. Il gadget usato per svuotare la RSB è definito in Figura 3.6: quando eseguito, carica un indirizzo nel buffer che appartiene all'area di codice dall'etichetta ".Lflush" e un indirizzo per l'area di codice con etichetta ".Lrsb", iterando questo comportamento per 16 volte. Quando viene chiamata l'operazione di ritorno all'indirizzo RET, il programma salterà all'indirizzo che è stato ottenuto usando l'istruzione Load Effective Address (LEA), non permettendo all'applicazione di ritornare al normale flusso di esecuzione fino a quando tutte le righe della RSB non saranno svuotate.
A questo punto, configurando tutti i registri usati dalle istruzioni malevoli con valori specifici, possiamo assicurarci che la memoria dell'enclave possa lasciare tracce nella cache che possono essere riconosciute e studiate. Una volta che le trappole sono state preparate fuori dall'ambiente protetto appartenente all'enclave, si esegue il codice protetto, permettendo quindi che la memoria sia letta e che i valori siano estratti grazie ad attacchi Prime+Probe [46] (già descritti nella Sezione 2.3.4).
I fix applicati da Intel per contrastare SgxPectre sono stati aggiornamenti del microcodice dell'hardware per supportare l'Indirect Branch Restricted Speculation[23],

[23]Permette di non controllare la predizione di salto dell'enclave dal software che è eseguito fuori dal contesto trusted.

il Single Thread Indirect Branch Predictors[24] e l'Indirect Branch Predictor Barrier[25] e di aggiornare il CPUSVN, in modo tale che quando la quote dell'enclave viene letta dall'IAS, riconosce che la firma di attestazione è eseguita con una firma non aggiornata alla CPUSVN attuale, notificando con un messaggio l'errore di obsolescenza.

3.5 Tecniche alternative di RA

In questo Capitolo, verranno presentate alcune tecniche conosciute per implementare l'attestazione remota in un sistema informatico. La funzionalità di attestazione non è garantita automaticamente nel sistema di un TEE: basti pensare a Trustzone di ARM che implementa una parziale attestazione remota solamente grazie a protocolli open-source per dispositivi mobile [52].
Per SGX, sono stati ideati numerosi protocolli open-source che espandono le funzionalità di attestazione remota della piattaforma, come MAGE [53], OPERA [54] e TruCE[26].

3.5.1 TXT+TPM

Trusted eXecution Technology permette di effettuare sul sistema attestazione remota, combinando l'architettura di Intel con il Trusted Platform Module. Per avere una attestazione della piattaforma, bisogna che essa abbia una Root Of Trust for Measurement, Root Of Trust for Storage e una Root Of Trust for Reporting. TPM[27] svolge il ruolo di RTR e RTS, mentre Intel TXT funge da RTM[28]. TXT deve misurare l'integrità del software che viene eseguito per ottenere una credenziale come prova dell'identità del sistema. La credenziale deve essere trasmessa ad un verificatore che potrà reputare il sistema del cliente come affidabile e sicuro. L'attestazione potrà essere poi essere ottenuta con due modi diversi: OpenAttestation e Intel Trust Attestation Solution.

[24] Meccanismo che limita la condivisione dei predittori di salti tra i processori logici di un sistema, per info:https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/single-thread-indirect-branch-predictors.html

[25] Crea una barriera di memoria per bloccare il software eseguito prima della creazione della barriera dal controllare gli obiettivi predetti dai salti indiretti.

[26] https://github.com/IBM/sgx-trust-management

[27] https://trustedcomputinggroup.org/wp-content/uploads/2019_TCG_TPM2_BriefOverview_DR02web.pdf

[28] https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf

OpenAttestation

OpenAttestation[29] è una soluzione per sistemi Linux che mette a disposizione un SDK Open Source fondato su un protocollo di attestazione remota ideato da TCG. Può essere integrato in servizi di sicurezza Enterprise e all'interno di piattaforme cloud, installandolo sui server che devono attestare e sui client che richiedono l'attestazione. Tuttavia, è in stato di deprecazione, dato che i suoi sviluppatori hanno deciso di trasformarlo in un framework che controlla l'integrità di documenti con l'integrazione delle blockchain per operazioni crittografiche[30].
Il server deve essere predisposto con il sistema operativo corretto, uno a scelta tra Ubuntu 12.10 e Fedora 19, deve installare l'SDK e deve creare delle whitelist in cui sono contenuti i sistemi che fanno parte del Measurement Launch Environment e le misure registrate. Il client deve abilitare TPM e TXT dal BIOS del dispositivo, e installare gli agenti di Open Attestation che sfruttano il TPM per la misurazione. Gli agenti creano sessioni di comunicazione con i server di Open Attestation per registrarsi e comunicano con API che seguono il protocollo REST per le operazioni e i dati relativi ai sistemi (macchine virtuali, hosts). Anche OpenAttestation contiene uno schema di whitelist e di API dedicate alla loro gestione.

Intel Trust Attestation Solution

Intel Trust Attestation Solution invece è una soluzione offerta da Intel che usa una "soluzione di attestazione e verifica affidabilità multi-ipervisore, multi-dispositivo per server fisici, virtuali, client, sistemi embedded e reti"[31]. Per far partire il sistema, bisogna:

- installare il componente lato server per avere un Server di Attestazione centrale;
- installare il componente agente sicuro sugli host protetti dai Measurement Launch Environment;
- creare e aggiornare i MLE aggiungendoli nelle whitelist di dispositivi da proteggere, e ripetere il passo solo in caso di update del software dei componenti;
- registrare nella whitelist anche i dispositivi degli host che sono controllati tramite i valori dei MLE (quindi i componenti software come BIOS, VM, ipervisori, eccetera);
- installare il componente APIClient per poter registrarsi al Server d'Attestazione, ottenere le chiavi di identificazione e usare le API per la gestione del componente.

[29] https://01.org/blogs/2014/openattestation-oat-project

[30] https://www.openattestation.com/

[31] https://www.intel.com/content/www/us/en/developer/articles/guide/intel-trusted-execution-technology-intel-txt-enabling-guide.html

Gli agenti comunicano tra di loro grazie a API REST che permettono l'esecuzione di funzioni scritte in Java, C# e Python [55, Cap. 4]. Le API si dividono per le operazioni eseguite in più tipi:

- registrazione componenti e agenti;
- query per avere asserzioni sullo stato di fiducia degli host e dei componenti;
- report delle misure e delle whitelist;
- automazione di registrazioni di tutti gli host all'interno di una VM e creazione di un MLE con base un host sicuro;
- gestione degli utenti, dei loro permessi e gestione dei certificati.

3.5.2 KeyLime

KeyLime [56] è una soluzione alternativa Open Source integrabile con i sistemi TPM 2.0 che può essere installata su Docker, Ansible e sistemi Rust. Permette l'attestazione remota sulla misura del boot di un sistema, la misura del boot di una applicazione e funzioni di checkup in run-time dell'integrità di componenti. Usa shim[32] come modulo che estende le misure dei registri PCR del TPM quando viene lanciato nella fase di boot, e permette di indicare nella configurazione quale deve essere il PCR da misurare. Le misure saranno conservate in un file JSON: il parametro *tpm_policy* è un oggetto JSON codificato nel formato di una stringa che contiene i registri PCR che devono essere presenti nella quote generata dal TPM. Per ognuno dei PCR inseriti, saranno inclusi i valori che sono considerati accettati, creando così un sistema di whitelist di misure attendibili. KeyLime divide il suo workflow in quattro ruoli: un agente, un verificatore, un registratore e una istanza tenant.
L'agente lavora sul sistema da attestare e comunica con il TPM per ottenere le misure registrate e generare le quote da mandare al verificatore. Quando il nodo è attestato, è possibile cifrare le informazioni scambiate usando payload sicuri, ossia payload cifrati con chiavi che devono essere condivise con meccanismi di Key Sharing. Può anche eseguito in modalità Certificate Package Mode, in cui le comunicazioni vengono fatte con certificati X.509 e le istanze Tenant assumono il ruolo di Certificate Autority nei confronti degli Agent.
Il registratore è l'istanza che si occupa del processo di abilitazione di un agente al ruolo di verificatore del sistema, verificando che le chiavi e i certificati ricevuti in fase di abilitazione siano validi.
Il verificatore è l'agente che ha ottenuto il ruolo di attestare gli agenti comunicando i risultati del controllo dei dati di attestazione ricevuti (quote che comprendono i valori di misura dei boot, dei PCR e dei dati che compongono la TCB).
L'istanza tenant è uno strumento disponibile per shell di comandi che permette la gestione degli agenti del sistema e che si occupa anche della revoca di essi e dello scambio del payload dei dati, che può essere anche cifrato usando chiavi condivise

[32] https://github.com/rhboot/shim

create con protocolli di Key Sharing. Offre la possibilità di rilevamento di attacchi di manipolazione delle quote del TPM installato.
L'agente tenant verifica il certificato relativo alla Endorsement Key[33] ripercorrendo tutta la catena di fiducia dei certificati per assicurare che la chiave appartenga ad un vero software/hardware TPM. Usa successivamente comandi TPM (*MakeCredential, ActivateCredential*) per confermare che la chiave di Attestazione[34] sia legata alla EK identificata dal certificato. Usa poi la chiave di Attestazione, fornita nella fase di registrazione dell'agente, per verificare la quote.
Verifica anche l'integrità del Boot attraverso la sua misura totale, permettendo anche di esaminare specifici PCR. Inoltre permette di essere integrato con Linux Integrity Measurement Architecture[35] (IMA), con interazione remota attraverso un agente lanciato per interagire da remoto e con comandi da terminale. Linux IMA è un sottosistema che si occupa principalmente di due operazioni:

- *misurazione* dell'hash dei file e dei programmi prima che siano lanciati;
- *verifica* delle misure con gli hash memorizzati per verificare l'integrità dei dati.

Il sistema di Linux IMA deve essere in primis attivato dalla configurazione del kernel, e permette la ridefinizione delle regole di misurazione e di verifica delle misure.

[33] Chiave RSA segreta residente nel TPM che può essere usata solo per la cifratura. Il certificato contiene la chiave pubblica asimmetrica ed è firmato dal produttore del TPM

[34] Attestation Key del TPM, usata per la firma del digest creato dal TPM

[35] https://sourceforge.net/p/linux-ima/wiki/Home/#integrity-measurement-architecture-ima-measurement

Capitolo 4

Framework SGX

Il framework SGX nasce nel Politecnico di Torino come lavoro di tesi di Matteo Costa [57]: l'obiettivo da raggiungere fu la creazione di un framework, sviluppato in Python, che si occupasse della conversione semi-automatica di applicazioni scritte in C in applicazioni che sfruttano l'architettura SGX della piattaforma di esecuzione, in modo tale da creare ambienti di esecuzione protetti isolati dal resto del sistema. Sebbene con alcune limitazioni che non permettono la piena compatibilità con ogni codice sorgente, l'uso di notazioni all'interno del codice sorgente permette di indicare i punti strategici di intervento di trasformazione: il framework riconosce questi appunti, crea tramite una analisi statica del codice un modello che modifica per crearne un nuovo sorgente dell'applicazione, e restituisce come output i nuovi file sorgenti modificati, i file che servono a compilare il codice e i dati appartenenti all'enclave.
L'applicazione SGX cambia il contesto di esecuzione del programma in base al tipo di funzione che deve eseguire, salvando in memoria i dati appartenenti al momento precedente della transizione e ricaricandoli al termine dell'esecuzione: le chiamate alle funzioni eseguite nel contesto protetto (ECall) e le chiamate alle funzioni nel contesto non protetto (OCall) che saranno usate sono definite in un file chiamato EDL, acronimo di *Enclave Definition Language*[1]. In questo file sono definiti i prototipi delle funzioni che chiederanno di essere eseguiti in un ambiente di esecuzione diverso, i tipi delle strutture dati usate nei prototipi delle funzioni e le librerie che devono essere incluse per eseguire le funzioni: nella fase di compilazione dell'applicazione, questo file verrà letto e tradotto per generare ed eseguire il link delle librerie del Software Development Kit di Intel che contengono le funzioni di SGX.
In questo capitolo verranno presentate notazioni usate dal framework e la loro sintassi nella Sezione 4.1.1, e nella Sezione 4.1.2 viene descritto il processo di conversione e dei moduli usati nella versione del framework. Nella Sezione 4.3 verranno poi descritti gli strumenti esterni per elaborazioni di tipo intermedio durante l'analisi statica del codice sorgente.

[1] https://www.intel.com/content/www/us/en/developer/articles/training/intel-software-guard-extensions-tutorial-part-7-refining-the-enclave.html

4.1 Architettura e workflow

Il processo di conversione del framework segue i seguenti passi:

1. analizza il codice cercando le strutture dati usate e le modifiche richieste per inserire le funzionalità di protezione dell'applicazione di Intel SGX, dividendolo in parte protetta e parte non protetta;

2. crea i nuovi file relativi al contesto di esecuzione dell'applicazione protetto e al contesto non protetto, i file relativi alle funzioni proxy, in cui si effettua un cambio di contesto di esecuzione per eseguire funzioni fuori o dentro la memoria protetta, e sostituisce il vecchio codice inserendo le chiamate alle funzioni proxy e spostando le strutture in base al contesto in cui devono essere eseguite;

3. crea il file EDL, il file *enclave_manager.c* e l'header associato *enclave_manager.h* in cui sono presenti le funzioni di creazione e modifica dell'enclave dell'applicazione, e salva il risultato finale.

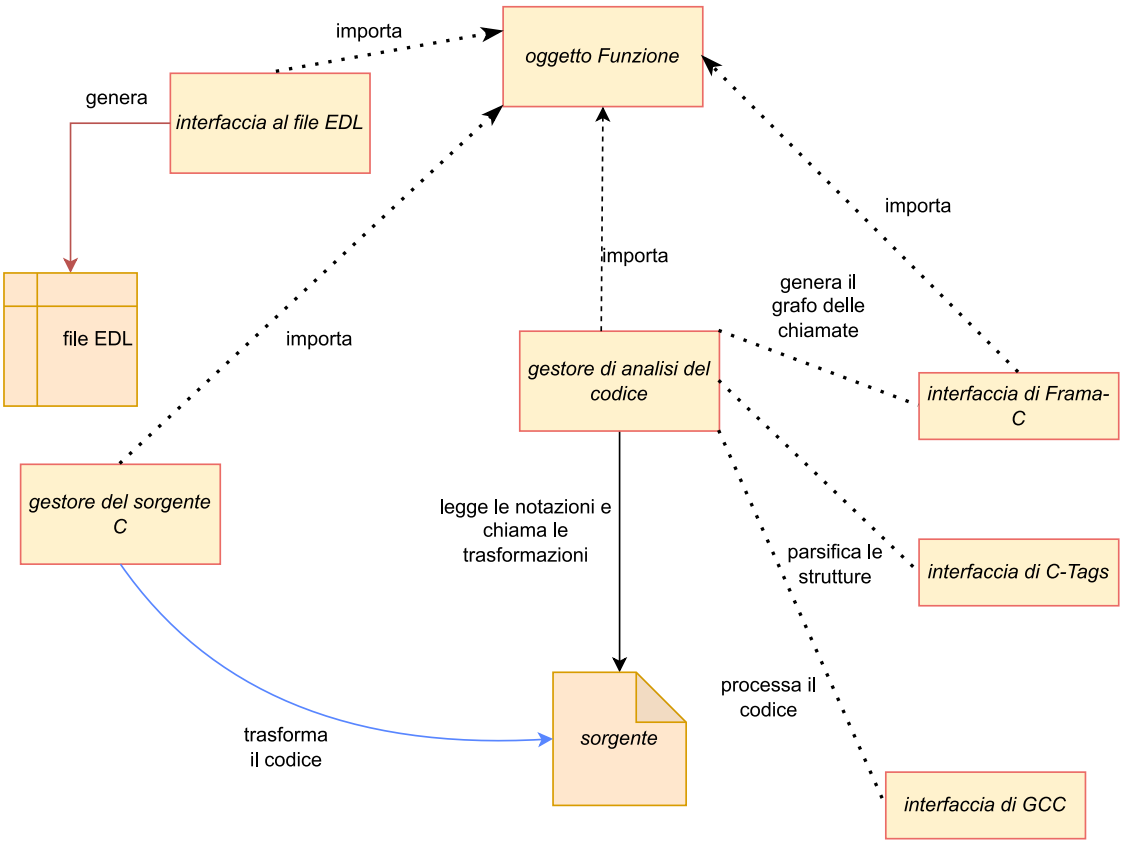


Figura 4.1: Architettura del framework

In Figura 4.1 è illustrata l'architettura del framework, e le relazioni tra i suoi moduli. Il processo è gestito da un modulo che si occupa dell'analisi del codice letto da un sorgente in cui sono state inserite notazioni dallo sviluppatore per creare enclavi SGX all'interno dell'applicazione. Questo gestore chiama tutti i sottomoduli

che sono dedicati allo svolgimento delle operazioni descritte nella lista precedente. Tramite interfacce, chiama gli strumenti esterni Frama-C, C-tags e GCC per poter ottenere informazioni come le strutture dati usate. Uno dei sottomoduli sarà incaricato della fase del processo in cui si genera il file EDL per poter compilare l'enclave finale. Il framework importa una lista di parametri dal file di configurazione *frameworkConfig*: il parametro *files* indica quali sono i sorgenti con le notazioni che devono essere convertiti, e la cartella in cui devono essere cercati è indicato dal parametro *rootFolder*. Letti questi parametri, il framework lancia il modulo principale di gestione della conversione, su cui sono presenti metodi di lettura e ricerca di notazioni del codice.

4.1.1 Uso delle notazioni

Per indicare le modifiche richieste, lo sviluppatore deve inserire nel codice sorgente notazioni con una sintassi precisa da seguire. Alcune hanno come scopo la riscrittura di funzioni in funzioni ECall o funzioni OCall, definendo gli argomenti di tipo puntatori e vettori di dimensione statica come puntatori a buffer. Questi buffer sono usati poi dalle librerie SGX di cambio di contesto per trasferire i dati, copiandoli dalla memoria puntata alla memoria protetta dell'enclave per le ECall o viceversa per le OCall. Nelle notazioni, deve essere definita la dimensione dei buffer usati e la modalità di copia dei valori, ossia come l'enclave, che non può accedere alla memoria non protetta, deve definire il buffer di copia della variabile per poter conoscere il suo valore. Questo valore sarà poi inserito nel file EDL compilato dal framework secondo le notazioni di definizione delle ECall e OCall all'interno del codice. Il framework inoltre consente di copiare variabili all'interno della zona di memoria dell'enclave, o spostare dati al suo interno, che fungono da confini alle sezioni di codice interessate al movimento. Non è obbligatorio tuttavia che l'applicazione convertita debba copiare o spostare dei dati all'interno dell'enclave.

Notazione per ECall e OCall

La sintassi per definire una ECall e una OCall differisce soltanto per il termine identificativo del tipo di funzione effettuata.

```
#define sgx_ecall_<nome_funzione> (<[nome_param>, <tipo_copia>,
   <dim_buf>]..)

#define sgx_ocall_<nome_funzione> (<[nome_param>, <tipo_copia>,
   <dim_buf>]..)
```

Lo sviluppatore usa questa notazione per indicare che una funzione appartenente al normale flusso dell'applicazione non protetto deve diventare una funzione che viene eseguita all'interno del nuovo contesto protetto definito o se è chiamata dal contesto, definendo anche la modalità di trasferimento degli argomenti. Per indicare che tipo di trasformazione deve essere svolta sulla funzione, bisognerà usare una tra le parole "ecall" e "ocall". La direttiva #define avvisa il compilatore che bisognerà

creare una traduzione per l'oggetto SGX definito. All'interno delle parentesi tonde, vanno definiti gli argomenti delle funzioni secondo lo schema indicato:

- <nome_param> indica il nome del parametro che si sta passando, rispettando l'ordine della dichiarazione della funzione;
- <tipo_copia> segnala il meccanismo di copia usato dal puntatore al buffer di quel parametro;
- <dim_buf> indica la dimensione in byte del buffer allocato nella memoria destinazione.

La modalità di copia del buffer indicata dal parametro <tipo_copia> è scelta in base alla direzione dei dati da trasferire[2]:

- quando sono copiati nella regione opposta al valore contenuto, trasferendo i dati attraverso un buffer creato in questa memoria e non registrando le modifiche dei dati nella regione di partenza (modalità *in*);
- quando nella regione opposta è creato un buffer di dimensione specificata inizialmente vuoto su cui si opera, e alla fine il valore modificato è copiato nel buffer appartenente alla regione di partenza (modalità *out*);
- è possibile adottare i comportamenti precedenti insieme, allocando un buffer nella regione opposta al valore che viene copiato alla partenza della chiamata, e ritornando il valore aggiornato alla fine della chiamata con una copia nel buffer di partenza (modalità *in, out*);
- passare direttamente il puntatore senza fare una verifica dei contenuti (modalità *user_check*).

La definizione dei parametri all'interno delle parentesi quadre è valida soltanto per i vettori e i puntatori.
Per la conversione di una applicazione, è necessario che almeno una funzione sia definita come ECall: in caso contrario, non esisterebbe un ponte per entrare nell'enclave associata all'applicazione, e la generazione di un contesto protetto diventerebbe così inutile. Per questo motivo, il framework controlla che ci sia almeno una notazione di questo tipo, altrimenti interrompe il processo, restituendo la causa dell'errore come messaggio.
In Figura 4.2 è presente un esempio di trasformazione di funzione in una ECall seguendo la sintassi del framework: la notazione deve essere inserita prima della dichiarazione della funzione, deve iniziare con il suffisso *sgx_ecall_* seguito dal nome della funzione. Subito dopo, poiché la funzione usa un vettore di caratteri di dimensione conosciuta per poter generare il messaggio, bisogna trasferire il contenuto di quel buffer in uno che si trovi all'interno dell'enclave: al posto del campo

[2] https://github.com/intel/linux-sgx/blob/master/SampleCode/SampleEnclave/Enclave/Edger8rSyntax/Pointers.edl

<tipo_copia> si usa il valore "i" per rappresentare la modalità "in", in modo tale che il messaggio contenuto nel vettore sarà trasferito all'interno della memoria dell'enclave e potrà stamparlo, non richiedendo la copia del valore della variabile al ritorno nel contesto non protetto dell'applicazione dato che la funzione chiamata non esegue modifiche sui dati da registrare.

Copia e movimento dei dati

Un enclave può anche contenere dati sensibili che non devono essere direttamente accessibili, quindi lo sviluppatore potrebbe volere spostare un dato definitivamente all'interno di essa, rimuovendolo dalla regione di appartenenza iniziale, o creare una copia del dato da poter usare nell'ambiente isolato, in modo tale che si faccia accesso solo a questa. Nel primo caso, chiamato anche *movimento* dei dati, si usa per indicare l'inizio della sezione di dati da spostare in maniera esclusiva la coppia di direttive:

```
#pragma move_start
    ...
#pragma move_end
```

mentre nel caso in cui bisogna effettuare una copia dei dati:

```
#pragma copy_start
    ...
#pragma copy_end
```

La Figura 4.3 è un esempio di codice che indica come vengono usate le direttive: il codice relativo alla dichiarazione delle variabili da trasferire o copiare deve essere all'interno del blocco che inizia con la direttiva start e termina con la direttiva end, e queste direttive delimitano il perimetro del codice da trasferire. Nell'esempio, la matrice e il vettore saranno inseriti all'interno dell'enclave, e le loro definizioni saranno rimosse dal codice appartenente all'applicazione: se i riferimenti nel codice non sono stati modificati, l'applicazione conterrà istruzioni che fanno riferimento a variabili che non esistono più nella memoria riservata all'applicazione quando viene lanciata. Dopo l'applicazione delle notazioni, le variabili sono memorizzate nella memoria protetta dall'enclave, perciò bisognerà modificare il sorgente relativo alla parte dell'applicazione eseguita nel contesto non protetto, spostando le istruzioni che interagiscono con le variabili spostate in funzioni eseguite nel contesto protetto che possono accedere alla memoria protetta, e tramite ECall attivare il cambio di contesto.

```
1
2 #define sgx_ecall_printcard ([str, i, len])
3 int printcard(unsigned char *str, int len){...}
```

Figura 4.2: Esempio di notazione ECall usata

4.1.2 Conversione dell'applicazione

La Figura 4.4 mostra la sequenza di operazioni eseguite ad alto livello per il funzionamento del framework. Il processo richiede come passo iniziale l'interazione dello sviluppatore che deve inserire nel codice le macro per indicare le funzionalità da inserire nell'applicazione finale. A questo punto, il framework viene lanciato e si esegue una analisi statica del codice (tramite gli strumenti Frama-C e C-tags, che saranno presentati nella Sezione 4.3.2 e Sezione 4.3.1) per generare un modello di sorgente che rappresenta l'applicazione originale e che dovrà essere modificato in un modello che rappresenti la nuova applicazione. Per realizzare questo modello, il framework legge le macro inserite all'inizio del processo per segnalare le modifiche, e si avvale dei moduli incaricati delle varie funzionalità di attestazione. Dal modello così modificato, infine sono generati i file che appartengono alla applicazione compatibile con SGX e i file necessari per poterla compilare.

La Figura 4.5 invece mostra i file generati al termine del processo di conversione dell'applicazione usando il framework SGX. Il file sorgente è il file iniziale su cui deve essere effettuato un lavoro di trasformazione. In una cartella apposita, saranno generati i file convertiti in due parti: la prima è dedicata all'applicazione normale, con le funzioni spostate che saranno sostituite da chiamate che invocano il cambio di contesto di esecuzione e indicano con un puntatore la funzione da eseguire; la seconda parte è quella che comprende tutte le enclavi che lo sviluppatore ha richiesto e il codice che dovrà essere eseguito all'interno.
Ogni enclave avrà una sua cartella *enclaveXX*, con l'identificativo dell'enclave generata usato come suffisso, per contenere i file necessari, tra cui il file "*enclave.edl*" per poter generare le chiamate proxy per i due ambienti. Il framework poi si occupa anche di preparare l'ambiente per la fase di compilazione, che avverrà in maniera automatica o manuale a seconda dell'opzione scelta nel file di configurazione: se sarà scelta la compilazione automatica, il framework userà il terminale aperto per il lancio del framework per eseguire il comando *make* che legge in input un file testuale *Makefile*, in cui sono presenti le regole di compilazione e le istruzioni per linkare le librerie ai file oggetto, creando l'eseguibile finale.
Il modulo *CodeAnalyser* è il modulo principale che gestisce il metodo "main", la prima funzione chiamata dal framework e che riceve per input tutti gli argomenti del file di configurazione. Questo modulo ha come primo scopo quello di inizializzare tutte le strutture necessarie alla conversione e di lanciare l'analisi statica

```
1
2 #pragma move_start
3
4 char password_users[100][100];
5 char usernames[100];
6
7 #pragma move_end
```

Figura 4.3: Esempio di uso della notazione di movimento dei dati di una applicazione

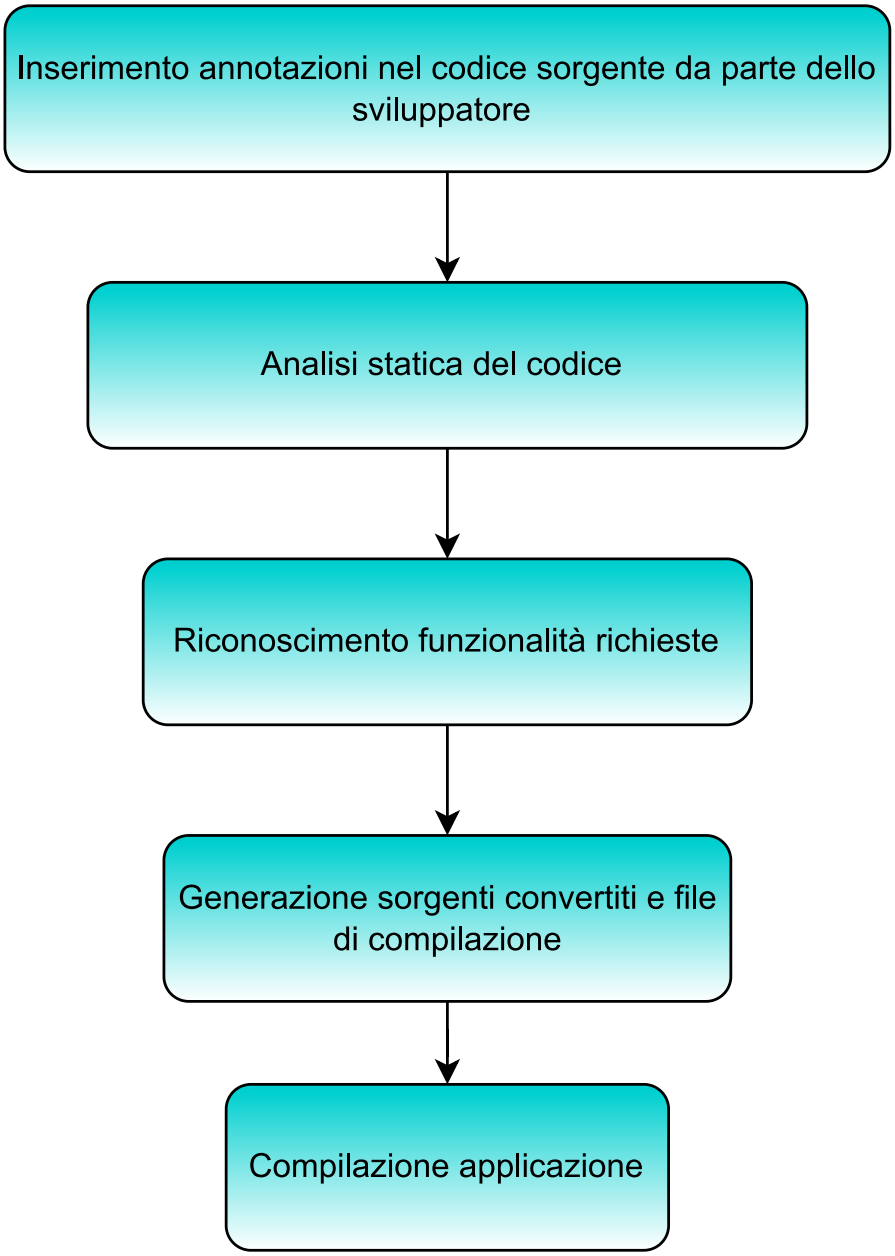


Figura 4.4: Operazioni eseguite nel processo operativo del framework

del codice: chiama l'interfaccia a C-tags (*ctagsInterface*) per poter avere la lista dei nomi di tutte le strutture usate (funzioni, enumerazioni, unioni, definizioni e struct), assicurandosi con l'uso di asserzioni che non ci siano errori come duplicati di nomi.
Il passo successivo è il parsing di tutti i file richiesti, per creare un dizionario che contenga al suo interno le strutture usate. Ogni file viene passato al modulo *cFileReaderWriter* per creare una istanza della rispettiva classe e poterla inserire nel dizionario. La parsificazione avviene mediante espressioni regolari che leggono il codice per poter distinguere tutte le strutture usate e salvarle nella voce del dizionario che avrà per ogni tipo di struttura una voce apposita. Al completamento del dizionario, vengono cercate tutte le funzioni, separandole in base al contesto di esecuzione in quelle protette appartenenti all'enclave e quelle che rimarranno nel

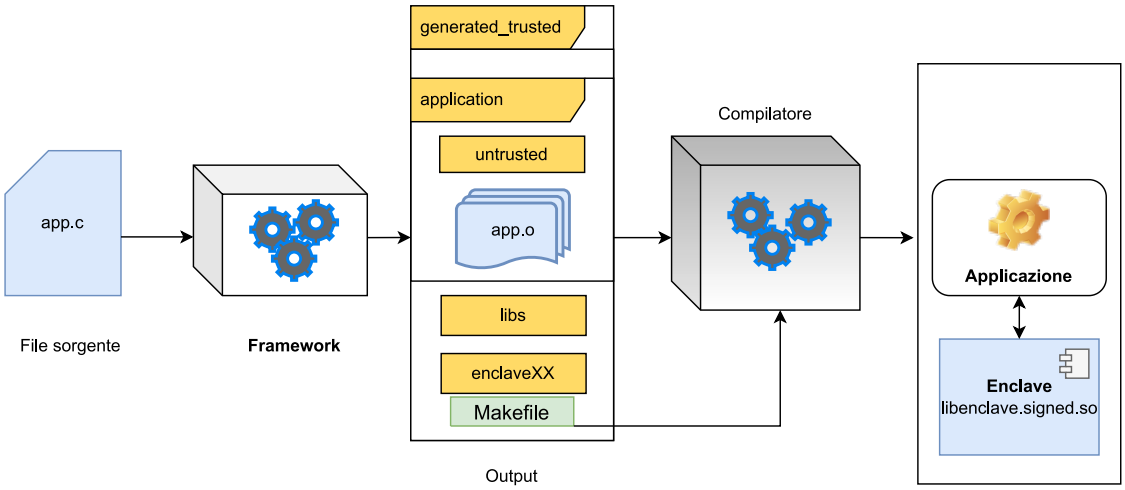


Figura 4.5: Schema della generazione dell'applicazione finale

lato esposto dell'applicazione con le notazioni usate nella Sezione 4.1.1. In questa ricerca, è usato il modulo *gccInterface* per avere un vettore in cui sono inseriti tutti i file header delle funzioni eseguite dalle ECall o dalle OCall.
La definizione delle funzioni dell'enclave è fondamentale per la definizione del codice, e permette quindi di incominciare a generare il codice dell'enclave. Le funzioni devono essere definite in una parte non sicura e una parte sicura:

- la parte chiamante (non sicura per una ECall, sicura per una OCall) è una funzione wrapper che si occupa della transizione al contesto protetto. Verifica che l'enclave sia inizializzata (se non lo è, si occupa di inizializzarla), e chiama l'altro lato della funzione, passandole come argomenti l'id dell'enclave (solo se la funzione definita è una ECall), il puntatore al valore di ritorno della funzione, e gli argomenti originali. La funzione chiamata invece restituirà un intero che rappresenta l'esito dell'operazione avvenuta dentro l'enclave, nel formato sgx_status_ret, che segnala eventuali errori avvenuti come l'invio di argomenti sbagliati;

- la parte chiamata contiene il codice originale della funzione, rinominando il nome della funzione aggiungendoci un suffisso "_trusted" (se è una ECall) per indicare il diverso contesto di esecuzione.

Viene generato quindi un file *enclave_u.c* e un file header *enclave_u.h* che l'applicazione principale includerà per effettuare le chiamate, insieme ad un file *enclave.c* in cui inserire il codice implementativo del contesto protetto delle funzioni convertite. Il modulo *Function* contiene i metodi che distinguono e separano le due parti, e verifica le condizioni necessarie per le funzioni che devono essere eseguite all'interno dell'enclave, come il rispetto della sintassi dei puntatori degli argomenti delle funzioni.
Dopo aver svolto questa divisione del codice delle funzioni, e aver aggiunto direttive di inclusione alle librerie al file "enclave.c", vengono inserite all'inizio del codice le definizioni e i dati segnalati dalle notazioni di copia e spostamento. Definita la parte di codice appartenente al lato protetto dell'applicazione, bisogna generare le interfacce che permettono alla parte esposta di comunicare con quella isolata.

Generazione chiamate proxy

Il file EDL è un file testuale simile al file sorgente di una applicazione scritto in C, usato per definire le funzioni di interfaccia, o *funzioni proxy*, usate per passare dall'applicazione non protetta all'enclave e viceversa. Le funzioni proxy sono generate grazie allo strumento *edger8r*[3] di Intel, il cui ruolo è la traduzione del codice del file EDL. Il principale obiettivo delle funzioni proxy generate è quello di passare i dati all'interno della memoria dell'enclave o all'esterno di essa; i valori ritornati non sono passati per valore ma per riferimento tramite un puntatore che indica l'indirizzo del risultato. Il file EDL di un'applicazione convertita è compilato da un modulo del framework SGX che legge le notazioni inserite per definire ECall e OCall: per ognuna delle notazioni riconosciute, è inserito all'interno del file il prototipo della rispettiva chiamata proxy, con i parametri usati e le opzioni di copia dei dati richieste dallo sviluppatore. Il file conterrà all'interno:

- i prototipi delle funzioni fidate nel file header "enclave_t.h";
- i prototipi delle funzioni non fidate nel file header "enclave_u.h";
- le strutture dati degli argomenti dei prototipi delle funzioni citati;
- le librerie da importare, altri file EDL, le costanti che devono essere presenti nell'enclave.

"Enclave_u.h" e "Enclave_t.h" dovranno essere inclusi rispettivamente nel codice dell'applicazione e in "enclave.c". La Figura 4.6 riassume come il file EDL viene tradotto dall'edger8r per generare l'enclave, su cui poi dovrà essere eseguita una operazione finale di firma con una chiave RSA per essere caricata in memoria.
Il file EDL deve contenere le definizioni per poter definire il lato dell'applicazione eseguito dall'enclave all'interno del suo spazio operativo e all'esterno: per poter chiamare funzioni della libreria standard, è necessario che vengano importate librerie che siano compatibili con l'enclave, ovvero che non usino chiamate di sistema che contengono vulnerabilità per l'applicazione. Per questo, vengono definiti due blocchi di codice, *fidato* e *non fidato*, che rappresentano rispettivamente la sezione di codice relativo allo spazio interno dell'enclave e allo spazio che è esterno all'enclave. All'interno di questi blocchi, il cui inizio e fine sono indicati dalle parentesi graffe, devono essere indicati i prototipi delle funzioni ECall (nel blocco fidato) o OCall (nel blocco non fidato) e le librerie necessarie per eseguirle.
Quando la scrittura del file è completata dal modulo apposito del framework che separa automaticamente i contesti di esecuzione grazie alle notazioni dello sviluppatore, deve essere tradotto dallo strumento *sgx_ edger8r* fornito da Intel: leggendo le definizioni contenute all'interno dei blocchi, il traduttore genera il file appartenente al contesto non protetto e il file appartenente al contesto protetto, differenziandoli con il suffisso "_u" per il codice appartenente al blocco non fidato e il suffisso "_t" per quello fidato. All'interno di questi file non sono contenuti il codice delle funzioni

[3] https://download.01.org/intel-sgx/latest/linux-latest/docs/Intel_SGX_Developer_Reference_Linux_2.17_Open_Source.pdf

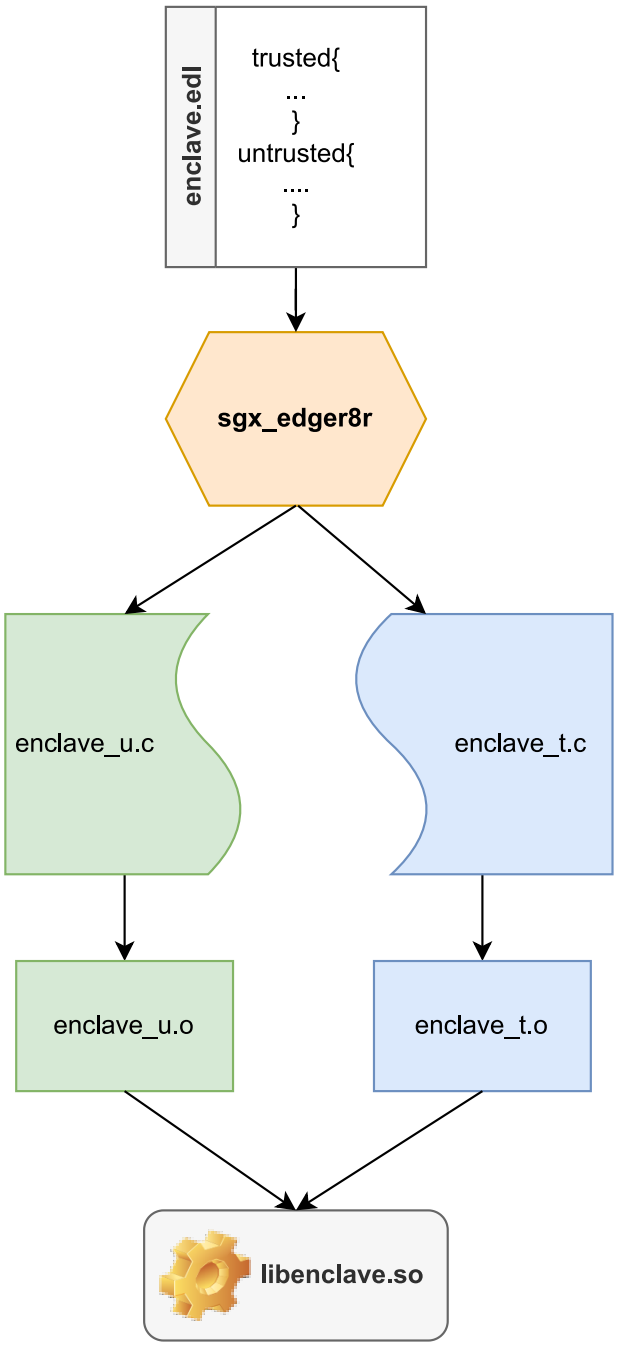


Figura 4.6: Traduzione compiuta dall'edger8r sul file EDL.

e i dati, ma le chiamate alle funzioni proxy che permettono di cambiare contesto di esecuzione e indicare, tramite puntatori, la funzione da eseguire, gestendo eventuali errori e il cambio degli indirizzi passando da una memoria protetta a quella esposta e viceversa. Questi file sono letti dal compilatore per creare i rispettivi file oggetto, fondamentali per la generazione del file che rappresenta l'immagine dell'enclave che sarà caricata in memoria, *libenclave.so*. Prima che sia caricato in memoria dall'applicazione in stato di esecuzione, sarà fondamentale firmare l'enclave con una chiave privata che può essere fornita o generata, ma questo processo è indipendente dalla generazione delle funzioni proxy. Il modulo *edlFileWriter* è una interfaccia creata

```
 #define sgx_ecall_compute_message ([message, i, 1024])
int compute_message(char* message, int nchars, int width);

 #define sgx_ocall_putchar_ocall ()
int putchar_ocall(char c);


enclave{
   include "sgx_trts.h"
   include "sgx_utils.h"

   trusted {
       public int compute_message_trusted([in, count = 1024]
           char* message, int nchars, int width);
   };

   untrusted {
       int putchar_ocall(char c);

   };
};
```

Figura 4.7: Esempio di generazione di un file EDL del framework

proprio per popolare l'EDL, usando le funzioni riconosciute all'inizio dell'analisi di codeAnalyser e inserendo all'interno del blocco di codice fidato o esposto.

Nella Figura 4.7 viene mostrato un esempio di un file EDL, partendo dalla definizione dei prototipi delle funzioni con le notazioni aggiunte dallo sviluppatore che non sono comprese nel file ma appartengono al sorgente dell'applicazione che verrà convertito. La funzione compute_message è stata definita come una ECall, e perciò il framework si occupa di generare la funzione proxy compute_message_trusted, usata per far entrare l'applicazione nel contesto protetto dell'enclave; la funzione putchar_ocall è invece definita come una OCall che riceve un carattere e lo stampa fuori dall'enclave (questo perché printf non è una funzione supportata da SGX per motivi di sicurezza legati all'I/O e per essere eseguita bisogna uscire dall'enclave e chiamarla come una funzione normale). Il file EDL consiste nella definizione della variabile MAXMSG, e in un blocco *enclave* che all'interno delle parentesi graffe conterrà i file header da includere, altri file EDL da importare (in questo caso nessuno), le strutture dati degli argomenti passati dalle funzioni (in questo caso, sono tutti tipi elementari), e due blocchi di codice relativi *trusted* e *untrusted* rispettivamente alla sezione di codice protetto e alla sezione di codice esposto. Nel primo blocco vediamo la definizione del prototipo di compute_message_trusted, con il vettore di dimensione statica message definito con la modalità

di trasferimento dei dati "in" (vedesi Sezione 4.1.1) per indicare che sarà copiato il suo contenuto in un buffer della memoria protetta quando verrà eseguito il cambio di contesto. Il parametro count indica la dimensione del buffer che dovrà essere allocato per il contenuto del vettore, in maniera coerente alla dichiarazione usata dallo sviluppatore. Nel secondo vediamo il prototipo della funzione che diventerà una OCall, e non c'è bisogno di definire una modalità di trasferimento dei dati.

Termine conversione

Una volta compiuto anche questo passo, viene generato per l'enclave un file *enclave_manager* per fare in modo che l'applicazione possa invocare i metodi per la gestione dell'enclave: le funzioni definite automaticamente dal framework permettono il caricamento dell'enclave in memoria durante l'applicazione e la sua rimozione. Tutti i file sono stati creati dentro la cartella generated_trusted, ed è possibile lanciare la compilazione da terminale (se su sistemi Unix) o con Visual Studio (se si usa Windows).

4.2 Moduli del framework

In questa Sezione, verranno presentati tutti i moduli che compongono il framework SGX.

4.2.1 cFileWriterReader

Questo modulo si occupa della lettura del file sorgente .c o del file sorgente .h che il framework deve convertire e della gestione dei suoi contenuti testuali. Il suo scopo è fornire l'accesso diretto alle strutture dati presenti all'interno del sorgente dell'applicazione, come le costanti definite, il codice implementativo che dovrà necessariamente essere modificato per renderlo compatibile con le funzionalità SGX richieste dallo sviluppatore, e le funzioni che dovranno essere modificate. Il file sorgente su cui opererà verrà trasformato quindi in un oggetto di questa classe, e quando le modifiche da fare saranno terminate, lo si salverà in un nuovo file. Le operazioni di cui si occupa il modulo sono:

- la parsificazione delle strutture usate nel codice;
- la modifica del codice sorgente;

Modifica codice

Il codice che viene letto e salvato in un attributo interno della classe può essere modificato in maniera flessibile: esistono funzioni che permettono di aggiungere codice all'inizio del testo, comodo nei casi in cui si vuole includere una nuova libreria, o di aggiungerlo alla fine del codice, come una funzione. La modifica può

essere parziale, ossia eseguita su un blocco di codice che viene sostituito da un altro, o totale, sostituendo una intera funzione con un'altra. La capacità di riscrittura del codice serve in queste operazioni:

1. trovare le funzioni SGX attraverso le macro, rimuoverle e ritornare un vettore di definizioni SGX;
2. rimuovere i commenti nel codice;

Inoltre, il modulo è incaricato di eseguire operazioni di ricerca delle annotazioni di attestazione client e server: il codice viene letto e si inserisce ogni occorrenza trovata in un vettore, ma se alla fine della lettura il vettore non ha elementi, allora il processo viene abortito e il flag che segnala una richiesta di attestazione remota non viene modificato. Se invece il vettore con le occorrenze trovate non è vuoto, si controlla il codice delle funzioni: per ogni funzione che contiene la notazione, si aggiunge quella funzione ad un vettore *functions* che conterrà tutte le funzioni che contengono nel codice la notazione dello sviluppatore. Spostandosi dalla ricerca nel codice di qualsiasi notazione alla ricerca dell'oggetto funzione che la contiene, si possono trovare con esattezza le strutture da modificare e restituirle all'interfaccia.

4.2.2 Function

Questo modulo serve a rappresentare come classe una funzione presente nel codice. Esso presenta infatti gli attributi tipici di una funzione:

- un attributo che indica se la funzione è statica o no;
- il tipo di ritorno della funzione;
- il nome della funzione;
- gli argomenti che sono passati;
- la directory del file in cui la funzione è definita;
- il codice di implementazione della funzione, definito dentro le parentesi graffe;
- il codice completo della funzione, partendo dal tipo di ritorno;
- quando richiesto, una lista delle coppie che hanno come funzione chiamante la funzione stessa e come funzione chiamata la funzione che è raggiungibile da questa;
- le funzioni che vengono chiamate nel codice.

Questi ultimi due attributi servono durante la generazione del grafo delle chiamate effettuato con Frama-C (Sezione 4.3.2). In questo modo, è possibile avere una struttura che possieda tutte le caratteristiche delle funzioni di un file sorgente, poterle modificare con enorme facilità e creare nuove funzioni che abbiano attributi simili. Moduli come l'interfaccia di Frama-C importano Function per sostituire

facilmente le notazioni e sezioni di codice, accedendo al codice implementativo di un oggetto funzione che è stato definito, creando nuove funzioni di attestazione da inserire nel dizionario del sorgente. Anche il modulo di gestione del file sorgente usa frequentemente la classe Function, accedendo al codice di implementazione per cercare le notazioni dello sviluppatore, e con il semplice nome usato come chiave permette di avere restituita l'intera struttura. Sono anche riservati due attributi che indicano a quale enclave, nel caso si trattasse di una funzione ECall/OCall, la funzione appartenga e la definizione dei suoi argomenti nel prototipo inserito nel file EDL.

4.2.3 ctagsInterface

L'interfaccia ad Universal C-tags lancia dalla shell processi che chiamano C-tags con diverse opzioni in base al metodo invocato, eliminando i caratteri di tabulazione non necessari:

- trova tutte le variabili del programma, e controlla con espressioni regolari quali sono quelle globali, ritornandole;
- trova tutte le funzioni statiche con la lettura del codice sorgente, restituendo un output nella sintassi 'nome_funzione' : 'path del file', '(argomenti passati...)';
- trova tutti i nomi che appartengono alla stessa struttura dati, che deve essere specificata come argomento passato.

Questa interfaccia è invocata all'inizio della conversione del programma, in modo tale da controllare che non ci siano due o più variabili che abbiano lo stesso nome. Inoltre, risulta utile avere una lista delle variabili globali quando si deve verificare se una variabile da spostare all'interno dell'enclave ha una visibilità globale.

4.2.4 framaInterface

Questo è il modulo che lancia dalla shell il processo di Frama-C, per generare grafi di chiamate a funzioni. Lo scopo di Frama-C nel framework è quello di aiutare a ricostruire la lista di funzioni in cui l'applicazione entra, partendo da una funzione iniziale, e verificare se esiste un legame di percorrenza tra due funzioni. In una applicazione SGX, una funzione eseguita in una enclave non può dover tornare liberamente in una funzione che è eseguita nel contesto di esecuzione non protetto, e viceversa, essendo questo vietato dal principio di isolamento della memoria. Il modulo Python quindi restituisce i nomi delle funzioni attraversate da un nodo di partenza e verifica che il codice non protetto e il codice protetto non usino una stessa funzione.
Anche il modulo Function usa questa interfaccia per scoprire quali sono le funzioni che sono raggiungibili ricorsivamente dalla funzione di partenza.

4.2.5 gccInterface

Il modulo è usato come interfaccia a GCC. Serve per preprocessare il file, usando un file di configurazione che si trova nei file inclusi del framework, e per avere informazioni su come i dati saranno spostati. I metodi sono due: il primo si occupa del preprocessing del file, rimuovendo tutti le linee di commento che sono presenti; il secondo controlla nelle librerie incluse nella cartella *include* del framework che siano presenti le funzioni del file. Se le funzioni non sono definite negli header SGX che appartengono al framework, allora le funzioni devono essere riscritte per essere eseguite nell'enclave.

4.2.6 edlFileWriter

Questa interfaccia al file EDL è usata dal framework per generare il file "enclave.edl": al suo interno devono esserci i prototipi delle e trasferendo i dati delle funzioni pronte per essere convertite al suo interno. Il modulo gestisce una lista delle funzioni da inserire nel blocco "trusted" e una lista di funzioni per il blocco "untrusted", inizialmente vuota e che sarà riempita aggiungendo le funzioni che hanno come attributo interno la definizione SGX rispettiva. Il modulo contiene un metodo che aggiunge le dichiarazioni di queste funzioni all'interno del testo, e riceve anche dal modulo che gestisce la conversione del file il flag di attestazione remota e un flag che indica se l'enclave deve essere attestata. In questo caso, il modulo importa all'interno della memoria protetta le librerie necessarie alla attestazione remota, aggiungendo dichiarazioni di import dentro il blocco relativo all'enclave. Vengono anche inseriti i prototipi delle funzioni di basso livello di attestazione del SDK Intel, usate dalle funzioni inserite dal framework SGX, che devono essere eseguiti dal codice protetto dell'enclave.

4.2.7 codeAnalyser

Questo modulo è lanciato dal framework e gestisce la conversione chiamando tutti i moduli presentati, registrando le operazioni compiute e i messaggi di errori e debug in un log che raccoglie queste informazioni e il tempo impiegato nel processo. Si occupa di verificare che i moduli esterni siano pronti e installati così come l'SDK SGX, lanciando cTags per avere i nomi di tutte le strutture usate nel codice. Il processo di conversione può essere diviso in diverse fasi, che saranno presentate nella Sezione successiva. Crea il dizionario del sorgente usando il modulo cFileReaderWriter, e controlla se sono presenti notazioni di divisione del codice e di richiesta di inserimento di attestazione remota, salvando il sottoinsieme delle enclavi che devono essere attestate e su cui dovrà inserire le funzionalità. Se non trova nessuna enclave da generare, il framework lancerà una eccezione e interromperà il lavoro di conversione: una applicazione SGX non può esistere senza che sia riservata la zona di memoria dell'enclave, essendo necessaria per l'architettura del protocollo, e lo sviluppatore deve assicurarsi che ci sia almeno una enclave da inserire, anche quando viene spostato il codice su un server, dato che in quel caso si crea una enclave che deve essere attestata per poter eseguire le funzioni remotamente.

L'interfaccia a cFile viene chiamata anche per ottenere tre liste: una per le ECall, una per le OCall e una con tutte le funzioni usate, per poter distinguere quale sarà il codice da salvare dentro le enclavi e quale dovrà rimanere nella parte esposta ad attacchi dell'applicazione. Sono presenti vari metodi per controllare che il codice che si sta provando a convertire e le notazioni e i parametri usati rispettino i vincoli decisi in fase di ideazione. Uno dei controlli svolti è la verifica che i due contesti di esecuzione siano ben separati: una funzione appartenente ad un contesto non può usare una funzione che sia usata anche da una funzione che appartiene al contesto opposto. Ogni controllo di validità non rispettato genera un messaggio di errore, illustrato nel manuale programmatore, che indica con un codice identificativo di una lettera e tre cifre decimali la causa dell'errore avvenuto.
Il framework, dopo aver registrato tutte le modifiche che deve effettuare, ha anche il compito di ripulire il sorgente dell'applicazione per la leggibilità, e cancella tutte le notazioni inserite dal programmatore. Deve generare i file appartenenti all'enclave, creando la cartella e genera al suo interno tutti i file necessari per compilare l'immagine. Infine, salva nella cartella del progetto gli ultimi file tramite operazioni di scrittura su disco dedicati alla compilazione dell'applicazione e dell'enclave.

4.3 Strumenti di analisi statica

Sono usati diversi strumenti di supporto all'interno del framework, che devono essere installati sulla macchina dell'utente prima che inizi la conversione. La loro inclusione serve principalmente nella analisi statica del testo contente il codice del file di input: la lettura dovrà infatti parsificare il codice fornito dall'applicazione per poterlo scomporre in entità, o istanze, che saranno inserite in una struttura dizionario che raccoglie tutte le entità presenti, come le costanti usate, le macro definite, i nomi delle funzioni che sono contenuti, i rispettivi codici di implementazione, i rispettivi tipi di valore ritornato, e altri ancora.

4.3.1 C-tags

C-tags[4] è un framework che consente l'analisi statica di un testo di un file sorgente. In particolare, divide il testo letto in vari token che permettono rapidamente ad un programma di associare valori alle loro classi, usando una rappresentazione testuale. In particolare, il framework usa il supporto esterno di Universal C-tags, una delle versioni disponibili nata da un fork non ufficiale.
La Figura 4.8 rappresenta un esempio di output generato da C-tags nel formato tabulare comprensibile ad una lettura umana[5]: nel primo token abbiamo il nome dell'oggetto identificato, seguito da un token che rappresenta il tipo di oggetto (NBYTES è una costante definita con la direttiva define, main è la funzione di partenza dell'applicazione e data_table una variabile di tipo const dalla dimensione

[4] https://ctags.io/

[5] https://docs.ctags.io/en/latest/output-xref.html#output-xref

fissa). I token successivi rappresentano rispettivamente il numero della linea di codice in cui l'oggetto è stato trovato, il nome del file a cui l'oggetto appartiene e il contenuto intero della linea di codice.

```
NBYTES macro 59 banner.c #define NBYTES 9271
data_table variable 93 banner.c const char data_table[NBYTES]
   = {
main function 1041 banner.c int main(int argc, char *argv[])
```

Figura 4.8: Esempio di output generato con C-tags

C-tags è compatibile con molti linguaggi di programmazione, tra cui ASM, C, C++, Java, Python, R e SQL. Il riconoscimento del linguaggio di programmazione usato è svolto automaticamente dal framework di C-tags.

4.3.2 Frama-C

Frama-C[6] è un framework Open Source per Windows e sistemi Linux estendibile con plugin usato nell'analisi di un software scritto in C. Può essere eseguito lanciando da terminale il comando dedicato o con una interfaccia utente grafica.
Il framework usa Frama-C per effettuare una lettura del testo del programma per generare un file in formato DOT[7] che contiene una rappresentazione per ogni funzione presente del codice della lista delle funzioni che la chiamano. La rappresentazione scelta è quella di un grafo con nodi e archi: i nodi rappresentano le funzioni e gli archi indicano una relazione di chiamata tra i nodi che si trovano alle estremità dell'arco, per la precisione il nodo da cui parte l'arco è la *funzione chiamante* e il nodo che l'arco punta è la *funzione chiamata*. Il grafo delle funzioni è effettuato usando il plugin *Callgraph*.
Frama-C recupera, fornite le stringhe con due nomi di due funzioni dell'applicazione, la lista delle chiamate fatte per arrivare dal nodo che rappresenta la prima funzione al nodo che rappresenta la seconda funzione, usando come funzione di arrivo o una funzione definita dall'utente o la funzione "main". Nella Figura 4.9, è presente un esempio di grafo di chiamata di funzioni generato: a sinistra il codice dell'applicazione che Frama-C legge in input, mentre a destra si trova un grafo che percorre per ogni ramo le chiamate di ogni funzione, rappresentata dal nodo. L'applicazione chiama tre funzioni (main, foo e foo2), e foo2 invoca la funzione print_test per stampare caratteri su standard output. Il grafo ha come nodo di partenza la funzione main, che è stata scelta come punto di ingresso per l'analisi delle chiamate, e per ogni ramo è mostrato quali sono le funzioni successive. *printf_va_1* non è una funzione del sorgente, è una funzione variadica[8] generata dalla funzione di libreria standard printf e inserita in maniera automatica dal plugin

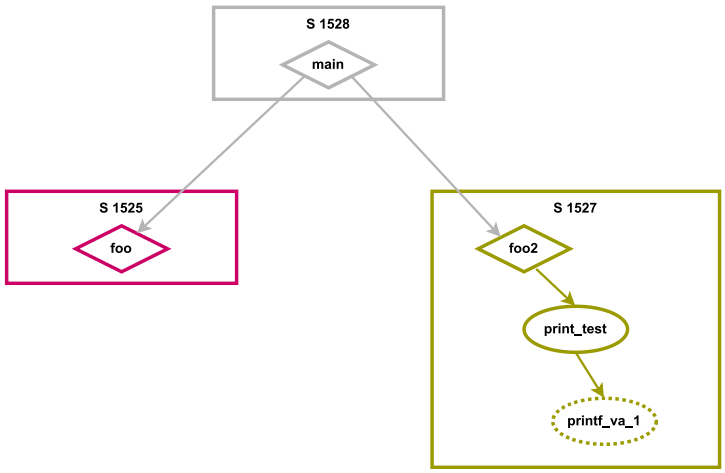
[6] https://Frama-C.com/

[7] https://graphviz.org/doc/info/lang.html

[8] Una funzione variadica è una funzione che ha un numero di argomenti passati variabile.

```
1   void print_test(){
2    printf("test\n");
3   }
4
5   void foo(){
6   ...
7   }
8
9   void foo2(){
10  print_test();
11  }
12  int main()
13  {
14      foo();
15      ...
16      foo2();
17      return 0;
18  }
```

(a) Codice applicazione



(b) Grafo delle chiamate

Figura 4.9: Esempio di programma e grafo delle chiamate generato

Variadic. Questo plugin compie una traduzione di funzioni variadiche in funzioni semanticamente equivalenti ma non variadiche, in modo tale che il funzionamento del codice sia lo stesso e che l'analisi del sorgente modificato sia identica a quella del sorgente originale.
Il grafo generato da Frama-C è nel formato *dot* come risultato intermedio. Deve essere convertito in forma grafica con uno strumento di visualizzazione dei dati.[9].

4.4 Limitazioni del framework

Il framework SGX si prefigge come scopo principale quello di creare un contesto di Trusted Execution Environment per una applicazione convertendo i file secondo le richieste dello sviluppatore. L'architettura di una applicazione SGX si basa su regole molto strette dal punto di vista della flessibilità, poiché per garantire la protezione delle enclavi è necessario non lasciare la completa libertà allo sviluppatore, che potrebbe intenzionalmente o non usare del codice che porterebbe ad eventuali vulnerabilità. Il framework non ha solo limitazioni sul numero di enclavi generabili, ma chiede di dover modificare il codice dell'applicazione per fare in modo che sia valido al modello di SGX, che impone alcuni vincoli alle funzionalità eseguibili.

[9]Nel caso del framework, viene usato il modulo *pydot*.

Queste limitazioni devono essere considerate quando si vuole applicare la conversione di un sorgente, altrimenti il framework stesso o il compilatore segnaleranno gli eventuali problemi allo sviluppatore.

Le limitazioni del codice dell'applicazione sono le seguenti:

1. a causa del principio di isolamento della memoria dell'enclave, al suo interno è possibile solo usare funzioni definite nel codice dell'enclave, funzioni presenti nelle librerie standard SGX o funzioni OCall, e per eseguire una funzione non appartenente al contesto protetto bisogna invocarla con una OCall;

2. non è possibile definire variabili globali condivise da funzioni ECall e OCall, per il principio di isolamento della memoria protetta dell'enclave da quella non protetta dell'applicazione[10];

3. non è possibile definire funzioni ECall e OCall di tipo statiche, poiché le funzioni statiche hanno una visibilità (o scope) limitata al file oggetto rispettivo, e il codice esposto e protetto sono memorizzati su file diversi, rendendo così non accessibili le funzioni di un file all'altro;

4. il tipo di ritorno delle ECall e delle OCall deve essere per il framework necessariamente un intero "int", anche se le linee guida SGX consigliano l'uso di buffer di copia;

5. tra i tipi degli argomenti delle ECall e delle OCall non è possibile usare i puntatori di dimensione dinamica e strutture dati dal nome ridefinito con la direttiva typedef;

6. la sintassi dei programmi C precedente allo standard C11[58] non è riconosciuta dal framework, perciò è necessario convertire il codice alla sintassi più moderna di ANSI C[59].

Inoltre, non tutte le funzioni della libreria standard del C sono disponibili all'interno del contesto protetto sotto forma di funzione della libreria standard di SGX per motivi di sicurezza[11].

[10] Il principio di isolamento si basa sulla memoria PRM che non può essere letta e modificata da qualsiasi agente software e hardware che non sia il controllore del processore, bloccando ogni tentativo di accesso durante la fase di traduzione degli indirizzi.

[11] Ad esempio, possiamo segnalare le funzioni di input e output, le funzioni che cambiano il livello di privilegio con istruzioni di basso livello come SYSCALL e SYSENTER e istruzioni che possono scatenare interrupt hardware.

Capitolo 5

Descrizione della soluzione

Il Capitolo 4 descrive la versione del framework SGX sviluppata in passato. Nel corso di questa tesi nuove funzionalità sono state sviluppate per il framework: su questa base, è stata aggiunta la funzionalità di attestazione remota e la tecnica di divisione del codice, dimostrando un esempio concreto di applicazione che unisse queste due funzionalità. In questo capitolo, verrà perciò riesaminata l'architettura del framework nella Sezione 5.1.1, dopo che sono state compiute modifiche per poter aggiungere funzionalità al framework. La Sezione 5.1.2 riguarda l'implementazione della divisione del codice e la Sezione 5.1.3 è dedicata alla rimozione della limitazione precedente di una sola enclave per applicazione convertita, mentre nella Sezione 5.1.4 sarà presente una parte dedicata all'implementazione della attestazione remota del framework e una parte in cui sono spiegati a basso livello i messaggi che compongono il protocollo di attestazione remota di una applicazione convertita. I blocchi architetturali usati sono descritti nel Sezione 5.2, indicando anche per i moduli che erano già presenti le eventuali differenze con quelli della versione finale. Infine, nella Sezione 5.3 sono descritte nuove limitazioni del framework che possono essere la base di future modifiche.

5.1 Architettura e workflow

Le modifiche effettuate sul framework non hanno cambiato la struttura originale delle classi e dei file importati: alcuni moduli sono stati modificati e altri sono stati aggiunti per poter fornire nuove funzionalità, tra cui quelle relative alla attestazione remota di SGX. L'implementazione della funzionalità di attestazione remota è basata su una applicazione esempio fornita da Intel e disponibile pubblicamente[1]: tra le modifiche effettuate, le librerie non devono più essere compilate eseguendo uno script, i parametri di configurazione sono passati dall'utente tramite il file di configurazione del framework ed è possibile permettere di avere più valori nelle politiche di approvazione dell'enclave.

[1] https://github.com/intel/sgx-ra-sample

5.1.1 Architettura

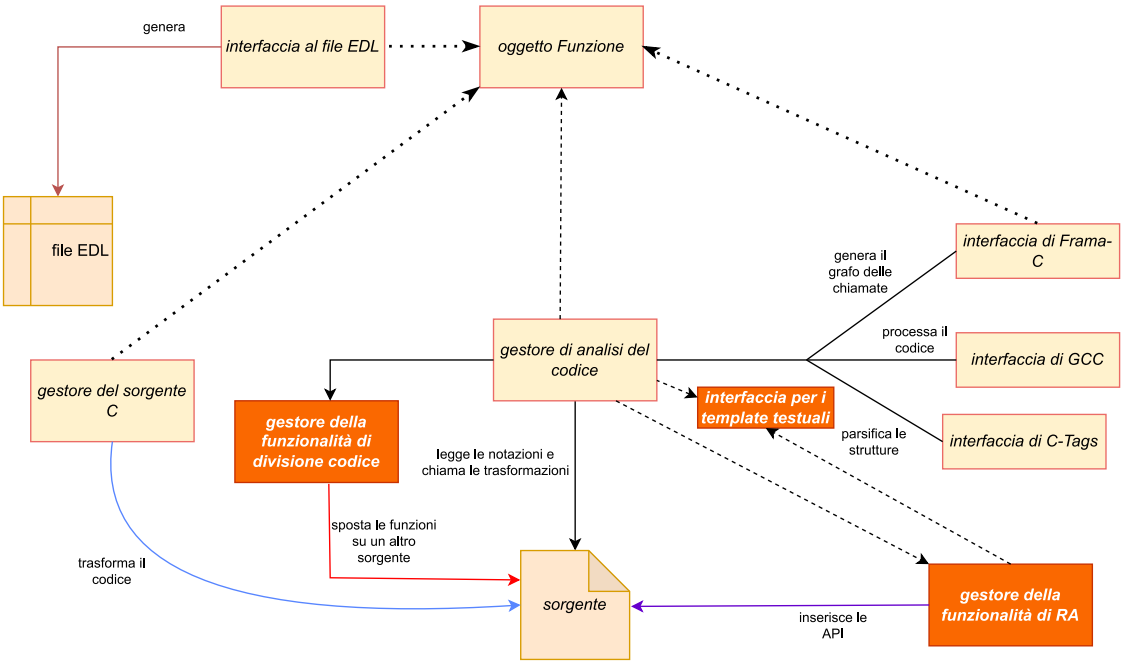


Figura 5.1: Architettura del framework SGX modificato

La Figura 5.1 mostra l'architettura del framework attraverso i moduli che lo compongono e le relazioni tra di essi, con nuove aggiunte rispetto alla versione della Sezione 4.1. Le linee tratteggiate indicano una dipendenza software che chiede l'importazione di quel modulo. I moduli aggiunti sono i seguenti:

- un modulo per gestire la funzione dell'attestazione remota nell'applicazione;
- un modulo per gestire l'applicazione della tecnica di divisione del codice;
- un modulo per contenere tutti i template testuali di grandi dimensioni che devono essere modificati in base alle modifiche da effettuare (enclavi da generare e funzionalità richieste).

Durante lo sviluppo di questo progetto, uno degli obiettivi principali da raggiungere è stata la rimozione di un vincolo precedente importante, la possibilità di creare una sola enclave per la singola applicazione : come conseguenza, le notazioni riconosciute e usate dal framework devono ora rispettare una nuova sintassi, per poter indicare quale enclave tra quelle generate dall'applicazione deve essere associata all'operazione richiesta. Le nuove notazioni sono usate per inserire le funzionalità di attestazione remota (con macro distinte per le librerie che devono essere usate da un client e per quelle che devono essere usate da un server) e per applicare la tecnica di divisione del codice, indicando le funzioni da spostare sul server remoto. Il framework si occupa anche della generazione del contenuto del makefile per poter tradurre tutti i sorgenti dell'applicazione convertita e i sorgenti dell'enclave, indicando al compilatore quali sono i file oggetto da usare come input per creare l'eseguibile finale. Inoltre, compila la libreria dinamica che corrisponde all'immagine

dell'enclave (il file generato con il compilatore è un file "*libenclave.so*"), e chiama uno strumento di utilità *sgx_sign* fornito dall'SDK per eseguire con una chiave privata RSA da 3072 bit, con esponente pubblico pari a 3, in formato PEM[60] una firma sul file per generare una *enclave firmata*.
Il risultato dell'operazione di crittografia asimmetrica è una enclave firmata e pronta per essere caricata dall'applicazione con il nome di "*libenclaveXX.signed.so*", con XX sostituito dall'identificatore dell'enclave. La dimensione e le proprietà dell'enclave seguono le indicazioni forniti nei metadati del file di configurazione XML "enclave.config.xml" che può essere trovato all'interno della corrispettiva cartella con i file relativi, e può essere cambiato con completa libertà dal programmatore. In questo modo è possibile decidere di cambiare le sue proprietà senza ricompilare e riconvertire il sorgente, ma cambiando i parametri del file e eseguendo nuovamente lo script Makefile che ricompilerà soltanto gli oggetti necessari.

5.1.2 Funzionalità di divisione del codice

La tecnica di divisione del codice è stata presentata nella Sezione 3.3: all'interno è stata anticipata la possibilità di poter unire alla tecnica di divisione del codice un meccanismo di attestazione remota con SGX. Il framework è stato modificato in modo tale da poter implementare questa tecnica, e in questa Sezione sarà descritta l'implementazione all'interno del framework, la modalità per segnalare al framework la richiesta di applicazione e il risultato finale dell'operazione di conversione sul codice dell'applicazione.
Quando si desidera effettuare la divisione del codice dell'applicazione, bisogna inserire sopra la linea di codice appartenente all'implementazione della funzione la seguente notazione:

```
#pragma move_to_server
```

In questo modo, si segnala che la funzione che segue la notazione deve essere eseguita remotamente. Il framework sostituirà il codice implementativo di questa funzione con uno nuovo, il cui obiettivo è preparare l'applicazione a comunicare con il server e chiedere l'esecuzione di una specifica funzione (l'implementazione è lasciata per il Capitolo del Manuale Programmatore).
In Figura 5.2 è mostrata la sequenza di operazioni del framework sull'applicazione quando viene applicata questa tecnica di protezione del codice ad alto livello.

Il processo di conversione, descritto a sinistra, parte dal riconoscimento delle funzioni attraverso la notazione apposita che segnala le funzioni su cui è stato richiesto il trasferimento. Per assicurarsi che la loro esecuzione remota sia eseguita correttamente, il framework pone vincoli sulle funzioni che devono essere trasferite: queste funzioni sono definite come *funzioni entrocontenute*. Una funzione è entrocontenuta se rispetta le seguenti condizioni:

- non deve usare variabili globali;
- i parametri ricevuti come argomenti non devono essere passati per riferimento ma solo per valore;

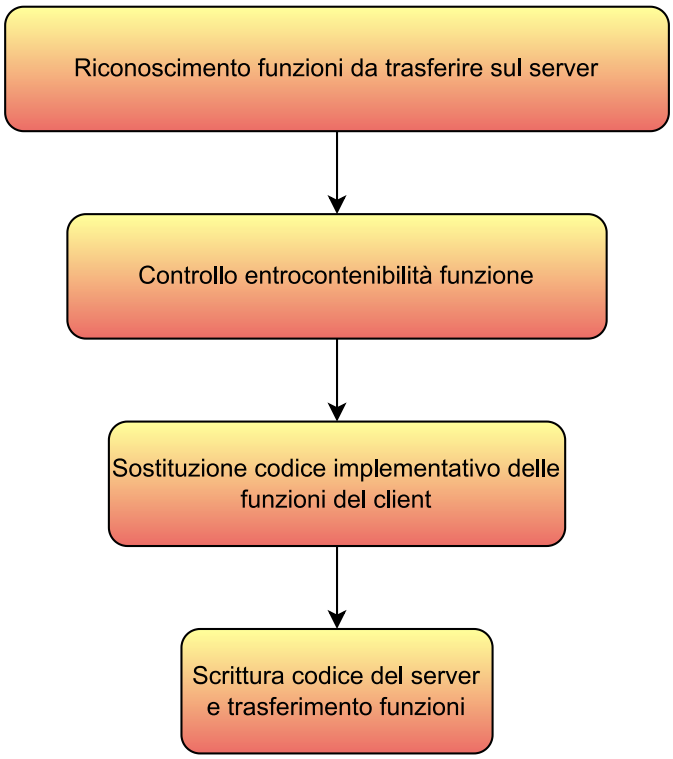


Figura 5.2: Operazioni svolte dal framework

- deve usare tipi di dati elementari, quindi int, char, float e double.

Dato che la funzione dovrà essere eseguita su un ambiente remoto, per permettere di accedere a dati localizzati su indirizzi della memoria della macchina del client come se l'applicazione venisse eseguita nello stesso ambiente bisognerebbe eseguire una traduzione degli indirizzi. La traduzione non è semplice tuttavia da progettare e implementare, e per questo motivo è stata inserita la limitazione di poter convertire soltanto funzioni che devono essere entrocontenute. Dopo la verifica delle condizioni di entrocontenibilità, il framework esegue la sostituzione dell'implementazione delle funzioni: in questa operazione si inserisce nell'applicazione una logica di serializzazione dei parametri della funzione che devono essere inviati al server remoto, che sarà scritto in modo tale da sapere quanti dati dovrà ricevere dal client per eseguire una determinata funzione richiesta. Soltanto eseguendo una analisi statica del codice dell'applicazione originale è possibile poi generare il codice di un server: il codice delle funzioni trasferite viene copiato all'interno del sorgente, associando ad ogni funzione un identificativo che servirà a selezionare quella richiesta dal client che apre una connessione inviando insieme ai dati anche l'identificativo. Nei sorgenti delle due applicazioni è anche inserito il codice che si occupa della deserializzazione dei dati ricevuti, in modo tale che il server possa leggere i parametri che riceve quando la connessione TCP è aperta e il client possa conoscere il risultato dell'elaborazione remota richiesta.
Le funzioni su cui sono eseguite le modifiche vengono individuate mediante notazioni inserite prima della dichiarazione della funzione dallo sviluppatore nella fase precedente al lancio del framework: queste funzioni vengono raccolte in una lista,

e saranno sostituite all'interno di un dizionario che contiene tutte le strutture usate dal file sorgente convertito. Usando il nome di queste funzioni come chiave di ricerca, dal dizionario viene estratto l'oggetto che equivale alla funzione con quel nome, e per ognuno di essi il framework verifica che la funzione corrispondente sia entrocontenuta, affinché sia eseguibile correttamente in un ambiente remoto. All'apertura di ogni connessione, l'applicazione server generata dal framework esegue un processo di attestazione remota per confermare l'identità del client che si è connesso: le librerie di attestazione e la modifica del codice sorgente per inserire le chiamate sono tuttavia gestite nella fase di gestione dell'attestazione remota.

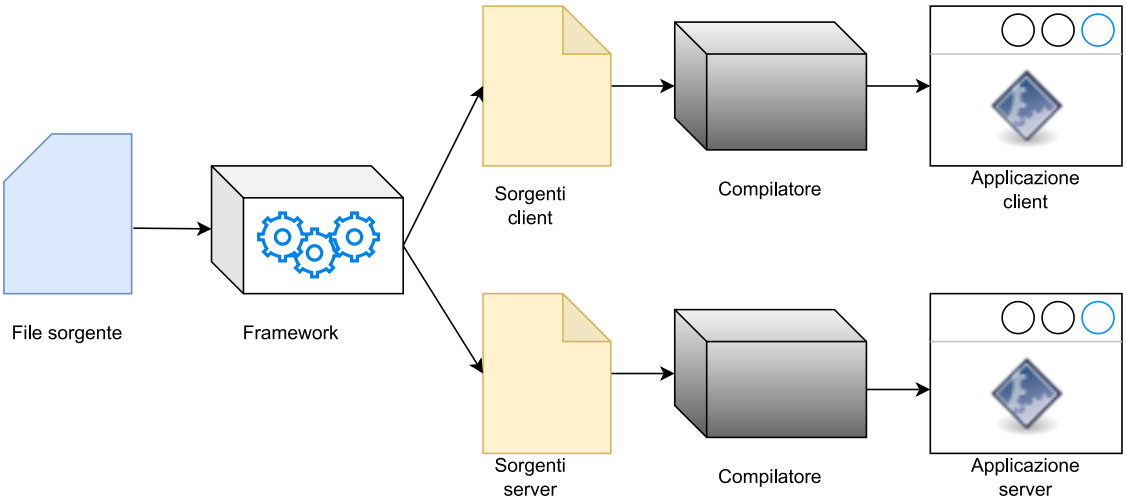


Figura 5.3: Flusso di generazione di una applicazione

Nella Figura 5.3 invece viene mostrata l'applicazione della divisione del codice dal punto di vista dell'utente che usa il framework e richiede la tecnica: il framework riceve un file sorgente dove vi sono inserite le rispettive notazioni e genererà due sorgenti diversi, uno appartenente all'applicazione lato client e l'altro all'applicazione lato server. Tutte e due i sorgenti dovranno quindi essere compilati, con l'uso dello script Makefile fornito appositamente dal framework, per poter ottenere due applicazioni distinte che dovranno essere eseguite parallelamente per poter operare.

5.1.3 Rimozione del numero massimo di enclavi

Lo sviluppatore deve poter essere capace di poter usare più enclavi che possano essere indipendenti tra di loro, e poter assegnare a ciascuna di essi dati e funzioni separate. Questa flessibilità permette di poter ampliare le potenzialità dell'applicazione: ad esempio, si può assegnare ad una enclave l'esecuzione di alcune funzioni dell'applicazione e ad un'altra il compito di attestare il dispositivo di Intel o le altre enclavi presenti tramite un processo di attestazione che può essere locale, e all'interno di queste enclavi inserire invece i dati più sensibili.
Per conoscere il numero di enclavi che il framework deve creare, tramite il processo di conversione dell'applicazione e traduzione del file EDL da generare, è stato scelto di usare una diversa sintassi per le notazioni da scrivere nel codice, tramite un

prefisso e un suffisso che distinguono le enclavi da creare. Quando si definiscono funzioni ECall e OCall, o si dichiara che alcuni dati devono essere trasferiti/copiati, si devono legare queste modifiche alla rispettiva enclave: il framework conta quanti sono gli indici diversi utilizzati, e per ogni indice riconosciuto esegue le modifiche richieste, creando diverse cartelle con i file che appartengono a quella enclave. Le notazioni dedicate alla definizione delle funzioni in funzioni ECall e OCall devono rispettare la sintassi della Figura 5.4: al posto del carattere del cancelletto si inserisce un carattere numerico come identificativo dell'enclave che si vuole associare. Gli identificativi da usare per più enclavi non devono essere numeri consecutivi, ma servono solo a distinguerle l'una dall'altra.

```
#define sgx_E#_ecall_<nome_funzione> (<[nome_param>,
   <tipo_copia>, <dim_buf>]..)

#define sgx_E#_ocall_<nome_funzione> (<[nome_param>,
   <tipo_copia>, <dim_buf>]..)
```

Figura 5.4: Sintassi delle notazioni per definire una Ecall e una Ocall

In Figura 5.5 sono presenti le notazioni che permettono di copiare dei dati all'interno di una enclave o di trasferirli completamente, seguendo lo schema già presentato nella Sezione 4.1.1. Per l'operazione di movimento dei dati e per l'operazione di copia nella zona protetta della memoria dell'applicazione, le due notazioni conservano lo stesso meccanismo, con unica differenza la scelta di usare la sigla per indicare l'enclave in cui i dati saranno spostati come suffisso della macro.

```
#pragma move_start_E#
    ...
#pragma move_end_E#

#pragma copy_start_E#
    ...
#pragma copy_end_E#
```

Figura 5.5: Sintassi delle notazioni per definire la copia e movimento dei dati in una enclave

Il framework si occupa anche di generare metadati diversi per le enclavi, che non possono essere generate con gli stessi parametri, e di generare un makefile che compili tutte le enclavi e le firmi con la stessa chiave.

5.1.4 Funzionalità di attestazione remota

Il framework include la possibilità di verificare un' enclave usando la funzionalità di attestazione remota descritta nel Capitolo 3. L'attestazione remota richiede

interazione tra una applicazione client e una applicazione server. L'applicazione da convertire dovrà quindi importare e usare le librerie di attestazione del client se contiene una enclave da attestare, mentre nel caso lo sviluppatore abbia una applicazione server che fornisce un servizio e che deve controllare l'integrità del client che si connette e con cui scambia dati, allora bisogna che utilizzi le librerie di attestazione dedicate ad un server. Come già dichiarato, grazie al vincolo sul numero di enclavi di una applicazione rimosso si può implementare la funzionalità di attestazione remota per più enclavi.
L'attestazione remota va segnalata usando una notazione che indichi al framework se si desidera che l'applicazione chieda l'attestazione oppure se si desidera inserire funzionalità di verificazione dell'attestazione. Nel primo caso, lo sviluppatore deve inserire, nel punto dove desidera che il processo di verifica inizi, la notazione:

```
#pragma attest_enclave_XX
```

La direttiva #pragma serve allo sviluppatore per indicare la richiesta di una operazione specificata con le parole chiave successive. Quando il framework cerca annotazioni durante la lettura statica del codice, verifica sempre che ci sia questa parola che distingui il testo da quello del codice dell'applicazione.
"*attest_enclave_XX*" è la funzione che chiede di inserire all'interno dell'enclave una API che contatti un server per procedere all'attestazione. XX è un placeholder che deve essere sostituito con l'identificativo dell'enclave da verificare: è lo sviluppatore a dover indicare quale sia quella da attestare. Per indicare invece che l'applicazione esegua le librerie di attestazione di un server si usa la notazione:

```
#pragma SERVER_ATTESTATION
```

Esattamente come per la notazione del client, questa notazione è usata nel punto dell'applicazione in cui si desidera venga interrotto il flusso di esecuzione standard per poter aspettare che un client si colleghi alla porta su cui il server aspetta in ascolto e inizi il processo di attestazione remota. Non è presente un suffisso che indichi l'identificativo di una enclave, dato che il server non necessita di entrare in un contesto protetto durante il processo di attestazione.

Requisiti server

Il server necessita di una preparazione iniziale per poter contattare la piattaforma Intel tramite API. Lo sviluppatore dell'applicazione server deve iscriversi al sito Intel Developer Zone[2] creando un account. Una volta che la registrazione è completata, dal portale è possibile scaricare il certificato di Attestation Report Root Certificate Authority [3] in formato PEM che serve a controllare che le risposte ricevute dall'IAS siano accompagnate da una firma generata con una chiave valida

[2] https://api.portal.trustedservices.intel.com/

[3] L'Autorità Certificativa, o Certificate Authority (CA), è un ente terzo come una organizzazione o una compagnia reputata affidabile che si occupa della gestione, del rilascio e della revoca dei certificati che legano l'identità digitale di una entità ad una coppia di chiavi asimmetriche.

appartenente a Intel, controllando ricorsivamente la lista di CA indicate dal certificato. Tra i dati che sono disponibili iscrivendosi al portale, ci sono il Service Provider ID, o *SPID*, che consiste in una stringa da 16 Byte che deve essere inserita nella quote mandata alla piattaforma web Intel: lo SPID è l'identificativo unico del server (e di chi lo gestisce) che vuole interfacciarsi all'IAS e deve essere fornito per autenticarsi. Gli altri dati disponibili dopo l'iscrizione sono due chiavi di sottoscrizione da inserire nell'header della richiesta: queste sono rispettivamente la *chiave primaria di sottoscrizione* e la *chiave secondaria di sottoscrizione*, e servono al gestore del servizio server per autenticarsi al servizio di attestazione web. In caso la chiave primaria non sia trovata dal servizio Intel nell'header della richiesta HTTP o non sia riconosciuta come dato corretto di autenticazione, si riprova l'autenticazione usando la chiave secondaria: se anche questa chiave non è giudicata corretta, allora la connessione viene automaticamente interrotta e non è possibile la verifica dell'attestazione dell'enclave. Le chiavi di sottoscrizioni e lo SPID sono ugualmente necessari per il processo di autenticazione e devono essere inseriti nel file di configurazione del server, che li userà all'avvio del processo di attestazione, quando carica tutti le informazioni necessarie, compresi i valori che devono essere contenuti in una quote di una enclave ricevuta per ritenerla attendibile e sicura. Il file di configurazione è chiamato *policy*, e viene generato tramite uno script fornito dal framework (lo script è descritto nel Capitolo del Manuale Programmatore) che permette di effettuare la misura dell'enclave che si vuole attestare. Si può decidere dal file di configurazione se la misura dell'enclave deve essere tra le credenziali che devono essere incluse nel processo di attestazione della quote dell'enclave, o se verificare in base agli altri valori.

Reazione personalizzata del server

Il framework consente ad una applicazione server che contatta l'IAS per verificare l'enclave di poter chiamare una funzione scritta dallo sviluppatore quando l'enclave non supera il processo di attestazione remota. La reazione standard delle librerie inserite dal framework al fallimento del processo è :

- la deallocazione della memoria dinamicamente allocata per creare le strutture dati nei processi;
- la chiusura dei thread di allocazione lanciati;
- la chiusura delle porte usate in modalità di ascolto nella connessione con i client connessi.

Specificando la directory del file sorgente in cui si trova la funzione di reazione personalizzata, il framework cerca il file e lo inserisce nelle librerie dell'applicazione, aggiungendo le regole per compilarlo e linkarlo nell'eseguibile all'interno del file Makefile.

5.2 Modifiche architetturali

In questa Sezione sono presentati i nuovi moduli inseriti nel framework e le differenze dei moduli esistenti con le versioni descritte nel Capitolo 4, senza citare quelli per cui non è servito fare modifiche. Sono stati mantenuti i moduli esterni di Frama-C, Universal C-tags e GCC, e sono stati creati tre nuovi moduli: *constants*, *remoteAttestationInterface* e *codeSplittingInterface*.
Al centro del sistema, è presente "codeAnalyser", il modulo che gestisce il processo di conversione, chiamando gli altri moduli definiti quando serve usare metodi più specifici. La Figura 5.6 indica le dipendenze di codeAnalyser, chiamato dal

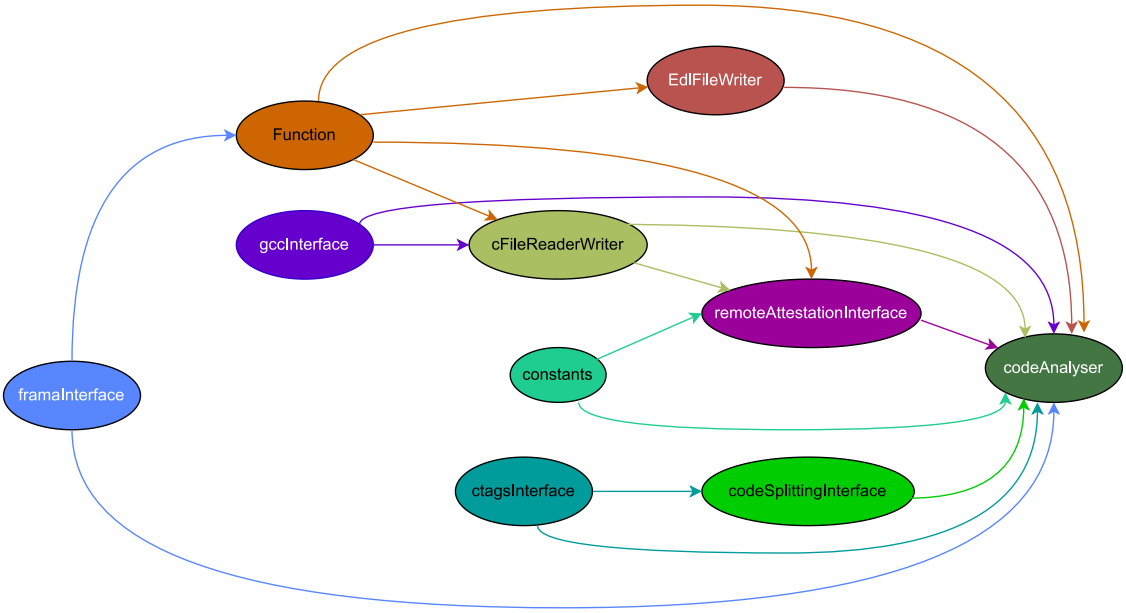


Figura 5.6: Diagramma dei moduli usati dal framework.

framework per la conversione del codice, con gli altri moduli del framework.

5.2.1 Gestore della funzionalità di attestazione remota

Questo modulo è specializzato nella gestione da parte del framework dell'inserimento della funzionalità di attestazione remota nell'applicazione. Il primo compito è quello di svolgere la ricerca di notazioni che richiedono l'intervento nel codice, leggendo il contenuto del dizionario costruito con la classe cFileReaderWriter, e in base al tipo di attestazione richiesta restituisce al modulo di gestione della conversione un valore intero.
Il modulo è pensato per l'inserimento delle funzioni di attestazione dove le notazioni apposite sono state inserite nel codice: indipendentemente dal tipo di applicazione che dovrà essere convertita, il modulo dovrà inserire il codice implementativo delle funzioni che introducono al processo di attestazione. Se si vuole permettere ad un client di connettersi ad un determinato indirizzo, il modulo consente con un altro metodo di inserire l'indirizzo specificato nel file di configurazione nel file "client_starter" al posto del valore "Localhost" che rappresenta la rete dell'utente. Infine, genera anche il Makefile di un server che è stato convertito dal framework inserendo

le librerie di attestazione da compilare.
Inoltre, il gestore dell'attestazione remota si occupa anche di aggiornare il makefile dell'applicazione finale, inserendo le regole aggiuntive per l'attestazione, come quelle dei file oggetto delle librerie usate. Le operazioni dedicate all'inserimento della funzionalità di attestazione remota vengono attivate se al lancio del processo di conversione si trovano nel codice notazioni di attestazione: in tal caso, in base alla notazione riconosciuta, viene settato un flag apposito. Durante il ciclo in cui si generano le diverse enclavi dell'applicazione si inseriscono nel file EDL dell'enclave le funzioni delle librerie di attestazione; contemporaneamente, nel sorgente vengono aggiunte le dichiarazioni di importazione dei file header relativi alle librerie e la chiamata alla funzione che inizia il processo. L'applicazione di attestazione remota originale lavorava sull'attestazione di una singola enclave, mentre il framework consente di poter ampliare la verifica a più enclavi legate ad una singola applicazione.

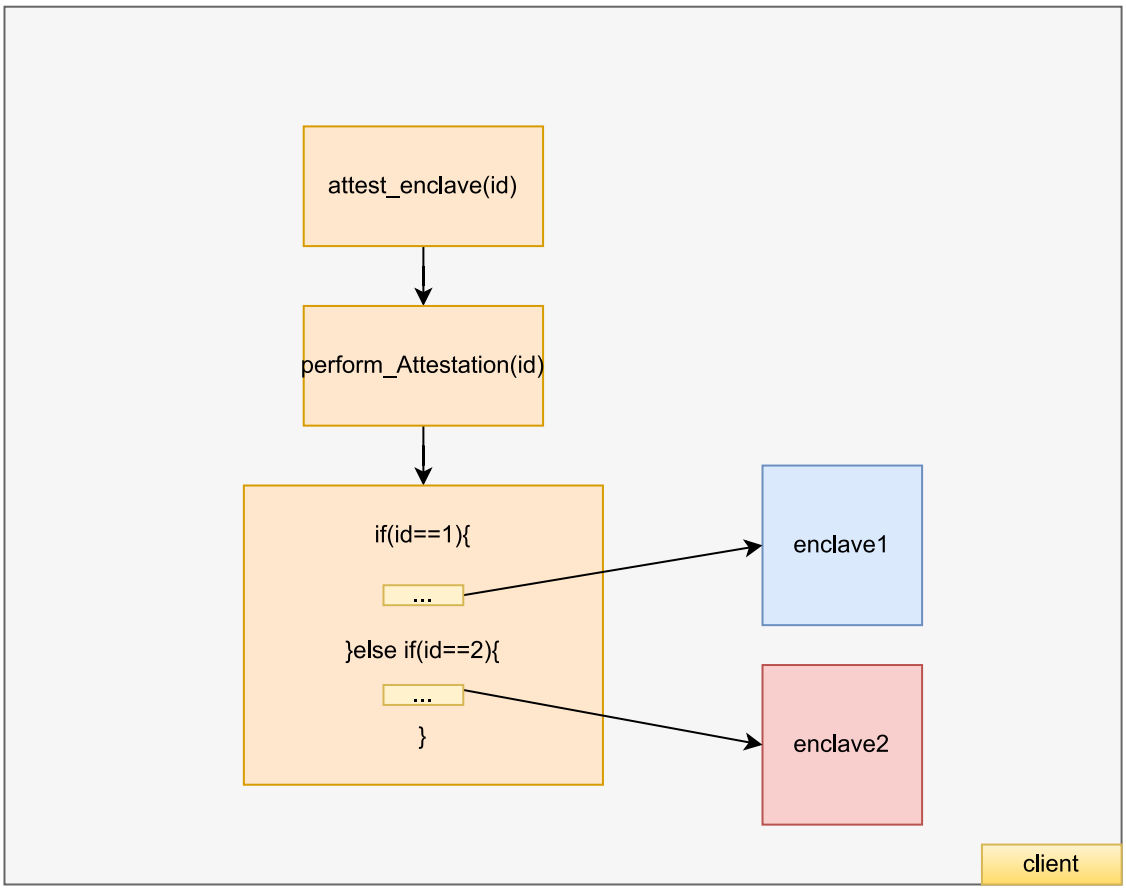


Figura 5.7: Struttura della selezione delle funzioni di attestazioni per enclavi multiple

La Figura 5.7 indica come la funzione che gestisce l'attestazione lato client opera nel caso di una applicazione che deve attestare più enclavi. Ogni enclave viene caricata in memoria e gestita tramite un identificativo: seguendo il protocollo ECDH l'applicazione deve entrare nel contesto protetto dell'enclave e con funzioni di basso livello generare i segreti e altri parametri. Non è possibile usare una ECall definita nel file EDL di una enclave in altre, per il concetto di isolamento che è alla base del

funzionamento di SGX. Il framework quindi, per ognuna delle funzioni di attestazione di alto livello che chiama la funzione rispettiva di basso livello, deve avere una apposita ECall associata esclusivamente alla sua enclave. Per ogni operazione del genere, deve essere costruito quindi uno switch che selezioni in base all'id passato come parametro in quale enclave entrare. In questo modo, si possono anche evitare problemi di name clashing durante la fase di compilazione dell'applicazione.

5.2.2 Gestore della funzionalità di divisione del codice

Il modulo è una interfaccia che codeAnalyser usa per gestire interamente il processo di divisione del codice, spostando funzioni selezionate su un server remoto da contattare ad ogni chiamata (vedesi Sezione 3.3). Con questo modulo, è possibile accedere a lunghe sezioni di codice memorizzate nel formato di stringhe: il loro scopo è contenere il codice delle funzioni che saranno inserite dal framework nell'applicazione, e attraverso modifiche il codice sarà riscritto parzialmente per inserire le funzioni e i dati richiesti per essere spostati remotamente. Alcuni esempi di codice salvato come stringa sono il sorgente del server generato dal framework per eseguire remotamente le funzioni del client e l'implementazione della funzione usata per contattarlo inviandogli i dati necessari. Gran parte delle operazioni svolte all'interno è dedicato all'analisi dei parametri delle funzioni e delle funzioni stesse per convertirle nelle strutture dati apposite (descritte nel Manuale Programmatore). Nei metodi raccolti, sono presenti quelli che servono alla verifica della entrocontenibilità della funzione da spostare sul server. Il framework legge nel codice se esistono funzioni che sono precedute dalla notazione di segnalazione di trasferimento ad un server. In primo luogo, si individua il tipo di parametri usati della funzione spostata e si crea il codice per serializzarli in strutture dati apposite. La funzione *moveFunctionsToServer* è chiamata proprio a questo scopo: viene costruita la lista delle funzioni su cui si opererà, e per ognuna di esse si leggono il numero di argomenti e il tipo di ritorno, in modo tale che si possa costruire una variabile che abbia i parametri coerenti al tipo di funzione che sta rappresentando, salvandoli in liste. Il server dovrà riconoscere quale sarà il codice della funzione che è stata richiesta di essere eseguita, scegliendo il blocco corretto tra quelli che contiene, identificabile da un numero sequenziale associato. Quando il server comunica con il client, il primo dato che riceve è quello dell'identificatore, e in base a esso l'applicazione eseguirà il blocco di codice rispettivo.

5.2.3 Interfaccia ai template testuali

Questo modulo, chiamato internamente al progetto anche come *Constants*, è creato per contenere blocchi di codice che devono essere scritti in nuovi file o devono essere aggiunti in file già esistenti. All'interno del modulo, sono salvati in attributi i seguenti contenuti:

- il codice implementativo di enclave_manager.c e il rispettivo file header;
- la configurazione in formato .xml dell'enclave;

- il codice del file .lds dell'enclave, uno script passato al linker per nascondere simboli non necessari;
- la chiave privata RSA di test per la firma dell'enclave;
- lo scheletro del makefile da eseguire;
- il file header con le funzioni di attestazione remota, e le implementazioni rispettive da inserire nella memoria protetta;
- la funzione che inizia il processo di attestazione remota del client, chiamando l'interfaccia "client_starter";
- i template per costruire il makefile.

Il makefile di una applicazione che possiede più enclavi dovrà, per ogni enclave creata, contenere le regole di definizione dei file dell'enclave. Non è possibile sapere a priori quante regole dovranno essere inserite, quindi il framework usa un template che viene aggiornato automaticamente. La scrittura del Makefile è eseguita tramite due funzioni che si differenziano in base al tipo di Makefile che deve essere scritto: la prima versione non comprende le regole che definiscono i file oggetto delle librerie di attestazione remota, mentre la seconda deve definire, in base alla notazione di attestazione remota inserita, le regole per linkare le librerie del client, del server o di entrambe. Il modulo Constants usa inoltre un template simile per definire i file "enclave_u".h e .c che contengono rispettivamente i prototipi e le implementazioni dei proxy esposti e delle funzioni ponte esposte. Viene usato lo stesso principio di sostituzione del carattere placeholder e di iterazione per generare i file che devono essere importati nel file dell'applicazione convertita.

5.2.4 Moduli presenti

Interfaccia al file EDL

Il modulo di interfaccia al file EDL dell'enclave è stato modificato, in modo tale da poter essere compatibile con la funzionalità di attestazione remota e con la generazione di enclave multiple. Avere più enclavi da generare significa infatti dover generare diverse enclavi per funzionalità, e deve poter essere permesso di identificarle distintamente. Il modulo si occupa di assegnare ad ogni enclave un id diverso che sarà segnalato nel rispettivo file di configurazione XML. Quando il gestore del processo di conversione segnala che una determinata enclave dovrà eseguire il processo di attestazione[4], l'interfaccia inserisce i prototipi delle funzioni dedicate al processo di attestazione remota appartenenti alla libreria SGX e i file EDL che contengono i rispettivi delle funzioni di più basso livello. Dato che applicare la divisione del codice su un server remoto implica la presenza di un processo di attestazione remota, verranno inseriti questi file anche in questo caso, e verrà scelta una enclave generica come l'enclave che chiede l'attestazione.

[4]La segnalazione è svolta passando un parametro come argomento della funzione che scrive il file EDL sul disco.

Gestore del sorgente C

Nel modulo sono stati aggiunti nuovi metodi che operano sul testo del sorgente C. Le prime funzioni inserite sono quelle che cercano le notazioni di attestazione remota all'interno del file, restituendo una lista che contiene le funzioni in cui verranno inseriti i blocchi di codice che chiamano le funzioni di attestazione. Un'altra aggiunta del modulo è l'analisi del numero di enclavi dell'applicazione, svolta all'inizio del processo di conversione: vengono cercate tutte le notazioni che definiscono i dati da spostare e che definiscono le ECall/OCall di una enclave, viene preso il suffisso finale relativo all'identificativo dell'enclave e lo si inserisce in un set, una struttura dati che ha la caratteristica di dover avere elementi tutti diversi. Il numero di elementi distinti indica quante sono le enclavi da creare nell'applicazione finale.

Gestore dell'analisi del codice

Questo modulo è stato modificato per poter coordinare le nuove funzionalità. Durante l'analisi statica, identifica le ricerche per le nuove notazioni, e se trova occorrenze per ognuna delle funzionalità inserite, lancia i rispettivi metodi. Un'altra modifica svolta riguarda il processo di generazione dell'enclave: il modulo trova il numero di enclavi massime che sono state richieste, leggendo le notazioni dedicate alla definizione delle funzioni protette da SGX, e compierà un ciclo in cui saranno raccolte per ogni enclave riconosciuta tutte le ECall, le OCall, i dati da trasferire o copiare, e le funzioni di attestazione remota. Il modulo registrerà queste modifiche, genererà i file dell'enclave nella rispettiva cartella, e ripeterà questo processo per tutte le enclavi che sono state richieste.
Un'altra modifica del modulo riguarda la verifica che il sistema operativo usato sia Linux: alla fine del processo di conversione, il framework verifica l'ambiente di esecuzione del sistema, e se è stato lanciato su Linux fornisce anche i file necessari alla compilazione dell'applicazione. Il contenuto dello script è ottenuto partendo da uno scheletro di codice pieno di valori provvisori definito nel modulo Constants, aggiornandolo con i file relativi al progetto in base ai flag che sono stati attivati precedentemente: ad esempio, se il flag di attestazione remota è stato settato, inserirà nel Makefile le regole per tradurre le librerie di attestazione remota nei rispettivi file oggetto da linkare per creare l'applicazione finale. Le librerie di attestazione contengono al loro interno valori di default che devono essere sovrascritti dai parametri dello sviluppatore, come le credenziali che il server usa quando deve collegarsi all'IAS, e i parametri del file frameworkConfig letti al lancio sono quindi usati per aggiornare l'applicazione automaticamente ai bisogni dello sviluppatore, senza che debba inserire manualmente i valori nel codice delle librerie. Terminata la costruzione del codice dello script e la scrittura sul disco, vengono copiati i file relativi alla reazione personalizzata di una attestazione fallita (quando l'applicazione convertita funge da server remoto) e uno script che lancia la compilazione automaticamente se l'opzione apposita è stata attivata dal file di configurazione.

5.3 Limitazioni del framework

Il framework SGX si prefigge come scopo principale quello di creare un contesto di Trusted Execution Environment per una applicazione convertendo i file secondo le richieste dello sviluppatore. Le implementazioni che sono state aggiunte non hanno alleggerito le limitazioni imposte sul codice da convertire, che deve continuare a rispettare le limitazioni già presenti: è vietato l'uso di variabili globali, la definizione delle ECall e OCall come funzioni di tipo statico e che devono necessariamente ritornare un tipo "int", l'uso delle funzioni di librerie standard limitato soltanto a quelle ridefinite da Intel. Le modifiche effettuate hanno permesso la rimozione dei vincoli sull'esecuzione dell'applicazione, e successivamente di lavorare su una implementazione di funzionalità di attestazione per applicazioni più complesse.

5.3.1 Attestazione single-thread

Il server compilato dopo aver applicato la tecnica di divisione del codice e le librerie di attestazione lavorano con i messaggi che scambiano le informazioni sulla sessione tra le due entità e sull'identità dell'enclave che ha richiesto l'attestazione. Se l'applicazione crea più thread, e per ciascun thread viene richiesta l'attestazione di una enclave diversa, o anche solo della stessa enclave, il server di attestazione completa la prima richiesta effettuata per poi passare alla successiva.
I messaggi di attestazione vengono infatti scambiati all'interno di un socket a cui si connettono due host. Uno di questi funge da server, e l'altro da client, e entrambi usano funzioni di invio e ricezione dati per lo scambio di dati sotto forma di byte che sono trasferiti attraverso questo socket. Con più thread che svolgono processi di attestazione contemporaneamente, il socket assume comportamenti anomali dovuti alla race condition[5] che si crea tra i diversi host. Supponendo che ci siano due processi attivi contemporaneamente, e che entrambi, con un leggero intervallo temporale di differenza, usano una funzione di invio dati sullo stesso socket: il thread che termina per primo e che chiude la connessione usata influirà sul thread rimasto ancora in esecuzione, con il socket chiuso mentre è ancora in atto il trasferimento di un altro blocco di dati.
Un altro problema che si viene a creare è l'invio di dati diversi nello stesso buffer:un host potrebbe ricevere i dati che doveva ricevere un altro host, dato che la risorsa usata per l'invio delle informazioni è condivisa.

5.3.2 Sicurezza dei pacchetti scambiati tramite TCP

I messaggi scambiati tra client e server vengono scambiati dalla classe del buffer, che converte il testo in esadecimale prima di inviarlo all'altro host connesso al nodo opposto. I messaggi scambiati hanno un meccanismo di integrità grazie al MAC inserito all'interno, in modo tale che il server possa verificare se il messaggio sia stato modificato computando un MAC e confrontandolo con quello del client. L'integrità

[5]La race condition è il fenomeno che in cui una applicazione con più thread concorrenti ha un risultato finale dipendente dall'ordine temporale in cui sono eseguiti e non è prevedibile a priori.

è garantita da questa misura, ma non sono state ideate tecniche per rispettare la proprietà di confidenzialità dei dati trasmessi. Il framework non contiene nessun tipo di operazione che fornisce proprietà di confidenzialità all'interno delle funzioni che si occupano della attestazione. remota per

5.3.3 Limitazioni della divisione del codice

Gestione indirizzi di memoria nelle funzioni entrocontenute

Il framework applica il trasferimento del codice di una funzione su un server solo se questa rispetta le condizioni di entrocontenibilità: questi vincoli permettono di risolvere molti problemi di riferimento a variabili riferite nel blocco di codice selezionato, ma definite e modificate in sezioni del codice in cui non è stata applicata la tecnica di divisione. All'interno del codice della funzione, usare variabili che sono gestite con gli operatori dei puntatori può portare a comportamenti non previsti e a problemi di esecuzione: ad esempio, una scanf indica come argomento l'indirizzo di una variabile in cui deve essere salvato il valore letto da input. In quel caso, il valore della variabile sarà salvato in un indirizzo della memoria appartenente al server remoto, e non può essere restituito quell'indirizzo al client che è in attesa del risultato della funzione eseguita. Il framework non effettua nessun controllo sulla logica dei puntatori all'interno del codice implementativo. Un altro esempio avviene con i puntatori

Supporto ai tipi dei parametri di una funzione entrocontenibile

Al momento, come già elencato nella Sezione 5.1.2, le funzioni entrocontenibili di una applicazione possono avere soltanto parametri dal tipo elementare, non sono quindi supportate le strutture dati e le unioni.

Capitolo 6

Risultati sperimentali

In questo Capitolo, sono analizzati i risultati ottenuti testando il framework su pacchetti di applicazioni. All'interno della Sezione 6.1 sono presentate le due applicazioni usate per le prove che verranno eseguite per i test successivi. Successivamente, nella Sezione 6.2, si inizia l'analisi delle prestazioni delle applicazioni eseguite: sono descritti in Sezione 6.2.1 quali sono i fattori che portano ad avere un rallentamento delle prestazioni dei test effettuati, e nella Sezione 6.2.2 le misurazioni delle applicazioni con funzionalità di rete vengono analizzate dal punto di vista del traffico generato e della durata del processo.
Nella Sezione 6.3.1 vengono descritti i tempi impiegati dal framework per convertire le applicazioni, e come la compilazione sia rallentata da determinate librerie che sono aggiunte per fornire le prestazioni richieste. Infine, in Sezione 6.3.2 sono presenti i dati dovuti ai test per osservare il comportamento del framework SGX su una applicazione in base al numero di enclavi che è richiesto di generare.

6.1 Analisi delle applicazioni di test e dell'ambiente di esecuzione

Le applicazioni da testare dovevano permettere di poter applicare l'attestazione remota, la divisione del codice e di poter richiedere più enclavi senza modificare la struttura delle istruzioni usate. Per poter testare tutte le funzionalità separatamente e applicate contemporaneamente, sono state create le applicazioni presentate in questa sottosezione:

ecall_cost è una applicazione che riceve come argomento un numero da diminuire fino allo zero e incrementare fino al valore originale. Sono definite due funzioni distinte, una per incrementare il numero e l'altra per decrementarlo. In base al numero N di enclavi che si vuole testare, si divide il parametro ricevuto da linea di comando in N parti, e si decrementa e incrementa chiamando le due funzioni.

heavy_computations è una applicazione in cui si riceve un numero N, inizialmente molto grande, e calcola la somma dei prodotti tra il numero della

i-esima iterazione e i tre termini successivi secondo la formula $somma = \Sigma_{i=1}^{N}(i+1)(i+2)(i+3)(i+4)$.
Su questa funzione, si applica la divisione del codice in modo tale da spostare la computazione sul dispositivo remoto. Usa la funzione di calcolo del prodotto per cinque volte, cambiando il parametro passato: prima compie i calcoli ricevendo come parametro N pari a 100000000, poi calcola per due volte il prodotto con N pari 1000000, e infine usa i valori N=10000 e N=100.

nome applicazione	*SLOC*	*Decision point*	*Cicli*	*Assegnazioni*	*Chiamate a funzioni*	*Complessità ciclomatica*
ecall_cost	270	32	30	109	35	63
heavy_computations	96	15	5	40	9	21

Tabella 6.1: Metriche delle applicazioni scelte generate con Frama-C

La Tabella 6.1 rappresenta le metriche delle applicazioni usate nella fase di test del framework. Le metriche sono state raccolte con Frama-C, che all'inizio del processo di conversione raccoglie, stampandole successivamente in output, e filtrate per prendere solo quelle più rilevanti all'analisi. L'unità di misura *Source line of code*(SLOC) rappresenta la dimensione del sorgente in linee di codice, mentre la complessità ciclomatica è la metrica software che indica il numero di percorsi possibili di un grafo di controllo del flusso di un programma che sono tra di loro linearmente indipendenti.
La macchina su cui sono effettuate le misurazioni come benchmark del framework è compatibile con SGX, e le sue specifiche sono descritte nella Tabella 6.2. Il processore della macchina soffre di alcune vulnerabilità di sicurezza note ad Intel: nel report inviato da Intel come risposta alla Quote inviata, è segnalato che il dispositivo appartiene ad una famiglia di processori vulnerabili con il segnale *GROUP OF DATE*. Questo avviso non segnala che la Quote non è autentica, e nemmeno che il client non sia attendibile, ma solo che potrebbe trattarsi di un dispositivo non aggiornato: per motivi di testing quindi, i servizi di attestazione inseriti dal framework accettano report di Intel con questo avviso di sicurezza, ma è possibile modificare questa politica (vedesi Manuale Sviluppatore).

Specifiche	
Modello	Asus VivoBook15A512JA
CPU	Intel Core i5-1035G1 @ 1 GHz 3.6 GHz
RAM	8 GB DDR4
Sistema Operativo	Ubuntu 20.04

Tabella 6.2: Specifiche del computer su cui sono state testate le applicazioni

6.2 Test delle prestazioni in esecuzione

Il framework viene testato con la conversione di due applicazioni scritte con lo scopo di inserire in ciascuna di esse una funzionalità di attestazione. Le operazioni del framework sull'applicazione possono essere suddivise in tre fasi:

1. *conversione* del sorgente dell'applicazione usando il framework SGX, in questa fase il framework modifica il sorgente originale seguendo le notazioni inserite;

2. *compilazione*, in cui sono creati prima i file generati nella precedente fase e dopo il file eseguibile dell'applicazione, usando lo script makefile che il framework genera al termine della conversione;

3. *esecuzione* dell'applicazione con le funzionalità introdotte.

6.2.1 Degradazione delle prestazioni

L'uso di Intel SGX peggiora le prestazioni di una applicazione, che deve eseguire cambi di contesto di esecuzione per poter interagire con i dati che sono contenuti nella memoria protetta dell'enclave e con i dati appartenenti alla memoria non protetta. I tempi di esecuzione quindi dipendono da due fattori: numero di ingressi (ECall) e uscite (OCall) nelle enclavi dell'applicazione, e dipendenze della rete.
I cambi di contesti di esecuzione, come spiegato nel Capitolo 2, chiamano un interrupt che scatta una fotografia dello stato interno del processore nel momento dell'applicazione in cui è stato richiesto il cambio, e i valori contenuti nella memoria e nei registri devono essere salvati momentaneamente. Al termine dell'esecuzione nel contesto invocato, bisogna ritornare al precedente stato del processore, ripristinando i dati salvati precedentemente. Questa routine del sistema non può essere evitata o ridotta tramite ottimizzazioni del codice, perciò introducono un overhead inevitabile nell'applicazione.
Il processo di attestazione, e lo scambio di dati tra l'applicazione client e il server generato, sono eseguiti sulla rete locale del computer: il client si collega all'indirizzo del localhost(conosciuto come indirizzo di interfaccia di loopback), mentre l'applicazione server ascolta su una porta la richiesta di connessione in arrivo; ricevuta la richiesta, essa crea un socket che collega i due processi, simulando la connessione remota. Il processo di attestazione remota dipende sia dalla connessione tra client e server, e della connessione tra server e Intel Attestation Service: in questa connessione, bisogna verificare la validità del certificato X509 che il server deve richiedere ad Intel in fase di registrazione al suo servizio di attestazione, e deve poi contattare l'IAS inviando più richieste. Più è richiesta l'attestazione di un programma, più volte dovranno essere ripetute le richieste, creando così più situazioni in cui l'applicazione rimane in attesa di risposte che non sono immediatamente elaborate.L'ambiente di esecuzione in cui i test sono stati svolti è da considerare ottimale: i ritardi dovuti alla comunicazione tra dispositivi collegati a due reti remote sono trascurabili.

Analisi delle prestazioni via rete

In questa Sezione si analizzano dati appartenenti al traffico delle applicazioni che comunicano con un sistema remoto: questo può essere il server da contattare per eseguire le funzioni trasferite con la divisione del codice, il server di attestazione o il server delle funzioni trasferite che riceve la richiesta di inizio di attestazione del client.

Il singolo processo di attestazione ha un impatto leggero sulla scheda di rete del dispositivo, con il processo del server ad avere molto più peso rispetto a quello dell'applicazione convertita. La lettura dei dati raccolti è stata effettuata tramite gli strumenti offerti dall'applicazione Wireshark[1], uno strumento per analisi dei protocolli di comunicazione tra reti, che permette di utilizzare visualizzazioni grafiche dei pacchetti scambiati durante la fase di ascolto in cui registra tutti i dati. I dati osservati e presentati in questa Sezione sono ottenuti dopo aver raccolto il traffico della scheda di rete, e averlo filtrato in maniera opportuna per non visualizzare pacchetti non relativi ai flussi di traffico del processo delle applicazioni, di 100 esecuzioni dell'applicazione heavy_computations. I valori raccolti dallo sniffing del traffico sono stati salvati in un file csv e usati per creare i grafici usati nella Sezione.

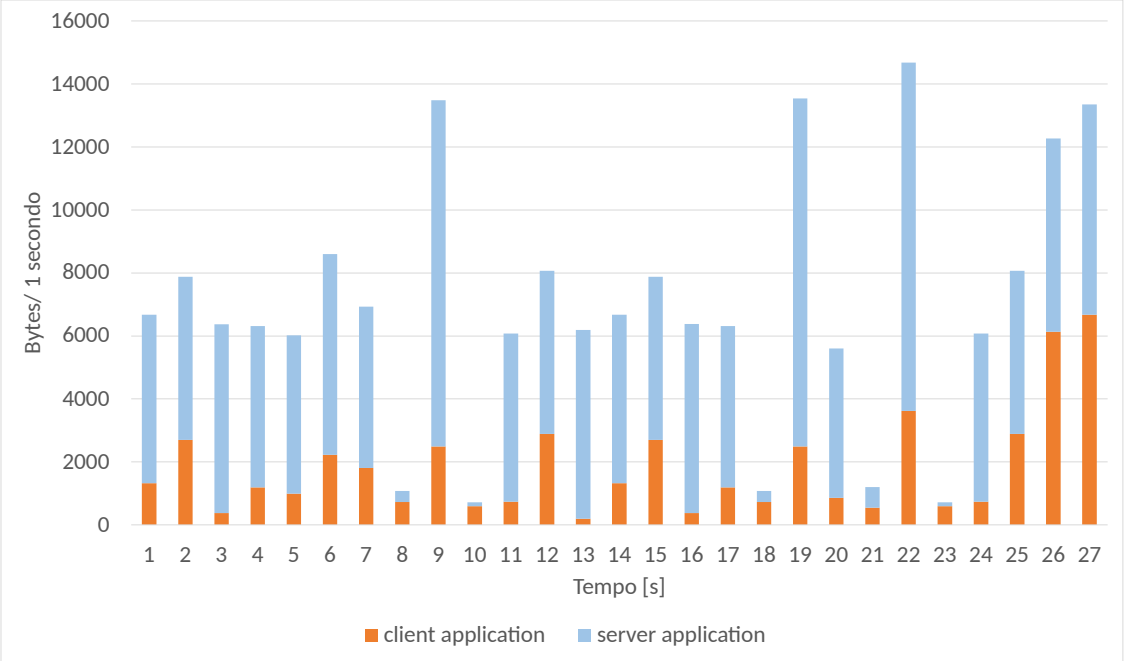


Figura 6.1: Grafico sul traffico trasmesso all'interno della rete locale durante più processi di attestazione.

indirizzo	*pacchetti*	*bytes*	*pacchetti tx*	*bytes tx*	*pacchetti rx*	*bytes rx*
40.87.90.88	31	20k	12	16k	19	4406
192.168.1.65	36	21k	22	4680	14	17k

Tabella 6.3: Traffico trasmesso all'interno della rete locale durante il processo di attestazione.

Prendendo come riferimento la Figura 6.1, in cui sono stati applicati filtri per visualizzare i pacchetti inviati dal processo del client al server in Bytes al secondo (in arancione) e in blu quelli generati dal server, sono illustrati soltanto i primi 27 secondi del processo per semplicità di visualizzazione dei grafici, sapendo che i dati scambiati nelle iterazioni successive hanno un comportamento quantitativo e

[1] https://www.wireshark.org/

qualitativo simile a quelli presentati. Possiamo notare come sia il server a trasmettere più dati tra i due processi: questo è giustificato dalla sessione TLS aperta, che prevede una fase iniziale di handshake tra i due peers in cui avviene uno scambio di certificato per identificarsi come server provider registrato ufficialmente con il servizio di Intel. La rappresentazione grafica scelta è quella di un grafico a barre impilate, in modo tale che si possa vedere il confronto qualitativo tra i diversi flussi di traffici, mentre la linea nera indica i byte totali scambiati. La Tabella 6.3 mostra i dati relativi al traffico totale scambiato all'interno del processo, come valori medi calcolati su 100 esecuzioni: in termini di dimensioni di byte la differenza tra i dati elaborati dai due processi (per riferimento, l'indirizzo IP 40.87.90.88 è quella dedicata al processo del server), per un confronto quantitativo.
Nella Figura 6.2, sono mostrati i primi 26 secondi del traffico scambiato quando l'applicazione heavy_computations include l'attestazione remota, richiesta al lancio dell'applicazione, e la divisione del codice, quando deve chiamare la funzione meno onerosa dal punto di vista del traffico scambiato, durante 100 esecuzioni consecutive: notiamo che tra il terzo e il quinto secondo di esecuzione registrato, i valori registrati sono simili a quelli registrati negli stessi istanti temporali della Figura 6.1, e questi valori sono relativi al processo di attestazione compiuto da entrambe le applicazioni. Una volta terminata la verifica con successo, avviene lo scambio di dati tra client e server, e il server trasmette molti più dati dovuti alla funzione di somma dei prodotti che è stata trasferita remotamente. Nei secondi successivi, il comportamento si ripete, considerando che alcuni valori variano in base al processo di attestazione che non ha mai la stessa durata, e che i valori più bassi registrati (al tredicesimo e all'ultimo secondo rappresentato) sono dovuti all'invio, da parte del client, del parametro della funzione sul server richiesta.

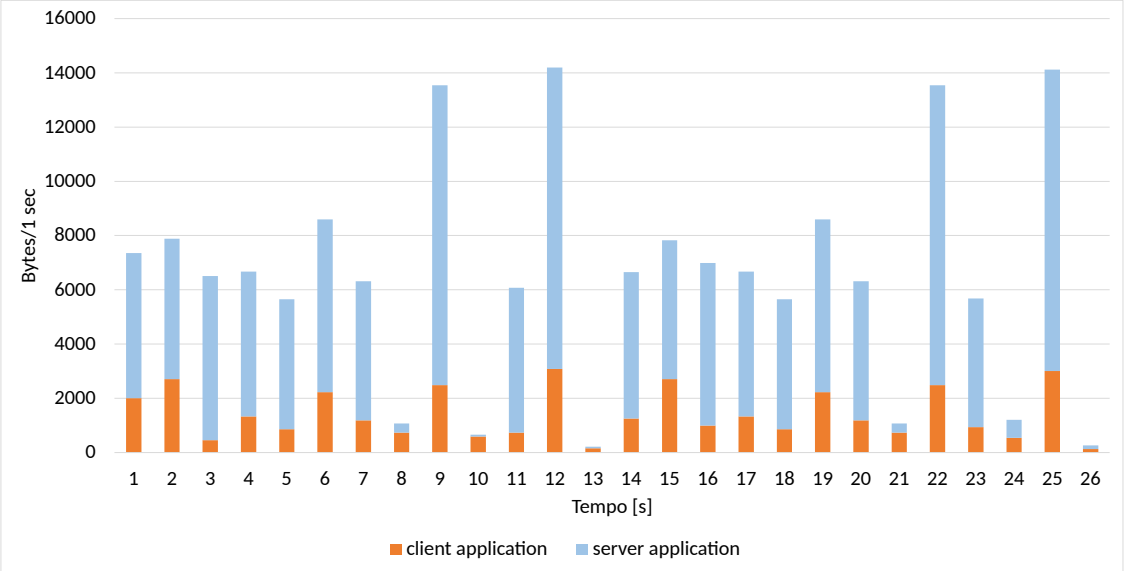


Figura 6.2: Grafico del traffico scambiato in una applicazione con attestazione e divisione codice

6.2.2 Analisi dell'impatto della attestazione remota

In questa Sezione, si implementa sui programmi di test l'applicazione remota, confrontando i valori misurati dopo la conversione del framework con quelli dell'applicazione senza richieste di funzionalità. Le misurazioni riportate sono valori ottenuti con la media di 50 esecuzioni per processo, e il programma ecall_cost è stato scritto generando una sola enclave. È analizzato solo il tempo di esecuzione dell'applicazione convertita dal framework; le fasi di conversione e compilazione del file binario sono invece state trattate nella Sezione 6.3.1.
Nella Figura 6.3 viene mostrato per ogni applicazione il tempo di esecuzione impiegato, quando all'interno dell'applicazione è richiesta l'attestazione remota solo una volta: a sinistra abbiamo il valore per l'applicazione convertita con il framework SGX, con il processo di attestazione inseriti all'inizio della funzione main dell'applicazione, e a destra abbiamo il valore della stessa applicazione senza l'aggiunta della funzionalità. Da questi confronti, si può vedere che il processo di attestazione impiega in media 2,3s: questo valore oscillatorio è ottenuto in un contesto di esecuzione in cui le degradazioni della rete sono minime, e non sono stati riprodotti problemi transitori imprevedibili all'interno del processo. Il processo di attestazione rallenta pesantemente la prima applicazione di test, che è eseguita in pochi millisecondi, mentre per quanto riguarda una applicazione più onerosa computazionalmente, la differenza è meno netta in termini di aumento del tempo di esecuzione in percentuale: le operazioni compiute dalle funzioni di attestazione (tra queste ci sono la creazione dei messaggi scambiati, i cambi di contesto di esecuzione per avere informazioni private dell'enclave, operazioni crittografiche asimmetriche e di hashing) hanno infatti una complessità maggiore, e impiegano un tempo di esecuzione dello stesso ordine di grandezza di una applicazione normale, mentre le applicazioni più semplici vengono eseguite in tempi molto più piccoli.

Si analizza ora l'impatto dovuto all'implementazione della attestazione remota in combinazione con la tecnica di divisione del codice. Anche per questo caso, sono testate le stesse applicazioni nelle stesse precedenti condizioni, ma la funzione trasferita è per ecall_cost una funzione di decremento del numero ricevuto in input, e nel caso di heavy_computations sono trasferite sul server remoto due delle funzioni di calcolo del prodotto, invocate quando è usato come parametro il valore N pari a 1000000. L'applicazione ecall_cost ha tempi di prestazione simili a quando è applicata soltanto l'attestazione remota, ma nell'altra applicazione vediamo che il tempo di esecuzione è di quasi 10 volte aumentato rispetto al precedente, salendo da 0,876 secondi ad un valore finale di 7,733 secondi. Possiamo giustificare questo aumento dalla complessità computazionale delle due funzioni trasferite sul server remoto: al crescere della dimensione del codice spostato, aumenta anche l'overhead dovuto sia all'attesa dell'esecuzione (risolvibile implementando meccanismi di programmazione asincrona) sia alla trasmissione della quantità di dati che devono essere ricevuti.

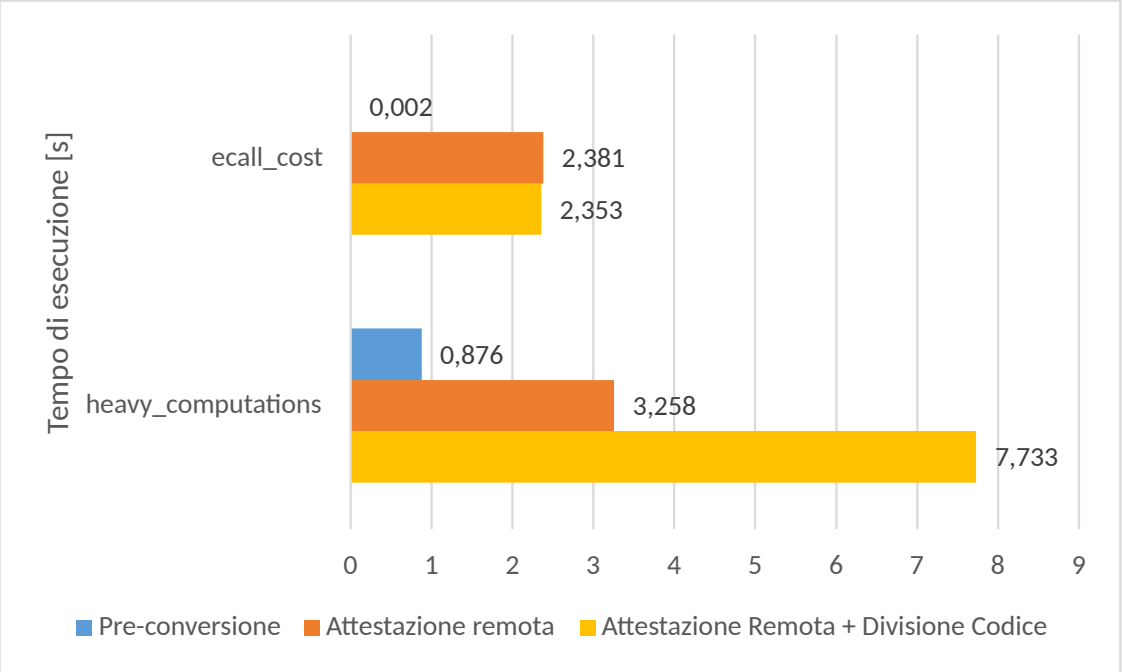


Figura 6.3: Confronto della durata di esecuzione delle due applicazioni con attestazione remota e senza

6.3 Analisi delle performance durante la generazione del binario protetto

In questa Sezione, si misura l'overhead nei due programmi di test introdotto dalle operazioni necessarie per la generazione del file eseguibile finale. La Sezione 6.3.1 mostra i risultati riportati prima dell'esecuzione del file binario generato, riferendosi quindi alla fase di conversione del sorgente e alla fase di compilazione. Nella Sezione 6.3.2 si analizza nello specifico l'impatto della richiesta di più enclavi da parte dello sviluppatore sulle prestazioni dell'applicazione.

6.3.1 Analisi delle prestazioni del processo di generazione del binario protetto

La scelta dello sviluppatore di inserire funzionalità di protezione SGX nel codice impatterà sulle prestazioni a seconda del punto in cui sono richieste. Le applicazioni sono testate nel caso in cui si applica solo l'attestazione remota e nel caso in cui è applicata l'attestazione remota insieme alla divisione del codice. Nei test effettuati, i valori sono calcolati come valori medi di 50 esecuzioni per il processo di conversione del sorgente e 50 esecuzioni per il processo di compilazione dei file, e il numero di enclavi richiesto per applicazione è pari a N=15 per l'applicazione ecall_cost e pari a N=1 per heavy_computations(in caso di semplice attestazione remota, altrimenti sarà pari a N=2): la scelta del numero di enclavi influirà molto sulla durata dei due processi, come si vedrà nella Figura 6.4.

La Figura 6.4 raccoglie i dati della durata del processo di conversione di ciascuna delle due applicazione. In questo caso, è ecall_cost ad essere l'applicazione più

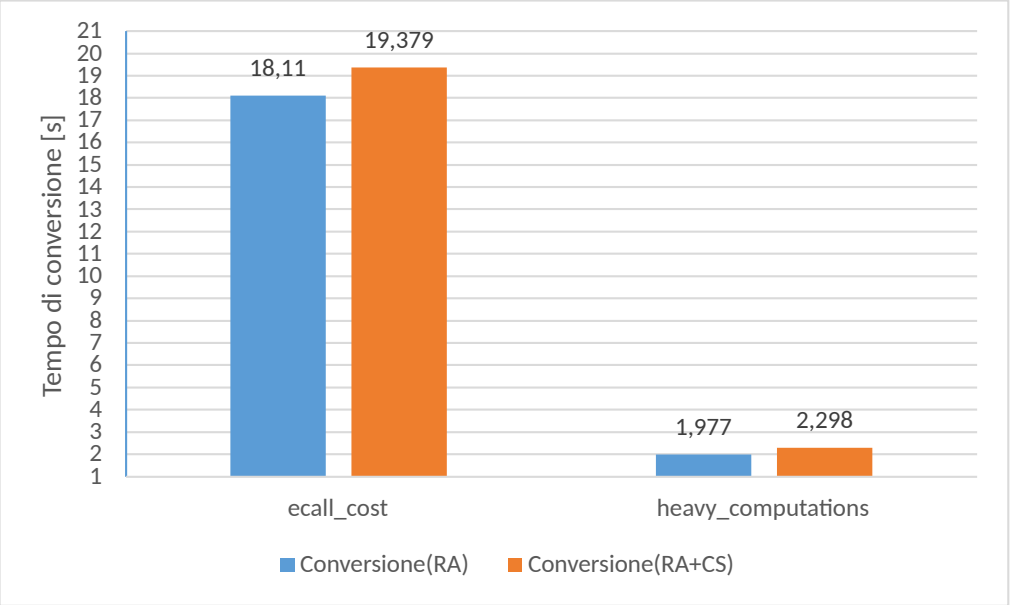


Figura 6.4: Confronto sulla media del tempo di conversione delle due applicazioni con l'applicazione delle due funzionalità

onerosa: questo è dovuto al fatto che per ogni enclave richiesta, bisognerà seguire il processo di analisi del codice per riconoscere le notazioni, i dati e le funzioni da trasferire, e le librerie da importare scrivendo il file edl e gli altri file necessari per generare la parte dell'applicazione eseguibile nella memoria protetta di SGX. A seconda del numero di richieste, il tempo di esecuzione totale aumenta sensibilmente: l'applicazione dell'attestazione remota non è consigliabile per applicazioni normalmente eseguite in tempi brevi quindi, se non accettando un rallentamento delle prestazioni evidente. É possibile in alcuni casi applicare le stesse funzionalità del codice e avere risultati migliori, scegliendo ad esempio di richiedere l'attestazione in una sezione del codice diversa: inserire la funzionalità all'inizio di una funzione sensibile chiamata una sola volta è sicuramente più conveniente di inserirla all'interno di un ciclo annidato, o di una funzione richiesta più volte.
Nella Figura 6.4 il grafico rappresenta i dati ottenuti durante il processo di conversione per la generazione dei binari che devono includere attestazione remota con divisione del codice. Anche in questo caso, il tempo impiegato dall'applicazione con un numero alto di enclavi richieste è molto grande, e sommando il precedente valore della conversione, abbiamo già quasi 26 secondi di attesa per una applicazione. I tempi originali sono infatti confrontati nella Figura 6.5, in cui è rappresentato un confronto per il caso in cui è applicata l'attestazione remota con la divisione del codice e senza.

Vediamo come questi processi richiedano molto più tempo, e l'incremento è netto nel caso della compilazione delle applicazioni che chiedono attestazione remota e divisione del codice. Le librerie richieste per queste funzionalità sono molto pesanti da compilare con il file binario finale, ed è comprensibile l'aumento dei tempi impiegati dallo sviluppatore. In particolare, la libreria che impiega più tempo tra quelle inserite è l'interfaccia del server (si stimano 4,4 secondi per la compilazione

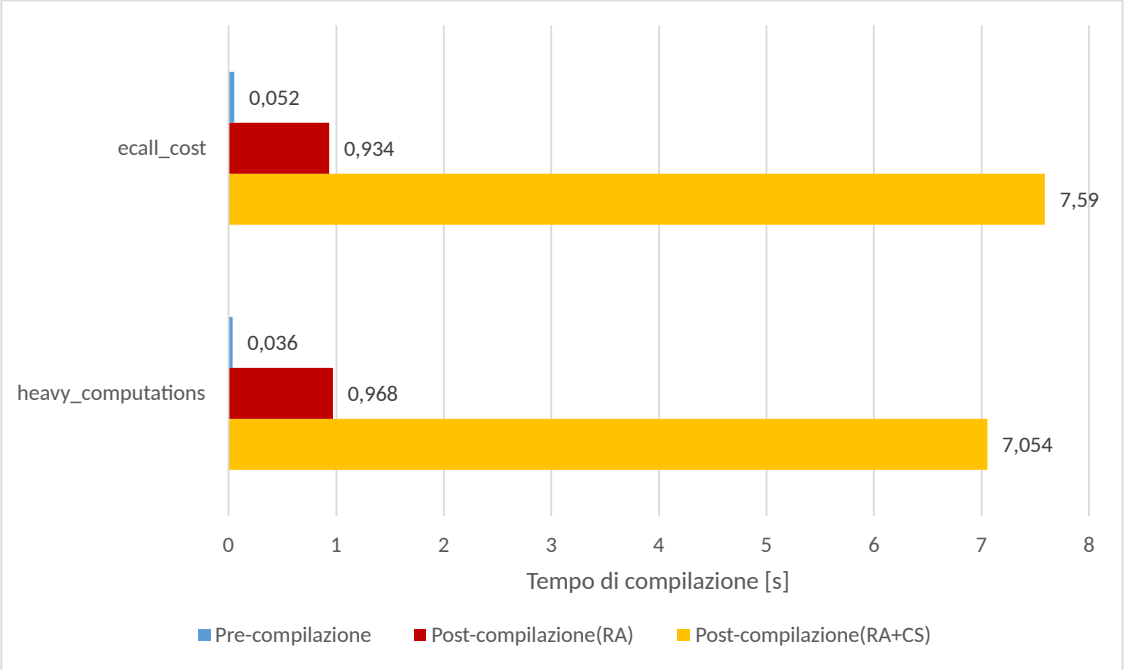


Figura 6.5: Confronto sulla media del tempo di compilazione delle due applicazioni per applicazione di attestazione remota con divisione del codice e senza.

delle librerie), generata poiché il server creato per la divisione del codice deve anche occupare il ruolo di verificatore di integrità del client: al suo interno sono incluse diverse librerie usate dalla verifica dell'enclave alla comunicazione di due peer via socket, e tra queste è presente quella dedicata alle operazioni crittografiche. Per questa libreria, possiamo infatti notare la presenza di funzioni dedicate al caricamento in memoria dei certificati X509, le firme con algoritmi di cifratura simmetrici, e della loro verifica; ma sono anche referenziate e definite al suo interno le funzioni per i digest SHA256 [39], inclusi nella Quote da inviare all'IAS. Il file oggetto di questo modulo pesa infatti 3 MB, e la compilazione statica implementata rallenta la durata del processo che genera l'eseguibile delle applicazioni server.
Questo processo è potenzialmente ottimizzabile: ad esempio, è possibile scegliere di usare librerie dinamiche da referenziare all'interno del file eseguibile, accorciando i tempi impiegati dal processo di compilazione, ma rallentando invece l'esecuzione dell'applicazione che dovrà caricare al lancio tutti i file necessari. Tuttavia, anche se per lo sviluppatore la scelta di rallentare la fase di esecuzione dell'applicazione è tollerabile, il numero di librerie da importare non è irrilevante, e bisognerebbe fare una selezione di quali sono le librerie più convenienti da includere e quali no. Un'altra possibile ottimizzazione può essere l'ottimizzazione del codice delle librerie di attestazione del server.

6.3.2 Analisi sull'uso di enclavi multiple

Per lo studio dell'impatto di più enclavi all'interno di un software, si analizza l'applicazione ecall_cost, sviluppata per poter facilmente trasformare funzioni in ECall

da chiamare senza modificare il codice. Come già presentato nella Sezione 6.1, l'applicazione riceve un numero come input da tastiera, e usando un ciclo *while* lo decrementa di una unità, entrando ogni volta dentro una enclave quando l'applicazione è convertita dal framework. Il valore viene diviso per un numero di variabili pari alla quantità di enclavi richieste come divisore: in questo modo, per N enclavi scelte in esame abbiamo N funzioni "*dec*" che decrementano il valore dell'argomento ricevuto in input della stessa quantità, effettuando lo stesso numero di chiamate di funzioni ma all'interno del contesto protetto appartenente ad ogni enclave generata. Successivamente, raggiunto il valore 0, si ripete lo stesso processo ma cambiando operazione aritmetica e aumentando ad ogni chiamata della funzione "*add*" il numero ricevuto, fino a quando sarà riottenuto il valore originale ricevuto in input. Queste semplici N funzioni che ritornano il valore incrementato o diminuito saranno dunque trasformate dal framework in N funzioni ECall nella successiva fase di conversione del codice.
Per nessuno di questi test sono state richieste le funzionalità di attestazione remota e divisione del codice, mentre i valori sono stati calcolati su una media di 50 esecuzioni per test e di 100 esecuzioni per quanto riguarda la fase di compilazione dei sorgenti.

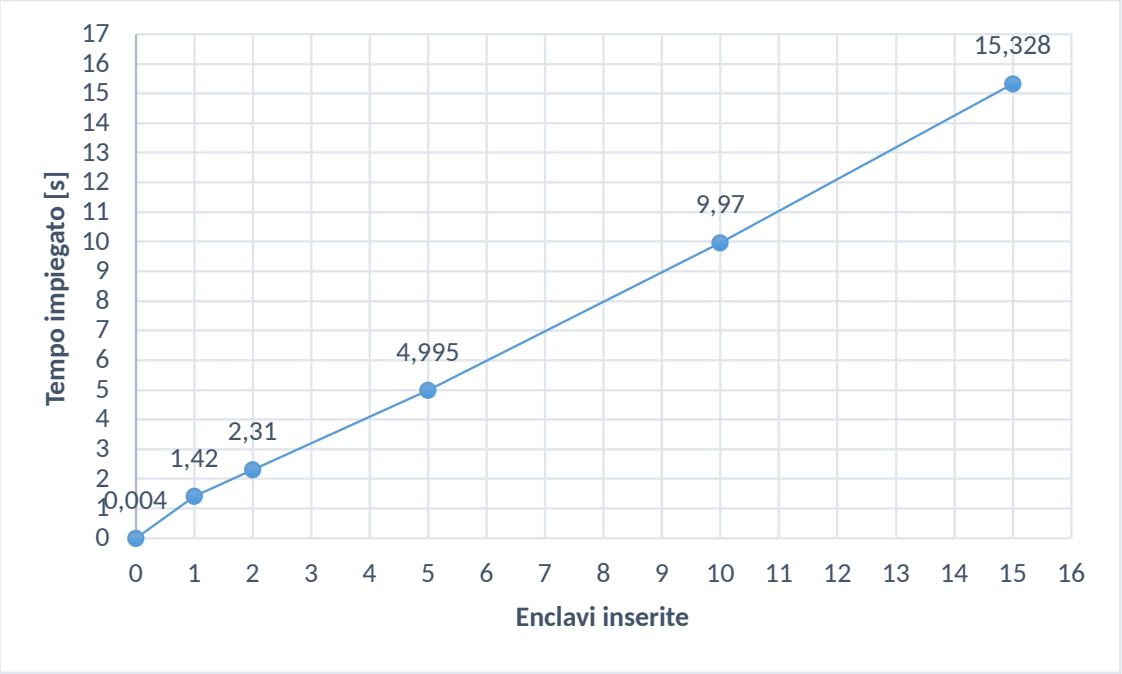


Figura 6.6: Benchmark sulla media del tempo di conversione dell'applicazione all'aumentare di enclavi

Nella Figura 6.6 osserviamo il confronto con il processo di conversione del sorgente dell'applicazione e il sorgente compatibile con le funzionalità SGX, aumentando il numero di enclavi che si vuole testare e introducendo N funzioni "add" e N funzioni "dec" per ciascuna di esse. Nonostante l'applicazione esegua le stesse istruzioni e operi con gli stessi dati, il costo di introduzione delle enclavi procede ad aumentare linearmente la durata del processo. La linea arancione rappresenta infatti il tempo dedicato alla compilazione del sorgente originale: si possono vedere diversi ordini di grandezza di differenza tra le due serie di tempi misurati, e l'applicazione

originale ha un ciclo di vita molto più breve della singola fase di conversione. Da questo esempio, abbiamo almeno un caso che dimostra come l'aumento del numero delle enclavi introduca rallentamenti a partire dalla prima fase di modifica del framework. Il costo da pagare in termini di prestazioni non influisce solamente sul processo di conversione dell'applicazione, come si vede nella Figura 6.7. La compilazione aumenta con il numero di enclavi in maniera lineare, sebbene in questo caso l'ordine di grandezza del tempo impiegato rimane uguale al precedente, e il tempo dovuto a questa fase non aumenta in maniera sensibile come osservato nei precedenti casi.

La Figura 6.8 invece riporta i dati dell'esecuzione dell'applicazione, facendo sem-

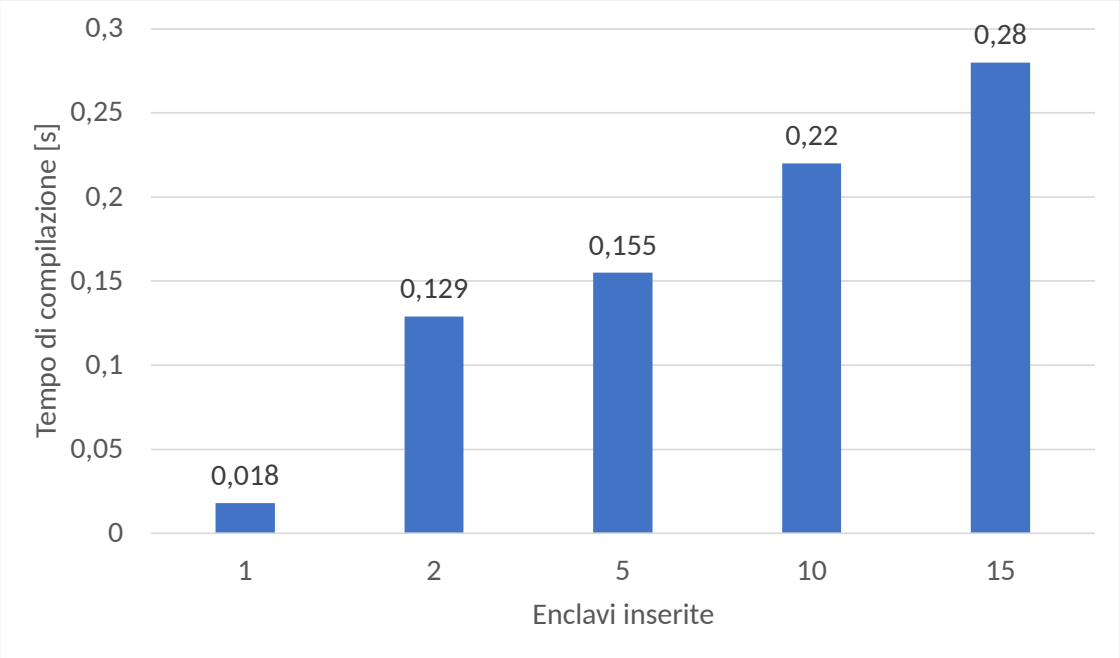


Figura 6.7: Benchmark sulla media del tempo di compilazione dell'applicazione all'aumentare di enclavi

pre un confronto tra la versione convertita con il framework e quella non. Nell'applicazione originale, possiamo vedere l'effetto dell'ottimizzazione del processore sul codice originale, che diminuisce il tempo di esecuzione impiegato nonostante il numero delle operazioni sia lo stesso. Nel caso del contesto delle enclavi invece, l'esecuzione è aumentata in maniera sensibile, arrivando nel caso peggiore da 0,0008s a 0,213s: la degradazione delle prestazioni dovuta al cambio di contesto di esecuzione è visibile chiaramente, e si può osservare la correlazione diretta tra l'aumento della differenza dei tempi e l'aumento delle enclavi generate. Nel caso più oneroso possibile, la differenza è di oltre tre ordini di grandezza, dimostrando l'enorme peso a livello computazionale dovuto alle librerie di Intel per la tecnologia SGX.

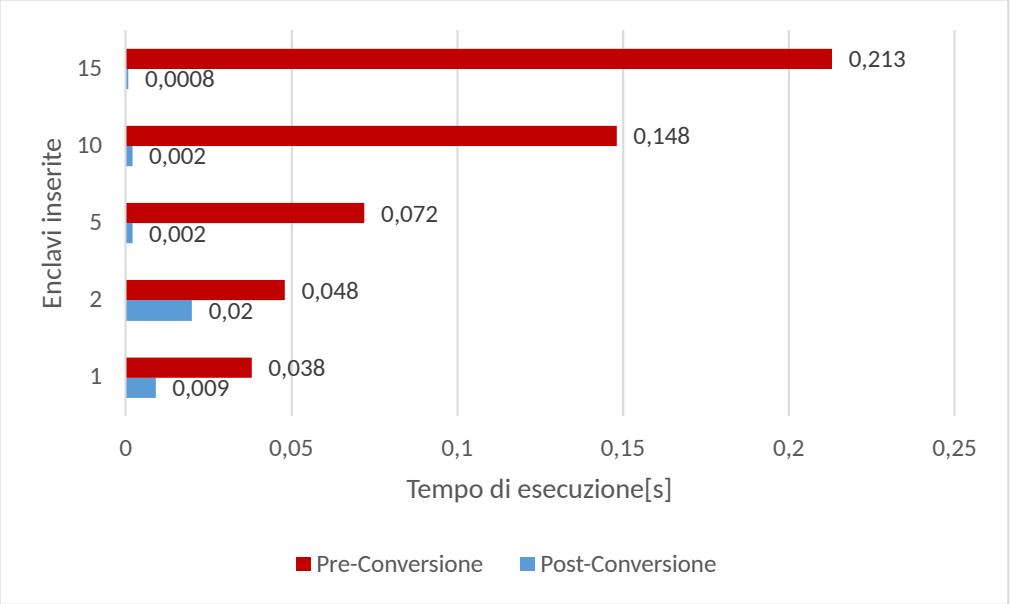


Figura 6.8: Benchmark sulla media del tempo di esecuzione dell'applicazione all'aumentare di enclavi

Capitolo 7

Lavori collegati

In questo Capitolo, l'obiettivo è mostrare nell'ambito dello stato dell'arte altri framework che sono stati pensati per poter offrire ad uno sviluppatore uno strumento che aiuti nello sviluppo di TEE e di applicazioni che usano l'Attestazione Remota. Oltre a questa tecnologia, si noterà come alcuni strumenti che eseguono l'analisi statica del codice dell'applicazione da convertire condivideranno anche l'uso di notazioni statiche, in maniera simile al Framework SGX sviluppato. Gli strumenti presi in esame saranno Glamdring, Enarx, Occlum, Inclavare Containers e Advanced Software Protection.

7.1 Altri strumenti per l'applicazione automatica di TEE

7.1.1 Glamdring

Glamdring[61] è un framework sviluppato dall'Imperial College di Londra, dall'Università Tecnica di Dresda, dall'Università Tecnica di Braunschweig e dall'Università di Otago che lavora su applicazioni scritte in C e compatibili con la tecnologia Intel SGX. L'obiettivo di Glamdring è quello di effettuare una divisione automatica dell'applicazione in partizioni usando le enclavi, aggiungendo ai bordi del contesto protetto controlli di sicurezza durante l'esecuzione dell'applicazione, e permettendo operazioni crittografiche come l'algoritmo di cifratura AES-GCM[61] per il rafforzamento della sicurezza. Una delle tecniche usate da Glamdring è quella del *backward slicing* sul codice che contiene annotazioni: l'analisi statica del codice permette di riconoscere fonti di potenziali vulnerabilità e le sezioni di codice che sono sensibili ad attacchi. Applicando questa tecnica, raggruppa questa parte dell'applicazione nella più piccola partizione possibile e la separa dal resto del codice usando SGX per definire come l'enclave deve difendere le informazioni raccolte.
Il flusso di conversione di Glamdring è diviso in quattro fasi:

1. inserimento delle notazioni nel codice;
2. analisi statica del codice;

3. partizionamento del codice;

4. generazione del nuovo codice.

Le notazioni usate sono simili a quelle scelte dal framework SGX: si usa un compiler pragma[1] per segnalare due tipi di variabili, le *fonti* (source) e i *ricevitori* di dati (sink). Le fonti sono variabili che fungono da canali nell'applicazione per il trasferimento di dati sensibili da proteggere, come ad esempio un canale socket usato da una chiamata di sistema come la funzione di lettura byte; i ricevitori sono quelle variabili attraverso le quali i dati sensibili escono dall'applicazione, come un buffer di input/output.
Nella fase di analisi del codice, raccoglie in un sottoinsieme tutte le istruzioni che dipendono dalle variabili esaminate dalle notazioni inserite. Viene usato un programma che crea un grafo di dipendenze del codice (si nota che anche l'approccio usato dal framework SGX prevede la generazione di un grafo di dipendenze con Frama-C), analizzando il flusso di dati per individuare le interazioni con i ricevitori di dati segnalati dalle notazioni e usando la tecnica del backward slicing in modo da sapere dove aggiungere le operazioni di cifratura dei dati per fornire integrità al programma.
La terza fase crea le partizioni, in modo tale da inserire all'interno di esse tutte le funzioni legate alle variabili segnalate dalle notazioni, e deve legarle all'interno della memoria protetta dalle enclavi SGX, modificando il codice in modo tale che l'uso delle interfacce delle funzioni porti l'applicazione a spostarsi nel contesto protetto tramite le chiamate di transizione delle ECall. Sono usate anche ottimizzazioni e strumenti come Gcov[2] per migliorare le prestazioni dell'applicazione dopo la conversione al protocollo di SGX, che rallenta drasticamente i tempi di esecuzione (come dimostrato nel Capitolo 6). Nell'ultima fase, Glamdring aggiunge nel nuovo codice generato sezioni di codice dedito al controllo dell'applicazione durante la sua esecuzione. Sono poi effettuate le operazioni di conversione tipiche per garantire un TEE: vengono spostate le funzioni all'interno dei file delle enclavi, il codice delle ECall e delle OCall viene compilato automaticamente dal framework, sono inseriti prototipi delle funzioni di cifratura e decifratura usando una chiave condivisa per AES-GCM, gestendo anche quanta memoria deve essere allocata per le enclavi. L'applicazione di Glamdring quindi inizia con il programmatore che seleziona variabili come punto di partenza dell'analisi da svolgere, e l'analisi del framework permette di percorrere il flusso di esecuzione ottenendo così le funzioni e il gruppo di istruzioni che interagiscono con fonti e ricevitori: il framework SGX invece differisce per un approccio di tipo speculare, in cui il programmatore seleziona con le annotazioni le funzioni che devono essere protette, e protegge i dati interni e le istruzioni contenute attivando l'esecuzione in un contesto protetto dall'ambiente esterno.

[1] Tramite le direttive inserite, si indica al compilatore alcune operazioni speciali che mirano alla modifica del codice prima della sua compilazione.

[2] Strumento usato insieme al compilatore GCC per fare test di analisi di copertura del codice sorgente.

7.1.2 Enarx

Enarx[3] è un framework open source appartenente al Confidential Computing Consortium della Linux Foundation[4] che permette di eseguire applicazioni scritte in Rust, in C, in C++, C#, Go, Java, Python e Haskell. Enarx si occupa di creare istanze di TEE, tra le tecnologie che supporta, senza dover modificare il codice sorgente dell'applicazione per implementare le librerie dedicate al TEE, e le applicazioni girano nell'ambiente di esecuzione creato da Enarx in formato WebAssembly. Un vantaggio di questo framework è di poter convertire l'applicazione in una applicazione indipendente dalla CPU su cui è eseguita, in modo tale da creare applicazioni portatili per ogni sistema senza preoccuparsi delle differenze di architettura e di implementazione delle librerie di sviluppo software delle tecnologie TEE. Le applicazioni create possono essere eseguite su diversi processori senza che debbano essere ricompilate, ma possono essere riconfigurate velocemente e avviate come nuovi binari applicativi lanciabili dal sistema di esecuzione in tempo reale di Enarx. Enarx si occupa anche dell'installazione dei componenti del framework, necessari per l'esecuzione dell'applicazione e per fornire un meccanismo di attestazione con un sistema di certificati, e delle librerie e delle interfacce che permettono di gestire i contesti protetti e l'esecuzione del sistema su cui i componenti sono eseguiti e comunicano tra di loro. Le tipologie supportate di TEE che devono agire da backend per far girare le applicazioni sono le seguenti:

- TEE fornita all'interno di un processo, implementata con Intel SGX;
- TEE fornita come Macchina Virtuale del kernel, implementata con Secure Encrypted Virtualization di AMD[5];
- una modalità di debug per gli sviluppatori.

Inoltre, è in sviluppo una compatibilità con la soluzione proprietaria di Arm *Confidential Compute Architecture*[6].
L'architettura di Enarx consiste in tre componenti che sono interconnessi e necessari per poter eseguire le applicazioni: *Enarx Keep*, *Enarx Drawbridge* e *Enarx Steward*. Il componente Keep è definito come l'insieme dell'istanza TEE, la cui creazione è richiesta dal binario di Enarx, e del sistema runtime di Enarx: il suo compito è poter fornire con forte confidenza prove crittografiche che l'esecuzione delle istanze avviene su una CPU valida, e compiono la cifratura delle applicazioni e dei dati mediante chiavi di crittografia asimmetrica di tipo effimere, definite secondo lo standard NIST SP 800-57 [62] come chiavi che vengono generate ad ogni esecuzione di un processo di generazione chiavi, risultando così uniche per quella sessione. L'applicazione deve essere caricata all'interno, e tutti i dati in ingresso e in uscita sono cifrati per fornire confidenzialità all'utente. Il modello di Enarx

[3] https://enarx.dev/

[4] https://www.linuxfoundation.org/

[5] https://developer.amd.com/sev/

[6] https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture

infatti prevede che tutto quello che appartiene all'esterno dell'architettura di Enarx sia reputato non affidabile, e solo il processore, che esegue l'applicazione che verrà caricata, viene assunto come agente esterno privo di rischi per la sicurezza.
Il componente Drawbridge funge da archivio di applicazioni scritte in WebAssembly che devono essere caricate all'interno dell'istanza TEE. Infatti, quando l'applicazione deve essere eseguita, viene passato un riferimento ad essa all'interno del Drawbridge, in modo tale che si cerchi nell'archivio quale applicazione deve essere consegnata all'interno dell'ambiente protetto. Il Drawbridge permette di memorizzare le applicazioni di un pubblicatore o di uno sviluppatore (che può coincidere con il pubblicatore) attraverso chiamate REST.
Il terzo componente, dedicato all'attestazione, prende il nome di *Enarx Steward*, ed è necessario per lanciare l'applicazione Enarx: il suo compito è verificare la corretta installazione dell'istanza TEE e preparare il processo di attestazione. In tal caso, lo Steward svolge il ruolo di Certificate Authority che richiede al backend dell'applicazione di generare prove di attestazione crittografiche sotto forma di un certificato firmato con la chiave pubblica generata dalla Keep con cui è in contatto. Dopo averle verificate estraendole, le traduce in un nuovo certificato di attestazione che rispetta lo standard X509[63] e che potrà essere usato nei confronti di un Relying Party esterno che potrà verificare l'applicazione remota aprendo una sessione TLS[64]. Il lancio di una applicazione, che comprende la preparazione dell'ambiente TEE e il processo dell'attestazione remota di esso, viene realizzato nei seguenti passaggi:

- l'ambiente di esecuzione runtime crea sulla macchina dell'host una istanza TEE e carica al suo interno il codice del componente Keep, compiendo ad ogni aggiornamento del codice inserito una misura di attestazione della macchina firmata a basso livello[7], con la chiave privata della CPU a cui solo lei può farvi accesso;

- quando il caricamento del codice è stato completato, l'istanza Keep è lanciata e la misura di basso livello è inviata allo Steward sotto forma di certificato e insieme sono anche comunicate le informazioni (ad esempio, il link di riferimento usato per scaricare l'applicazione dal Drawbridge) dell'applicazione che dovrà essere recuperata dal Drawbridge e eseguita;

- il componente Steward cerca la presenza della misura in un database di valori attendibili e se l'integrità è confermata firma un certificato da lui creato che rispedisce all'istanza Keep;

- il certificato è usato in una comunicazione che l'istanza Keep apre con il componente Drawbridge come prova di affidabilità.

Quando quest'ultimo passaggio va a buon fine, tutte le risorse necessarie all'esecuzione dell'applicazione richiesta saranno trasferite in forma cifrata all'interno dell'istanza TEE, grazie ad una connessione HTTPS. Nel certificato saranno presenti

[7]La misura è aggiornata continuamente, come avviene per l'aggiornamento della misura di un TPM al boot del sistema.

le informazioni necessarie per aprire connessioni affidabili, e in base alla tecnologia di TEE che è stata impiegata diversi parametri di sicurezza. Enarx permette anche una funzionalità di attestazione remota che implementa l'infrastruttura di una cache di certificati Azure[8].

7.1.3 Occlum

Occlum [65][9] è un sistema open source che permette di eseguire applicazioni che vengono eseguite in ambienti esposti ad attacchi e non sicuri all'interno di enclavi di SGX, invocate durante l'esecuzione dell'applicazione non tramite istruzioni inserite nel codice sorgente (che rimane non modificato) ma tramite comandi lanciati dai componenti del framework. Composto da tre componenti (la toolchain personalizzata *occlum-gcc*, il verificatore e la libreria LibOS), la libreria LibOS sostituisce la libreria standard di Intel SGX in modo tale da supportare il multitasking in maniera sicura e efficiente. L'ottimizzazione è ottenuta grazie a tecniche che eseguono moduli non fidati all'interno di sandbox, o domini: tramite una tecnica chiamata Processi Isolati su SFI, o SFI-Isolated Processes (*SIP*). SFI è il nome della tecnica *Software-based Fault Injection*[66] che usa i domini di protezione per isolare i componenti non affidabili, creando regioni di memoria riservate a questi componenti e a istruzioni potenzialmente pericolose dal punto di vista della sicurezza (ossia, possono contenere bachi che se sfruttati permettono l'accesso non autorizzato a informazioni o permessi normalmente negati). Questo cambiamento impatta sulle frequenze di utilizzo e di accesso di questi componenti, e può essere implementato con diverse tecniche e diverse politiche[66]. L'isolamento dei SIP è poi rinforzato con uno schema SFI a Multidominio basato su MPX (MPX-based Multi-Domain SFI,*MMDSFI*), che usa un layout della memoria unico: la regione dedicata ai dati è circondata ai suoi confini da due barriere di memoria che proteggono la regione sensibile da accessi di memoria e da istruzioni di controllo di salto [65].
Il linguaggio di programmazione scelto per queste librerie è Rust. Rust si basa sul concetto di *ownership*: ogni valore in memoria appartiene ad una variabile che la possiede e che deve essere necessariamente dichiarata. Il ciclo di vita della variabile è legato allo scope, e alla sua morte la memoria in cui è stato dichiarato il valore — di solito la memoria heap — viene ripulita, a meno che non sia assegnata ad un'altra variabile. La gestione della memoria dell'applicazione scritta in questo linguaggio diventa così più complessa, ma porta anche ad una grande ottimizzazione delle risorse.

I vantaggi dovuti all'uso di questa soluzione sono:

- pochi bug nella gestione a basso livello della memoria, grazie all'uso delle risorse di Rust;
- facilità di uso, visto che è possibile eseguire una applicazione con brevi comandi di shell;

[8] https://azure.microsoft.com/en-us/products/azure-attestation/#features

[9] https://occlum.io/

- supporto a diversi tipi di file system, come un file system cifrato e che permette di effettuare operazioni di scrittura;

La sequenza di passi da compiere per eseguire con Occlum una applicazione è la seguente:

1. si genera un binario dell'applicazione usando la toolchain di Occlum sul file sorgente;
2. si inizializza Occlum all'interno di una cartella, creando una sua istanza nel file system, e per più applicazioni o istanze di una singola applicazione si useranno istanze diverse;
3. all'interno della cartella in cui Occlum è stato inizializzato, il processo ha creato cartelle interne seguendo la struttura di un file system Linux[10] e si genera con un comando il file corrispondente all'immagine del File System usato da Occlum e poi il file corrispondente all'enclave SGX usata da Occlum;
4. si esegue il binario copiato all'interno della cartella che rappresenta l'istanza Occlum.

Se il sistema su cui viene installato Occlum non possiede un processore SGX, si può eseguire lo stesso il processo usando la modalità simulazione. Può essere anche lanciato su Docker, convertendo una sua immagine di una applicazione in una compatibile con Occlum[11], e anche con cluster appartenenti a Kubernetes[12].

7.1.4 Inclavare Containers

Inclavare Containers[13], abbreviato in *IC*, è un framework compatibile con la specifica Runtime di Open Container Initiative[14] sviluppato da Alibaba Cloud[15] e Ant Group[16] in collaborazione con Intel che esegue container[17] e enclavi su piattaforme compatibili con le tecnologie TEE dai sistemi cloud e dai suoi provider. Le enclavi servono per isolare i dati che si vogliono proteggere dagli agenti che hanno privilegio

[10] Ad esempio, ci sarà una cartella /bin per gli eseguibili, una /root per indicare il punto di mount del sistema, una /tmp per le operazioni che non devono essere salvate permanentemente.

[11] https://www.docker.com/

[12] https://github.com/occlum/occlum/tree/master/example/kubernetes

[13] https://inclavare-containers.io/en/

[14] Una struttura fondata da Docker e altri leader delle tecnologie dei container che hanno definito e condiviso pubblicamente tre protocolli di specifiche da rispettare: Runtime, Immagine e Distribuzione dei container. La specifica Runtime evidenzia le norme da seguire per la configurazione in json di un container che deve essere eseguito, a seconda del sistema operativo scelto.

[15] https://eu.alibabacloud.com/en

[16] https://www.antgroup.com/en

[17] Istanza di un software eseguibile in maniera isolata dal resto del sistema. Al suo interno contiene tutte le informazioni necessarie all'esecuzione (codice, dati, librerie) per essere autonoma.

di accesso e lettura ai dati delle applicazioni che vengono eseguite al suo interno, rimuovendoli dal dominio dei componenti che compongono la TCB del sistema.
L'architettura di IC prevede che l'applicazione protetta, insieme alla sua enclave, si trovi all'interno di un servizio cloud fornito da Kubernetes, in una rete di nodi (chiamata *cluster*): il sistema si basa sull'esistenza di un kubelet[18] che svolge il ruolo di Interfaccia del Container Runtime (*CRI*). La CRI è una istanza di *containerd*, l'ambiente di esecuzione backend che gestisce il ciclo di vita di tutti i container[19] legati a questo nodo. IC è compatibile con diversi ambienti di esecuzione e gestione di enclavi, tra cui Occlum e Graphene, e compie le operazioni di firma delle enclavi necessarie per creare una enclave. Per ogni enclave, fornisce uno stack in cui eseguire le rispettive applicazioni. Per lanciare le enclavi nell'ambiente di runtime si deve usare un protocollo *Enclave Runtime PAL API Specification*[20], che indica il sottogruppo di comandi da usare nell'installazione e per la gestione del ciclo di vita delle enclavi. Un pod di un cluster può gestire al suo interno più applicazioni protette, vengono gestiti da shim-rune, che chiama istanze di rune per ogni pod in modo tale da eseguire al loro interno le enclavi e le applicazioni. Le librerie di runtime usano soluzioni integrate con Intel SGX, e tra le soluzioni suggerite per creare un TEE di SGX è presente Occlum. Le alternative, open source o proprietarie dell'utente, devono implementare in ogni caso il componente runtime dell'enclave e le sue API.
La tecnologia di IC può anche fornire una infrastruttura per l'attestazione, chiamata *Enclave Attestation Architecture* (EAA), e al momento della scrittura della tesi è ancora in sviluppo un Request For Comment[21] che definisca formalmente il design di questa architettura. L'attestazione remota serve alla piattaforma cloud di poter avere una prova che le loro applicazioni siano eseguite con SGX: l'enclave genera un report che viene mandato al servizio di certificazione SGX, che controlla che l'enclave che l'ha contattato sia quella prevista nella propria whitelist, e consente l'esecuzione dell'applicazione in quel sistema e di avviare il trasferimento di dati sensibili.
Il funzionamento è basato sulla generazione di certificati X509 contenenti un campo aggiuntivo che inserisca la quote in una estensione[22] del certificato. Questa architettura è basata sul modello *Remote ATtestation procedureS* (RATS)[67], in cui una entità (*Attestatore*), come un componente software o un componente hardware, vuole essere definita come entità sicura e genera per conto di sé stesso prove di identità (un certificato, una firma generata con un sistema di crittografia asimmetrica), e una seconda entità(*Verificatore*) analizza le prove ricevute e in base alle politiche di attestazione decise durante l'implementazione le approva o no. La

[18]Si tratta di un nodo agente di importanza primaria eseguito su tutti i nodi del cluster creato da Kubernetes, e verifica la salute dei servizi che sono eseguiti su tutti gli altri pod della rete, che inviano le informazioni necessarie per poterla valutare.

[19]Sono oggetti simili alle macchine virtuali che contengono all'interno dei container le applicazioni da eseguire e tutto ciò che è necessario, come il filesystem, la memoria da riservare, e sono configurabili per essere installati su ogni nodo della rete.

[20]https://github.com/inclavare-containers/inclavare-containers/blob/master/rune/libenclave/internal/runtime/pal/spec.md

[21]https://github.com/inclavare-containers/inclavare-containers#attestation

[22]https://www.alibabacloud.com/blog/inclavare-containers-the-future-of-cloud-native-confidential-computing_598992

decisione sul tipo di azione di reazione da implementare viene presa dal Verificatore tramite il consulto con un Relying Party esterno che riceverà le informazioni necessarie per poter scegliere quale azione compiere con l'Attestatore, che dovrà essere ricevere avvisi sui motivi per cui il processo di attestazione non vada a buon fine. EAA implementa una versione di TLS specifica, che prende il nome di *RATS-TLS* e include le librerie di OpenSSL[23] per poter gestire i certificati X509.
Inclavare disabilita la gestione delle eccezioni all'interno delle enclavi, risultando così immune da un tipo di attacco informatico conosciuto anche come SmashEx [68]: questo attacco usa come exploit il meccanismo di gestione eccezioni di SGX, che a causa dell'assenza di chiamate di sistema atomiche dell'SDK lascia bachi di programmazione nelle chiamate di rientro effettuate dopo la gestione delle eccezioni. I danni dovuti a questo attacco possono essere corruzione nei dati privati delle enclavi e il leak dei dati e del materiale crittografico segreto usato nelle operazioni crittografiche.

7.2 Altri strumenti per l'applicazione automatica di RA

Le tecnologie commerciali di TEE che si basano su una soluzione di tipo hardware, presentate nel corso dei Capitoli precedenti, hanno avuto una enorme diffusione sul mercato, e la maggior parte dei strumenti sviluppati, open source e non, scelgono di supportare queste tecnologie per l'implementazione dei loro meccanismi. Dopo varie ricerche di opzioni che fornissero la possibilità di attestazione remota via software, si è deciso di illustrare solo Expert system for Software Protection, per la complessità della soluzione e per i vantaggi dovuti al suo utilizzo.

7.2.1 Expert system for Software Protection

Expert system for Software Protection (*ESP*) [69] è uno strumento di protezione software progettato durante il progetto ASPIRE[24] — fondato dalla Commissione Europea — che ha previsto la collaborazione tra l'Università di Gant, il Politecnico di Torino, Nagravision SA, la Fondazione Bruno Kessler, l'Università di Londra Est, SFNT Germany e Gemalto SA.
Nato originariamente con il nome *ASPIRE Decision Support System* (ADSS), con l'obiettivo di fornire uno strumento che rafforzasse la sicurezza del software eseguito su dispositivi mobile con Android OS come Sistema Operativo, il progetto è stato successivamente ampliato in una direzione diversa per rendere compatibile l'applicazione non solo con le architettura ARM per mobile ma anche per le architetture Intel destinate per i PC: sono stati sviluppati parallelamente strumenti che fornissero tecniche di protezione (data hiding, algorithm hiding, anti-tampering, attestazione remota e renewability) e che funzionassero con tutti i dispositivi di

[23] https://www.openssl.org/

[24] https://aspire-fp7.eu/

Intel. Al termine del progetto ASPIRE, si è deciso di continuare il supporto ad ADSS, implementandolo con diversi strumenti di protezione (incluso Tigress[25] che lavora su un sorgente ricevuto in input per generare un nuovo sorgente più robusto ad alcuni tipi di attacchi). A questo punto, è stato cambiato il nome del framework dal precedente ADSS all'attuale ESP: nato come sforzo congiunto di diversi gruppi di ricerca, il lavoro finale è il risultato della collaborazione tra i ricercatori del Politecnico di Torino Leonardo Regano (autore della Tesi di Dottorato [69]), Daniele Canavese e Aldo Basile. Lo scopo di ESP è quindi provare a fornire un software, scritto in C, protetto da possibili attacchi e simulare il comportamento di un esperto di sicurezza che generi una applicazione protetta il più possibile allo stato dell'arte. Il framework permette l'uso di notazioni da inserire nel codice per chiedere diverse proprietà di sicurezza, sotto forma di direttive pragma. Lo sviluppatore può scegliere quindi quali asset devono essere protetti, indicando la proprietà di sicurezza da fornire ai dati, lasciando a ESP il compito di decidere come implementarla. Il flusso di esecuzione di ESP si divide in diverse fasi:

1. analisi del codice sorgente dell'applicazione tramite un componente di analisi statica del codice, che inizia a costruire l'istanza di un metamodello[26] che sarà usato nelle fasi successive per realizzare una previsione di possibili attacchi che potranno essere subiti;

2. valutazione degli attacchi possibili che l'applicazione può subire, aggiornando il meta-modello;

3. protezione degli asset, che consiste nella selezione di tutte le migliori pratiche da applicare, basandosi sul meta-modello e dal tipo di attaccante previsto;

4. occultamento degli asset, prendendo altri tipi di asset non collegati a quelli sensibili, e su cui sono applicate protezioni per disturbare l'analisi dell'applicazione da parte degli attaccanti, facendogli perdere tempo prezioso nello studio di asset non fondamentali all'applicazione;

5. applicazione della soluzione, che lancia gli strumenti esterni di protezione software tramite un componente centrale che li collega, l'*Aspire Compiler Tool Chain* (ACTC), per generare come output il binario finale.

L'ACTC è uno script scritto in Python 3 che coordina le operazioni di modifica del file sorgente, secondo le istruzioni pragma inserite dallo sviluppatore per le proprietà di sicurezza richieste. Possono essere scelti diversi profili di protezione e sono supportate tecniche come l'offuscamento del codice binario eseguibile, rilevamento di modifica del codice oppure la richiesta di mobilità su un server remoto. La versione più recente del framework è disponibile come repository pubblico[27]. Tra le opzioni di sicurezza offerte da ESP, è presente, tramite strumenti esterni

[25] https://tigress.wtf/index.html

[26] Si tratta di una struttura che astrae i modelli realizzati per un processo informatico, in cui sono analizzati per ogni processo le entità che vi partecipano, le relazioni tra di esse e i suoi vincoli.

[27] https://github.com/daniele-canavese/esp

che vengono coordinati da ESP, l'attestazione remota implementata sotto forma di software, perché non si basa su tecnologie proprietarie commerciali come Intel SGX e il TPM. L'attestazione remota è implementabile in due principali modalità : una attestazione *statica*, usata in combinazione con la tecnica di divisione del codice su un server che sarà incaricato di verificare l'affidabilità delle misure calcolate, e una modalità di attestazione *dinamica*, su cui sono stati ideati due approcci di attestazione, la *Dynamic Remote Attestation* (DynRA) e la *Implicit Remote Attestation*, a sua volta suddivisa in una modalità Lineare (nota come *Simple Implicite Remote Attestation*) e una modalità ibrida (*Hybrid Dynamic IRA*).

Attestazione statica

L'attestazione statica è una tecnica di protezione del software che si basa sulla verifica dell'applicazione che chiede l'attestazione remota usando valori che non variano durante l'esecuzione, come le variabili costanti che sono caricate nella memoria e istruzioni del codice. L'applicazione della tecnica di attestazione remota statica in ESP è svolta dai componenti di attestatore, dal verificatore e dal manager dell'attestazione, nell'implementazione descritta nel dettaglio in questo modello [70]. L'attestatore è l'agente che dovrà verificare la validità dell'applicazione convertita, e viene definito da una etichetta che permette l'associazione nel codice. È possibile definire multipli attestatori per uno o più client che si connettono. Nella definizione dell'attestatore, bisogna usare una serie di parametri per indicare le caratteristiche che dovrà implementare. Alcuni degli argomenti che sono usati per definire l'attestatore sono la modalità di raccolta dei dati, l'algoritmo usato per generare i digest[28] dei dati di attestazione, due parametri per indicare come generare i nonce e come interpretarli e la frequenza temporale di invio di due richieste consecutive di attestazione. Gli algoritmi supportati dall'attestatore sono Blake2 [71], MD5 [72], SHA1 [6], SHA256 [39] e SHA512[29].
In questo tipo di attestazione vengono definite le aree di memoria riservate ai dati e al codice dell'applicazione da convertire, e le chiamate periodiche di attestazione compiute dal sistema servono a monitorare continuamente tutto ciò che è segnalato tramite le annotazioni del codice. L'annotazione usata deve anche indicare il nome dell'attestatore. È possibile indicare che l'attestazione della regione di memoria sia eseguita al lancio dell'applicazione.

La Figura 7.1 mostra i componenti di una applicazione modificata con ESP per implementare la funzionalità di attestazione remota statica. Un client e un server comunicano seguendo un protocollo organizzato per poter operare su un canale di comunicazione bidirezionale, basato sul protocollo WebSocket: il componente installato sul client che si occupa della comunicazione dal lato client prende il nome di *ASPIRE Client-side Communication Logic* (ACCL). Il rispettivo componente che funge da supporto al server, che deve ricevere le richieste dai dispositivi di apertura di una connessione, è invece l'*ASPIRE Server-side Communication Logic* (ASCL).

[28] Risultato dell'applicazione di un algoritmo di hash su dati restituito come output sotto il formato di stringa di dimensione fissa, secondo l'algoritmo impiegato.

[29] https://aspire-fp7.eu/sites/default/files/D3.04-Intermediate-Online-Protections-Report_v1.1.pdf

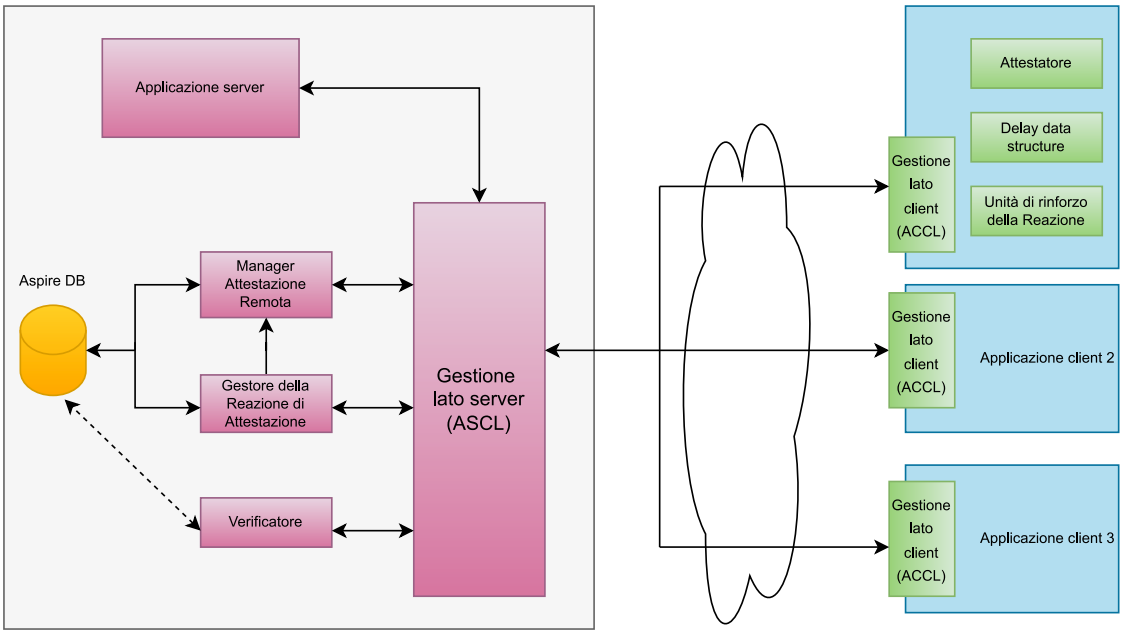


Figura 7.1: Schema dell'architettura di una applicazione con attestazione remota di client multipli

Le librerie condivise, attraverso le funzioni definite, permettono l'uso di tecniche di protezione come l'attestazione remota e la divisione del codice sulla comunicazione. Il client tramite il componente ACCL ha il compito di aprire la comunicazione con il server e chiama le API del canale WebSocket aperto. Le funzionalità del server possono anche essere assegnate ad un portale web ESP che gestirà multiple connessioni da parte di più client. Il server permette di effettuare tecniche di reazione online attraverso due componenti, il *Gestore della Reazione di Attestazione* e il *Manager di Attestazione Remota*: il primo componente prende decisioni in base ai valori che recupera dal Database, alle sue tabelle salvate e alle *Reaction policies*[30]. Il Manager di Attestazione Remota è incaricato di contattare i client che devono essere attestati secondo le policy scelte, e si basa su un *Manager RA Master* invocato dall'ASCL all'apertura o chiusura di un canale per richiedere dal database le informazioni dell'host con cui si dovrà iniziare il processo di attestazione. L'ASCL processa i dati ricevuti e li assegna ad un *Manager RA Slave* che sarà incaricato di gestire il processo con il componente Attestatore del client.

Attestazione dinamica

L'attestazione remota dinamica che implementa ESP si basa sulla verifica di misure calcolate in un preciso istante dell'applicazione eseguita dal client[31]. ESP usa una tecnica di nome *DynRA*, che consiste nel valutare lo stato di integrità di un

[30]Sono file di configurazione che descrivono la leggenda dei valori che le applicazioni devono avere per essere ritenute non compromesse.

[31]https://aspire-fp7.eu/sites/default/files/D3.09-ASPIRE-Online-Protections.pdf

binario eseguibile attraverso un criterio di invarianti: i valori delle variabili su cui sono basati gli invarianti devono restare durante il tempo dell'esecuzione immutati. Gli invarianti sono infatti un gruppo di asserzioni inserite in un programma che devono sempre risultare vere logicamente: DynRA verifica i valori di alcune variabili estraendole durante l'esecuzione, tramite lo strumento di ricerca di invarianti Daikon[32], e li usa in espressioni algebriche confrontando il risultato ottenuto con un valore aspettato. Per estrarli, sono inserite manualmente annotazioni nel sorgente e gli invarianti vengono aggiunti da componenti che iniettano funzioni che hanno lo scopo di verificare i valori e la logica di reazione, interrompendo l'esecuzione in caso di errori. Quando l'applicazione viene eseguita, e il server inizierà il processo di attestazione mandando la richiesta, verrà usato un nonce di 256 bit; il client userà il nonce in due possibili modi:

- come input dell'hash calcolato sui valori degli invarianti concatenati al nonce;
- come chiave per il calcolo del keyed-digest come HMAC [73] con una delle funzioni hash supportate.

Dal modello di DynRA, è stata costruita la tecnica di *Attestazione Remota Implicita* (Implicite Remote Attestation o IRA), suddivisa in IRA Semplice e IRA Dinamico Ibrido in base all'implementazione scelta. La tecnica IRA sfrutta la divisione del codice Client-Server per poter spostare alcune parti dell'applicazione su un server, costruito con tecniche di protezione per poter garantire che non sia costruito un server alias malevolo a cui reindirizzare la connessione, e calcolerà gli invarianti di un determinato istante sulla piattaforma remota. Quando i valori presi in esame per il calcolo degli invarianti, non più estratti automaticamente con strumenti esterni ma estratti da un sottoinsieme predefinito da cui è l'uomo a scegliere quali saranno usati, sono eseguiti sul server e i valori degli invarianti sono anche inviati al server per poter verificare l'esecuzione del client con le previsioni, si parla di *IRA Semplice*. Esiste l'approccio alternativo di *IRA Dinamico Ibrido* che usa componenti del modello di DynRA, come l'Iniettore di Funzioni ad-hoc nella applicazione originale e l'Estrattore di Invarianti Daikon, per selezionare automaticamente gli invarianti (e relative funzioni in cui sono calcolati) da monitorare, e dall'IRA Semplice prende le annotazioni IRA per applicare la divisione del codice da inserire nell'applicazione e la logica operativa della comunicazione tra il client e il server.
L'applicazione dell'attestazione remota tramite ESP offre dai vantaggi allo sviluppatore:

- la compatibilità con tutti i sistemi, conseguenza della scelta in fase di sviluppo di creare un framework che fosse indipendente dalle tecnologie che offrono un hardware apposito (inizialmente il progetto ASPIRE prevedeva infatti che ADSS dovesse funzionare per dispositivi mobile con processori ARM dal supporto limitato);
- il codice è modificato leggermente dallo sviluppatore, che può usare facilmente ESP e non deve rimodulare l'applicazione per integrare le funzionalità di protezione;

[32] https://plse.cs.washington.edu/daikon/

- l'implementazione scelta permette di proteggere l'applicazione da attacchi replay, dato che il verificatore associa ad ogni risposta di attestazione di una misura un nonce casuale e non replicabile.

In base al tipo di attestazione implementata, si avranno diversi svantaggi: DynRA soffre di vulnerabilità dal punto di vista della sicurezza offerta, non avendo implementato nessun tipo di canale di comunicazione sicuro, e per questo sono possibili attacchi di tipo spoofing e di tipo Man-in-the-middle; la tecnica di IRA Semplice invece limita l'implementazione della reazione del processo di attestazione, dato che la tecnica di divisione del codice modifica il codice in modo tale che non è possibile riscrivere la logica della reazione se non riscrivendo il codice del server e riapplicando la protezione. Inoltre, come già discusso nel Capitolo 6, l'aggiunta della divisione del codice dell'applicazione su un server contribuisce ad un degradamento delle performance, a causa dell'overhead dovuto all'uso della rete per la comunicazione e alle tempistiche di esecuzione del server.

Capitolo 8

Conclusioni

La tecnologia di Intel SGX permette di poter creare soluzioni di tipo Trusted Execution Environment, puntando all'enorme richiesta del mercato informatico di applicazioni che possono essere eseguite in modalità sicura da modifiche dovute ad attacchi esterni.
Il framework SGX è inizialmente sviluppato per trasformare in un processo automatico il codice sorgente di una applicazione scritta C in un sorgente che implementa le funzioni di librerie fornite dal Software Development Kit di Intel, permettendo allo sviluppatore di ridurre il lavoro necessario alla riscrittura manuale del codice. Ogni applicazione convertita ha così a disposizione una enclave dove poter conservare dati e eseguire funzioni in un contesto più protetto. L'obiettivo principale è stato usare il framework come punto di partenza per uno strumento migliorato rimuovendo le limitazioni presenti e aggiungendo nuove funzionalità.
La funzionalità più importante è sicuramente l'attestazione remota: tramite una notazione pragma da inserire nel codice sorgente, è possibile ora inserire in un punto specifico dell'applicazione il processo di attestazione secondo lo schema di Enhanced Privacy ID. Il ruolo del framework è sostituire le notazioni inserite nell'applicazione da convertire con le librerie di attestazione remota necessarie alla generazione del report del sistema che deve essere verificato. Analogamente, se l'utente ha una applicazione che comunica con quella convertita, il framework modificherà il codice ma usando librerie di ricezione del report. Questo processo è inserito nel flusso di esecuzione del framework già precedentemente realizzato, aggiungendo in alcuni punti nuove strutture dati che contengono i dati raccolti con gli strumenti di analisi statica del codice già inclusi: se non è richiesta una delle nuove funzionalità, le operazioni vengono ignorate.
Per rafforzare l'installazione della attestazione remota nell'applicazione, il framework SGX include la possibilità di usarla in modo combinato con la tecnica di protezione software nota come divisione del codice, presentata nel Capitolo 3: infatti, inserendo per ogni chiamata al server remoto che deve eseguire una delle funzioni spostate la richiesta di attestazione, si ha una sinergia che permette di individuare anche durante l'esecuzione modifiche non previste. Il codice viene quindi modificato per serializzare i dati che dovranno essere eseguiti remotamente, e viene costruito un server scritto in C con le funzioni segnalate. Il processo di attestazione dura indicativamente 2,24 secondi, quindi se le funzioni remote richieste non sono numerose, i tempi di esecuzione rimangono tollerabilmente nello stesso ordine di grandezza.

Tuttavia, bisogna ricordare che queste tempistiche dipendono anche da elementi esterni all'applicazione e al framework, come lo stato di disponibilità dei servizi di Intel, che non possono essere previsti, e i dati riportati si riferiscono sempre al miglior caso di esecuzione possibile.
Queste funzionalità arricchiscono il framework, che comprende altre modifiche come la scrittura di uno script makefile per la compilazione dell'eseguibile su un sistema Ubuntu e la generazione di una applicazione non più vincolata all'uso di una sola enclave: lo sviluppatore, a differenza del passato, potrà dichiarare diverse enclavi, e distinguere il loro uso differenziando i dati che devono proteggere e le funzioni che devono eseguire.
La modifica del framework ha portato ad un aumento del numero di moduli coinvolti e di operazioni di analisi svolte, a discapito di tempi di conversione più lunghi registrati. L'ottimizzazione del codice del framework può essere un obiettivo da raggiungere in futuro, così come la durata del processo di conversione della singola applicazione. Questo processo può essere velocizzato usando librerie dinamiche per diminuire il numero di librerie linkate prima della generazione dell'eseguibile.
Da questa versione del framework, adesso è possibile lavorare su nuove funzionalità, legate anche all'attestazione remota che ora è presente: con la rimozione del numero di enclavi, è necessario e tecnicamente possibile offrire, sempre tramite l'uso delle notazioni del codice, l'opzione di attestazione locale da usare o in alternativa alla remota o come procedura di riserva in caso di problemi di rete che bloccano in maniera non prevista l'utente per un problema esterno. Oltre ai futuri aggiornamenti di SGX che saranno rilasciati da Intel (la versione più recente, la 2.18.1 rilasciata a Gennaio 2023, comprende nuove API di gestione della memoria cache interna dell'enclave e il supporto al C++ 2017[1]), una possibile alternativa è l'attestazione usando lo schema di ECDSA: per realizzare questa strada, si prevede l'implementazione dell'applicazione da convertire all'interno di uno schema di container schierati nei nodi appartenenti ad un cluster fornito da Kubernetes. In questo modo, si contribuirà alla portabilità dell'applicazione, al momento è limitata a sistemi dotati di processori compatibili con Intel SGX: l'attestazione remota non è disponibile quando l'applicazione viene compilata in modalità simulazione dell'hardware Intel SGX, al contrario di altre funzionalità come il supporto alle enclavi.
Infine, è possibile anche aggiungere al framework il supporto ad applicazioni scritte in C++, supportate da SGX con gli ultimi aggiornamenti del Software Development Kit.

[1] https://download.01.org/intel-sgx/sgx-linux/2.18.1/docs/Intel_SGX_SDK_Release_Notes_Linux_2.18.1_Open_Source.pdf

Appendice A

Manuale utente

Lo scopo di questa Appendice è presentare per l'utente che vuole convertire l'applicazione come configurare il framework. Questa versione del framework è stata sviluppata considerando come Sistema Operativo principale Ubuntu, quindi tutti i comandi inseriti sono validi solo per quella distribuzione. Nella Sezione A.1, sono descritti i requisiti da soddisfare per poter usare il framework, compresi quelli dedicati al dispositivo hardware, con la procedura di installazione passo dopo passo per ciascuno dei componenti. Successivamente, nella Sezione A.2 viene mostrata l'esecuzione del framework descrivendo i passi da eseguire per generare una applicazione eseguibile sul dispositivo. Infine, nella Sezione A.3 è presente la lista degli errori che il framework può segnalare quando il processo di conversione non si comporta nel modo previsto, comprendendo sia gli errori aggiunti in questa versione del framework.

A.1 Installazione del framework

In questa Sezione, si procederà a descrivere come installare gli strumenti usati dal framework sul dispositivo dell'utente. Per poterlo eseguire, sarà necessario che l'utente soddisfi tutti i requisiti necessari, divisi in requisiti di sistema e requisiti di dipendenze software degli strumenti usati.

A.1.1 Requisiti di sistema

Il sistema è compatibile con il Sistema Operativo Ubuntu, dalla versione 18.04 in su, e in parte è compatibile con Windows. Le funzionalità inserite in questa versione del framework infatti non sono state testate per un sistema operativo diverso, mentre quelle inserite nella precedente versione rimangono compatibili. Il dispositivo che esegue l'applicazione convertita deve essere prima di tutto un processore Intel compatibile con la tecnologia SGX: il processore non serve per eseguire il framework, ma per la funzionalità di attestazione remota che non può essere riprodotta cambiando la compilazione del sorgente nella modalità di simulazione di SGX. La modalità simulazione è una opzione fornita da Intel per permettere di simulare via software l'uso di una enclave all'interno di una applicazione compilata con questa

opzione attivata, ma non può simulare l'uso della chiave privata fornita da Intel per eseguire la firma sull'enclave.
Per scoprire se il processore è compatibile, è possibile visitare la pagina web dedicata [1] o eseguire una applicazione disponibile su Github[2] che segnala se la CPU supporta la tecnologia di SGX e se è stata abilitata sul sistema tramite BIOS. Per poter eseguire l'applicazione, si può compilarla con un compilatore qualsiasi come `gcc`.
Il framework è lanciato con uno script di Python: è fondamentale che questo sia il primo strumento da installare. Per fare ciò, si può installare la versione più recente (3.10.2) dal sito ufficiale[3], dove è presente l'installer per qualsiasi Sistema Operativo usato dall'utente. Il framework Python importa moduli e strumenti esterni per eseguire specifiche funzioni, quindi è necessario anche installare questi strumenti. Per farlo, si usa il manager di installazione dei pacchetti `pip`: per installare pip, serve scaricare lo script di download dei file necessari di installazione `get-pip` [4], e poi lanciare il comando:

```
$ python get-py.pip
```

I moduli usati dal framework vanno installati dal terminale usando il comando

```
$ pip install <nomemodulo>
```

e la lista dei moduli obbligatori è la seguente:

- tqdm;
- logging3;
- regex;
- pydot.

Una volta chiamato pip per ciascuno di questi moduli, bisogna installare gli strumenti esterni Frama-C[5], Universal C-tags[6] e i compilatori gcc e g++. Ciascuno di questi strumenti ha multiple dipendenze esterne, quindi per l'installazione si seguono le istruzioni dei link allegati.

A.1.2 Requisiti compilazione applicazione

Il framework non genera solo i file sorgente dell'applicazione convertiti, e i file da compilare per creare l'immagine dell'enclave nell'estensione di una libreria dinamica: è generato il materiale per poter compilare automaticamente tutti i file,

[1] https://ark.intel.com/content/www/us/en/ark.html#\spacefactor\@m{}Processors

[2] https://github.com/ayeks/SGX-hardware

[3] https://www.python.org/downloads/

[4] https://bootstrap.pypa.io/get-pip.py

[5] https://frama-c.com/html/get-frama-c.html#

[6] https://github.com/universal-ctags/ctags

attraverso un solo unico comando, nell'eseguibile finale in modo tale che l'utente debba eseguire il minor numero possibile di operazioni. Il file generato è un Makefile che contiene tutte le regole di compilazione dei file, da lanciare da terminale di comando con il comando `make`. Tuttavia, la compilazione è legata al Software Development Kit (SDK) e alle librerie di SGX per eseguire determinate chiamate ECall: il framework deve conoscere dove le librerie si trovano, per poter creare i file oggetto dedicati alle funzioni SGX. Per compilare l'applicazione, oltre ai requisiti precedenti, è necessario soddisfare la dipendenza di altri componenti:

- OpenSSL;
- Intel SGX Software Development Kit;
- Intel SGX Platform Software Kit (PSW).

OpenSSL è necessario durante la compilazione delle librerie delle funzioni dell'attestazione remota, che importano le librerie di OpenSSL; il SDK permette di poter linkare le librerie di SGX invocate dalle funzioni e contiene strumenti necessari alla compilazione dell'enclave come l'edger8r, il traduttore del file EDL; infine, il PSW contiene diversi pacchetti, plugin e librerie per servizi usati dall'applicazione convertita con SGX, come il servizio che permette di usare la Quoting Enclave per generare le firme sulla Quote e il servizio di attestazione secondo schema EPID. Per installare OpenSSL, bisogna seguire i seguenti passi:

1. Aprire il terminale e installare Perl con i seguenti comandi:

   ```
   $ sudo apt update
   $ sudo apt upgrade
   $ sudo apt install perl
   ```

2. Scaricare dal sito ufficiale[7] l'ultima versione di OpenSSL disponibile e scompattare il contenuto dell'archivio dove si desidera installare i file;

3. Aprire il terminale e eseguire i seguenti comandi:

   ```
   $ ./config
   $ make
   $ make test
   ```

4. Se l'installazione è andata a buon fine nel precedente passo, il risultato finale sarà *PASS*. A questo punto, si termina l'installazione di OpenSSL con il comando:

   ```
   $ make install
   ```

Per l'installazione del Software Development Kit, è necessario seguire queste operazioni:

[7] https://www.openssl.org/source/

1. installare le dipendenze lanciando il seguente comando:

```
$ sudo apt-get install build-essential ocaml ocamlbuild
    automake autoconf libtool wget python-is-python3
    libssl-dev git cmake perl
```

2. installare l'Intel SGX Driver[8], soddisfando prima i pre-requisiti di dipendenza da altri software attraverso il lancio dei comandi:

```
$ sudo apt-get install build-essential ocaml automake
    autoconf libtool wget python3 libssl-dev dkms
$ sudo apt-get install build-essential python-is-python3
$ sudo update-alternatives --install /usr/bin/python python
    /usr/bin/python3 1
```

3. scaricare dal repositorio pubblico di Intel[9] l'ultima versione del file binario di installazione del Intel SGX Driver, con il comando:

```
$ wget https://download.01.org/intel-sgx/latest/dcap-latest
/linux/distro/
${distro_name}/sgx_linux_x64_driver_${version}.bin
```

   usando come distro_name "ubuntu18.04-server","ubuntu20.04-server" o "ubuntu22.04-server" a seconda della versione che si utilizza e come valore per il parametro version la versione corrente del driver (1.41 al 9 Marzo, come si vede in Figura A.1);

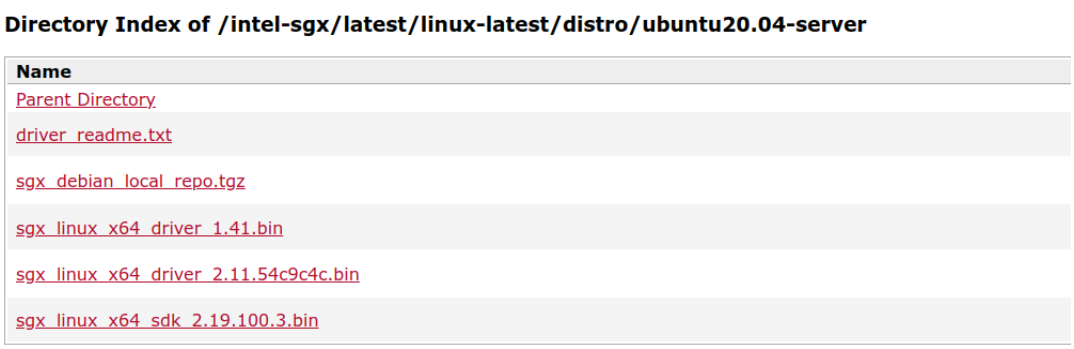


Figura A.1: Repositorio pubblico di Intel per scaricare il driver e il binario del SDK

4. cambiare i permessi di esecuzione del file, e eseguirlo con i seguenti comandi:

```
$ sudo ./sgx_linux_x64_driver_${version}.bin
$ chmod 777 sgx_linux_x64_driver_${version}.bin
$$ ls -la /dev/sgx*
```

[8]Necessario per installare i pacchetti software, lo si installa come Driver Digital Center Attestation Primitives(DCAP)

[9]https://download.01.org/intel-sgx/latest/linux-latest/distro/ubuntu20.04-server/

l'ultimo comando è da usare solo dopo il riavvio del sistema, per verificare che il driver sia stato caricato correttamente: in caso di installazione del driver positiva, si dovrebbe vedere il risultato della Figura A.2.

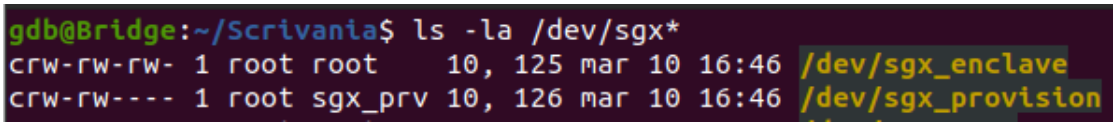


Figura A.2: Installazione dell'Intel SGX Driver riuscita

5. aggiornare la lista delle repository sicure del sistema con quella di intel-sgx, aggiungendo la chiave pubblica della sua repository all'archivio di chiavi sicure usate da apt per autenticare i pacchetti:

```
$ echo 'deb [arch=amd64]
    https://download.01.org/intelsgx/sgx_repo/ubuntu focal
    main' | sudo tee /etc/apt/sources.list.d/intel-sgx.list
$ wget -qO - https://download.01.org/intel-sgx/
sgx_repo/ubuntu/intel-sgx-deb.key | sudo apt-key add
```

Se il processo va a buon fine, si vedrà un messaggio di conferma, come in Figura A.3

Figura A.3: Conferma dell'installazione della repository riuscita.

6. Installare i seguenti pacchetti per eseguire una applicazione che usa le enclavi:

```
$ sudo apt-get update
$ sudo apt-get install libsgx-epid libsgx-quote-ex
    libsgx-dcap-ql
```

7. Si deve installare ora il SDK per gli sviluppatori, quindi scaricare il file di installazione dalla repository pubblica (vedere Figura A.1), cambiare i suoi permessi di esecuzione ed eseguire il file (usando il comando sudo se necessario):

```
$ wget -
    https://download.01.org/intel-sgx/latest/linux-latest
/distro/ubuntu20.04-server/sgx_linux_x64_sdk_2.19.100.3.bin
$ chmod +x sgx_linux_x64_sdk_2.19.100.3.bin
$ ./sgx_linux_x64_sdk_2.19.100.3.bin
```

Seguendo il setup di installazione, verrà richiesto se si vuole installare l'SDK nel direttorio in cui il terminale è eseguito o in un'altra posizione del file system. Un direttorio alternativo dove installarlo è nella root dell'utente, rappresentata anche con il carattere " ";

8. entrando nella cartella in cui si è scelto di installare il SDK, aggiungere questo percorso nei PATH usati dal sistema operativo con il comando:

```
$ source <User Input Path>/sgxsdk/environment
```

9. installare gli ultimi componenti in versione sviluppatore, con l'ultimo comando:

```
$ sudo apt-get install libsgx-enclave-common-dev
   libsgx-dcap-ql-dev libsgxdcap-default-qpl-dev
```

A.1.3 Requisiti attestazione remota

Come descritto nella Sezione precedente, il file di configurazione del framework `frameworkConfig` include parametri usati per la funzionalità di attestazione remota. I parametri `signerSPID`, `iasPrimaryKey` e `iasSecondaryKey` sono infatti credenziali fornite dopo che il gestore dell'applicazione server (denominato anche Service Provider) si iscrive al servizio di attestazione Intel tramite schema EPID. L'iscrizione è infatti un passaggio obbligatorio per poter usare le API che richiedono al servizio di attestazione Intel IAS le verifiche sulle credenziali crittografiche dell'enclave e del dispositivo da attestare: se si inseriscono nel sorgente le notazioni dedicate all'attestazione remota, e nessun valore è inserito per quei parametri, il framework lancerà un errore che segnala quale parametro manca nella configurazione e interromperà la conversione (riferirsi alla Sezione A.3). Inoltre, il framework effettua anche dei controlli per verificare che i parametri sono stringhe nel formato previsto di 32 caratteri ciascuno.
Per potersi iscrivere, bisogna registrarsi al sito ufficiale Intel[10], in modo tale da ottenere la licenza per le applicazioni in fase di sviluppo (development) o in fase di produzione. Dal sito ufficiale, è anche possibile scaricare il certificato X509 *Attestation Report Root CA*. Dopo essersi iscritti, bisogna fare la sottoscrizione al tipo di applicazione che si vuole sviluppare, e inserire le credenziali che vengono fornite all'interno del file di configurazione.

A.2 Uso del framework

In questa Sezione, si descrive come l'utente deve usare il framework per convertire l'applicazione nel modo desiderato. Sono elencate le notazioni supportate dal framework, e le operazioni da fare per ottenere l'eseguibile finale.

A.2.1 Modifica codice sorgente

Il primo passo da fare è modificare il codice sorgente dell'applicazione inserendo le notazioni desiderate. L'elenco delle funzionalità che l'utente può inserire tramite le

[10] https://api.portal.trustedservices.intel.com/EPID-attestation

notazioni è il seguente: conversione di una funzione in una funzione ECall o una funzione OCall, assegnandola ad una enclave tra più generabili; copia o spostamento dei dati; inserimento di funzioni di attestazione remota di un client o di un server; trasferimento di funzioni su una applicazione server. Per la conversione di una funzione, si usa la seguente formula prima della definizione o della dichiarazione:

```
#define sgx_E#_ecall_<nome_funzione> (<[nome_param>,
   <tipo_copia>, <dim_buf>]..)
#define sgx_E#_ocall_<nome_funzione> (<[nome_param>,
   <tipo_copia>, <dim_buf>]..)
```

In base al tipo di funzione da definire, si usa la prima o la seconda formula; se uno degli argomenti della funzione deve essere copiato per valore, all'interno delle parentesi bisogna non inserirlo. Al contrario, se l'argomento è un puntatore a un dato o ad un vettore statico, bisogna usare la sintassi indicata all'interno delle parentesi per ciascun argomento di quel tipo: al nome del parametro, va seguita la modalità di copia desiderata, e il numero di bytes che deve essere allocato per il buffer creato allo scopo di trasferire i dati (nel Capitolo 5 sono presenti esempi di applicazione). Con questa sintassi, è anche possibile distinguere le enclavi che l'applicazione deve usare, e associarle alla funzione appena convertita, attraverso il prefisso successivo alla prima parola "sgx".
Per poter copiare i dati (funzioni o variabili) all'interno di una enclave, o per poterli spostare dalla memoria non protetta a quella riservata, si usa la seguente coppia di notazioni:

```
#pragma move_start_E#
    <dati da trasferire>
#pragma move_end_E#

#pragma copy_start_E#
    <dati da copiare>
#pragma copy_end_E#
```

Queste notazioni possono essere usate sulle variabili globali, e si usano per delimitare la parte di codice che deve essere spostata. Anche in questo caso, è presente come sottostringa l'identificatore dell'enclave a cui dover associare i risultati delle modifiche svolte.
Le notazioni di attestazione remota si dividono in quelle da usare per una applicazione che desidera essere attestata (occupando il ruolo di client che si connette ad un attestatore) e quelle da usare per l'applicazione che desidera che un client che lo contatti sia attestato, ricevendo le sue credenziali crittografiche e inoltrandole all'IAS per ricevere le informazioni delle operazioni di autenticità. Nel primo caso, la sintassi seguita è la seguente:

```
#pragma attest_enclave_<id enclave>
```

Questa notazione va inserita nel punto dell'applicazione in cui si desidera che venga iniziato il processo di attestazione. Il framework sostituisce la notazione con il codice che esegue le operazioni del processo di attestazione, inserendo all'inizio del sorgente dell'applicazione le direttive #include dei file header appartenenti alle

librerie di attestazione remota necessarie al client.
Per quanto riguarda la notazione dell'attestazione remota per il server, la versione analoga ha questa sintassi:

```
//codice della funzione
#pragma server_attestation
//codice della funzione
```

Anche in questo caso, la notazione deve essere inserita nel punto in cui si desidera che il server chiami le funzioni che aprono il processo di attestazione remota su una porta che può essere specificata dal file di configurazione. In questo caso, non serve indicare nessuna enclave, dato che le operazioni fatte non prevedono l'uso di chiamate del SDK SGX.
Per le notazioni della divisione del codice invece, bisogna inserire delle notazioni che indicano quali sono le funzioni che devono essere spostate sul nuovo codice che verrà costruito. La sintassi da rispettare è:

```
#pragma move_to_server
int foo(){
    //codice implementativo funzione foo
}
```

La notazione va inserita prima della implementazione della funzione, e non all'interno del codice. A questo punto, il codice è pronto per essere modificato dal framework.

A.2.2 Configurazione framework e esecuzione

Il framework deve conoscere alcune informazioni per poter convertire l'applicazione, per poterle inserire nel sorgente delle librerie di attestazione remota. Usando questo file di configurazione, è possibile cambiare alcuni dati senza dover modificare manualmente il codice e rieseguendo il framework se si vogliono implementare nuove modifiche. Il file di configurazione è un modulo Python dal nome `frameworkConfig`, e contiene una lista di valori personalizzabili dall'utente. Il modulo principale del framework, `framework`, importa questo modulo per conoscere le opzioni richieste, ottenendo la seguente lista di parametri:

- files, il file sorgente dell'applicazione da convertire (obbligatorio);
- rootFolder, indica la cartella dove si trova il file sorgente (obbligatorio);
- SGX_SDK, indica la cartella di installazione del Software Development Kit di SGX (obbligatorio);
- signerSPID, rappresenta l'Intel Service Provider ID, stringa esadecimale di 32 bit usata nell'attestazione remota (obbligatorio per l'attestazione remota);
- iasPrimaryKey, credenziale primaria del servizio che contatta l'IAS per l'attestazione remota (obbligatorio per l'attestazione remota);

- iasSecondaryKey, credenziale secondaria del servizio che contatta l'IAS per l'attestazione remota (obbligatorio per l'attestazione remota);
- iasReportFilePath, è il path assoluto per indicare la posizione del certificato usato con l'IAS per l'attestazione remota (obbligatorio per l'attestazione remota);
- autorun, un flag che si attiva per compilare automaticamente i file sorgente dopo la conversione;
- serverToConnect, indica a quale indirizzo l'applicazione client si deve collegare (di default, si collega all'indirizzo di loopback);
- portAttestationProcess, indica su quale porta si desidera che sia lanciato il servizio di attestazione remota;
- portServerCS, indica su quale porta il server creato con la divisione del codice deve ascoltare le connessioni;
- MRENCLAVE, valore di debug per saltare i controlli di verifica della misura dell'enclave;
- customReaction, è una tupla di valori che indicano rispettivamente il nome della libreria che include la reazione personalizzata in caso di attestazione non andata a buon fine, e la sua posizione nel file system.

I primi due parametri sono obbligatori per la conversione dell'applicazione. I parametri signerSPID, iasPrimaryKey, iasSecondaryKey e iasReportFilePath sono obbligatori per compilare le applicazioni che sono state modificate per introdurre la funzionalità di attestazione remota, descritte specificatamente nella Sezione A.1.3. Se non sono inseriti ma non è stata richiesta l'attestazione remota, il framework non lancerà nessun errore.
L'utente che desidera lanciare il framework modifica quindi i dati all'interno del file di configurazione, come è possibile vedere in Figura A.4, in cui è evidenziato il file da convertire 'banner.c', la cartella e dove si trova il SDK.

Con le notazioni inserite nel sorgente, e con la sua posizione inserita nel file di configurazione del framework, è possibile iniziare il processo di conversione, aprendo il terminale nella cartella dentro `framework_code.zip` e lanciare:

```
$python framework.py
```

Durante la conversione, lo stato della conversione sarà rappresentato da una barra di caricamento, gestita dal modulo `tqdm`, e il modulo `logging` scriverà i messaggi di debug all'interno di un file testuale `logfile.txt` che si trova nella stessa cartella in cui il framework è stato lanciato.
Quando la conversione termina, il file testuale sarà spostato nella nuova cartella che restituisce l'output, `generated_trusted`: al suo interno, sarà presente il file di logging, la cartella `application`, le cartelle `enclaveXX` per ogni enclave creata dal processo di conversione, le librerie di attestazione remota `remote_attestation` e/o `server_material` se nel sorgente erano state inserite le notazioni dell'attestazione e il Makefile per compilare tutti i file.

```
files = ['banner.c']
rootFolder = 'sources'
SGX_SDK = '~/sgxsdk'
# [ONLY SERVER, REQUIRED]
signerSPID = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
# [ONLY SERVER, REQUIRED]
iasPrimaryKey = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
# [ONLY SERVER, REQUIRED]
iasSecondaryKey = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
# [ONLY SERVER, REQUIRED]
iasReportFilePath =
    './server_material/IAS_Report_Signing_CA_File.pem'
autorun = False
serverToConnect = ''
portAttestationProcess = 7777
portServerCS = 65433
#DEBUG#
MRENCLAVE = '22112021221120212211202122 1120'
customReaction = "helper","/python/addon_files/"
```

Figura A.4: Esempio di file di configurazione del framework

A.2.3 Compilazione applicazione

Per la compilazione dell'applicazione è necessario che i componenti descritti nella Sezione A.1.2 siano stati installati.
Il risultato della fase di conversione viene salvato all'interno della cartella generated_trusted. Nella cartella `application`, è possibile trovare all'interno il sorgente dell'applicazione modificato dal framework e che corrisponde all'applicazione eseguita nel contesto di esecuzione non protetto, e per ogni enclave il file di dichiarazione e di implementazione enclave_manager_X.h e .c. All'interno di questi file, sono definiti i metodi che gestiscono l'enclave nell'applicazione con l'operazione di creazione della memoria riservata all'enclave caricando la sua immagine salvata sul disco, l'operazione di cancellazione delle risorse allocate per l'enclave e un metodo per controllare che il dispositivo su cui è eseguita l'applicazione è compatibile con SGX.Quando l'enclave è caricata in memoria, le librerie di SGX ritornano un id associato all'enclave, che serve al processore durante il cambio di contesto per capire a quale enclave si riferisce, ed è passato infatti come argomento delle funzioni di più basso livello di SGX.
Il framework crea infine anche un altro file chiamato `enclave_manager_common`, solo se l'utente ha richiesto per l'applicazione la funzionalità di attestazione client: il file include le librerie di attestazione remota, e contiene un metodo che lancia le API di attestazione passando un unico parametro, l'id dell'enclave che vuole essere attestata. In questo modo, non si deve definire per ogni enclave una funzione dedicata all'attestazione, ma è definita una sola funzione di attestazione chiamata da più enclavi, che distingue l'enclave da attestare con l'id che riceve.

La cartella enclave ha al suo interno il file sorgente `enclave.c` che contiene il codice relativo alla parte protetta dell'applicazione (i dati e le funzioni spostate con le notazioni, le istruzioni eseguite con le ECall, il codice della OCall che chiama la funzione corrispettiva sul lato esposto dell'applicazione e attende il suo risultato), il file di configurazione dell'enclave `enclave.config.xml`, il rispettivo file EDL `enclaveX.edl` che sarà tradotto dallo strumento edger8r per poter creare le funzioni proxy di cambio contesto (vedesi Sezione 4.1.2), la chiave privata RSA da 3072 bit `enclave_private_test.pem` con cui l'immagine dell'enclave viene firmata per generare l'immagine definitiva durante la compilazione dell'applicazione, e un file di scripting per la fase di linking `enclave.lds`. Si nota che la chiave usata per la firma è la stessa per tutte le enclavi, ma il SDK Intel permette di poter usare diverse chiavi per diverse enclavi o di usarne una generata in quel momento.
Il Makefile scritto dal framework permette automaticamente di completare la compilazione dell'applicazione, con i seguenti passi:

1. entrare nella cartella generated_trusted;
2. aprire il terminare e lanciare il comando `make`

Il Makefile chiama lo strumento sgx_edger8r per tradurre il file EDL secondo il procedimento descritto nella Sezione 4.1.2, poi genera l'immagine dell'enclave "libenclaveXX.signed.so" tramite l'operazione di firma della chiave privata e gli eseguibili. L'eseguibile appartenente all'applicazione convertita, lanciabile dal terminale di comando, avrà lo stesso nome del file sorgente. Se l'applicazione convertita ha creato anche un server su cui sono state trasferite le funzioni con la divisione del codice, con l'esecuzione dello script viene creato anche l'eseguibile dell'applicazione server `server_cs`, da eseguire insieme all'applicazione originale convertita, per il suo corretto funzionamento.
Si nota che, nel caso in cui il flag `autorun` è stato attivato dal file di configurazione del framework prima di lanciarlo, la fase di compilazione viene eseguita automaticamente, saltando le operazioni descritte.

A.3 Codici di errori

Quando il framework trova errori durante il processo di conversione dell'applicazione, restituisce in output un codice identificativo e la causa dell'errore. In questa Sezione, saranno presentati tutti i codici d'errori che il framework può causare, sia quelli implementati nella precedente versione del framework, sia mostrando i nuovi errori inseriti per l'occasione:

#**A001** :Il framework ha trovato la definizione di più macro per una stessa funzione <nomeFunzione>.

#**A002** :Il framework ha trovato la funzione <nomeFunzione> referenziata all'interno di una macro, ma la definizione della funzione specificata non è stata trovata nel file.

#**A003** :Il framework non ha trovato la funzione <nomeFunzione> all'interno del contenuto salvato nel dizionario.

#**A004** :Il framework non ha trovato la struttura <nomeStruttura> all'interno del contenuto salvato nel dizionario.

#**A005** :Il framework non ha trovato la variabile enumerativa <nomeEnumerazione> all'interno del contenuto salvato nel dizionario.

#**A006** :Il framework non ha trovato la variabile di tipo union <nomeUnion> all'interno del contenuto salvato nel dizionario.

#**B001** :Nel codice non è presente nessuna macro per definire una ECall e non sono state richieste funzionalità aggiuntive, inserire macro.

#**B002** :Il codice sorgente è stato scritto con una sintassi non compatibile con lo standard ANSI C11.

#**B003** :Il codice esposto e il codice protetto dall'enclave condividono l'uso della stessa funzione <nomeFunzioneCondivisa>, la funzione <nomeFunzioneProtetta> non deve usare la funzione <nomeFunzioneCondivisa>.

#**B004** :Il framework ha trovato il parametro <tipoDatoParametro> <nomeParametro> referenziato senza che sia stato definito.

#**B005** :Il framework ha trovato una funzione ECall e una funzione OCall condividere lo stesso nome, usare nomi distinti per le funzioni definite nelle macro.

#**B006** :Il framework non ha trovato il file indicato nella configurazione.

#**B007** :Non è possibile applicare la divisione del codice senza aver creato almeno una enclave, definire una con la notazione sgx_E<idEnclave>_ecall_<nomeFunzione>.

#**C001:** Il framework non è riuscito a invocare C-Tags, bisogna ripetere l'installazione seguendo le istruzioni nell'Appendice.

#**C002:** Il framework ha trovato all'interno del file una struttura dati definita con un nome usato per definire un'altra struttura dati, i nomi non devono essere condivisi tra stutture diverse.

#**D001:** Il framework non è riuscito invocare Frama-C, bisogna ripetere l'installazione seguendo le istruzioni nell'Appendice.

#**D002:** Nel file sorgente <nomeFile> non è stata trovata la funzione <nomeFunzione>, è presente solo la sua dichiarazione.

#**D003:** L'invocazione di Frama-C ha trovato errori con il file <nomeFile>.

#**E001:** Il framework ha provato a inserire la funzione protetta <nomeFunzione> nel file EDL, ma era già presente nella sezione delle ECall.

#**E002:** Il framework ha provato a inserire la funzione non protetta <nomeFunzione> nel file EDL, ma era già presenta nella sezione delle OCall.

#**F001:** Il framework ha trovato una funzione ECall o OCall definita statica: togliere l'attributo per la funzione <nomeFunzione>.

#**F002:** Il framework ha trovato una funzione ECall o OCall che non restituisce un dato di tipo intero, si modifichi la funzione <nomeFunzione>.

#**F003:** Il framework ha trovato una funzione ECall o OCall che usa un vettore di tipo dinamico, non permesso da SGX: si cambi il vettore in un puntatore.

#**F004:** Il framework ha trovato una funzione ECall o OCall con un argomento costante che non sia un un puntatore, si deve rimuovere la parola chiave const.

#**F005:** Il framework ha trovato una macro che non contiene il numero corretto di elementi, al posto di <numeroArgomentiPrevisti> la funzione <nomeFunzione> ha <numeroArgomentiDefiniti>.

#**F006:** Il framework ha trovato un errore nella definizione dell'argomento <nomeArgomento> per la funzione <nomeFunzione>.

#**F007:** Il framework ha trovato un errore nella definizione dell'argomento <nomeArgomento> della dimensione del buffer di copia per la funzione <nomeFunzione>.

#**F008:** Il framework ha trovato un errore nella definizione dell'argomento <nomeArgomento> usato per definire la dimensione del buffer di copia per la funzione <nomeFunzione>: non è di tipo int.

#**F009:** Il framework ha trovato un errore nella definizione dell'argomento <nomeArgomento> usato per definire la dimensione del buffer di copia per la funzione <nomeFunzione>: è usato un tipo puntatore int o vettore int, cambiarlo in un tipo esclusivamente int.

#**F010:** Il framework ha trovato un errore nella definizione dell'argomento <nomeArgomento> contenuto dentro la macro SGX: l'argomento non è stato trovato all'interno della funzione <nomeFunzione>.

#**F011:** Il framework ha trovato l'argomento <nomeArgomento> usato per definire la dimensione del buffer di copia per la funzione <nomeFunzione> definito per un puntatore, ma non ha senso specificare la dimensione di un puntatore.

#**F012:** Il framework ha trovato l'argomento <nomeArgomento> usato per definire la dimensione del buffer di copia per la funzione <nomeFunzione> definito senza l'uso di puntatori.

#**F013:** Il framework ha trovato l'argomento <nomeArgomento> della funzione <nomeFunzione> definito con puntatori multipli ma senza la modalità di copia "u".

#**F014:** Il framework ha trovato l'argomento <nomeArgomento> usato per definire la dimensione del buffer di copia per la funzione <nomeFunzione> con l'opzione di copia user_check: con questa opzione, la dimensione non deve essere specificata.

#**F015:** Il framework ha trovato per la funzione <nomeFunzione> una opzione di copia <nomeOpzione> non esistente.

#**F016:** Il framework ha trovato l'uso di un tipo di ritorno non supportato tra quelli disponibili per la funzione <nomeFunzione>.

#**G001:** Il framework non è riuscito invocare il compilatore GCC, bisogna ripetere l'installazione seguendo le istruzioni nell'Appendice.

#**G002:** Il framework non è riuscito a caricare la configurazione delle librerie SGX. Verificare l'integrità e riprovare la conversione, rimuovendo dal file tutte le funzioni che non sono di SGX.

#**G003:** Il framework ha trovato nel file sorgente <nomeFile> un errore di sintassi.

#**G004:** Il framework non ha trovato la funzione <nomeFunzione> all'interno delle librerie SGX, si deve usare una funzione OCall per invocarla.

#**G005:** Il framework non ha trovato il file di configurazione di GCC all'interno della cartella `include`, si deve rieseguire il framework per generarlo.

#**G006:** Il file di configurazione del framework non contiene nessuno dei parametri necessari per la comunicazione con l'Intel Attestation Service: controllare che siano presenti valori per i parametri signerSPID, iasPrimaryKey,iasSecondaryKey e iasReportFilePath.

#**G007:** Il file di configurazione del framework contiene un Service Provider ID dalla dimensione minore di 32 Bytes: inserire uno Service Provider ID della corretta lunghezza.

#**G008:** Il file di configurazione del framework contiene una Chiave Principale IAS di Sottoscrizione dalla dimensione minore di 32 Bytes: inserire una Chiave Principale IAS di Sottoscrizione della corretta lunghezza.

#**G009:** Il file di configurazione del framework contiene una Chiave Secondaria IAS di Sottoscrizione dalla dimensione minore di 32 Bytes: inserire una Chiave Secondaria IAS di Sottoscrizione della corretta lunghezza.

#**G010:** Il file di configurazione del framework non ha trovato la cartella del Software Development Kit di Intel usando la posizione specificata in SGX_SDK: la cartella non esiste.

#**H001:** Il framework ha trovato per la funzione <nomeFunzione> argomenti dal tipo non elementare: questa funzione non può essere spostata su un server, non essendo entrocontenuta.

#**H002:** Il framework ha trovato per la funzione <nomeFunzione> dipendenze da variabili globali: questa funzione non può essere spostata su un server, non essendo entrocontenuta.

#**H003:** Il framework ha trovato per la funzione <nomeFunzione> l'uso di variabili copiate per riferimento invece di copiarle per valore: questa funzione non può essere spostata su un server, non essendo entrocontenuta.

Appendice B

Manuale Programmatore

Lo scopo di questa Appendice è descrivere l'implementazione del framework dal punto di vista descrittivo del codice. La Sezione B.1 si concentra sui moduli più importanti del framework, giustificando anche le modifiche eseguite rispetto alla versione precedentemente sviluppata.

B.1 Descrizione tecnica dei moduli

Il framework segue il modello e il flusso di esecuzione presentati nel Capitolo 5, il quale non scendeva nel dettaglio dell'implementazione a livello di codice. Le dipendenze dagli strumenti esterni sono descritte già nella precedente Appendice, e non è stato aggiunto nessun nuovo strumento per implementare le funzionalità inserite. Tutte le modifiche infatti hanno riguardato i moduli precedenti, inserendo un modulo contenente operazioni per l'attestazione remota (`remoteAttestationInterface`), uno per le operazioni usate per la divisione del codice (`codeSplittingInterface`) e uno usato per contenere sezioni di codice da inserire nella scrittura dei file generati dal processo (`constants`).
La descrizione si concentra sui moduli che hanno ricevuto le modifiche più importanti e sono fondamentali per il processo di conversione e di aggiunta di funzionalità di sicurezza del framework, approfondendo alcuni passaggi del processo per poter spiegare a cosa servono determinati metodi.

B.1.1 codeAnalyser

In questo modulo, è presente il metodo `main` dedicato al processo principale di conversione: esso richiede agli altri moduli di svolgere specifiche funzioni e specifici controlli, e riceve per argomenti tutte le informazioni che sono salvate nel file di configurazione.
Il metodo main per prima cosa controlla che ci sia un sorgente corretto da analizzare, e salva tutte le informazioni delle strutture dati usate all'interno del sorgente, delle funzioni e l'intero codice implementativo all'interno di un oggetto chiamato Dizionario, attraverso i metodi del modulo cFileReaderWriter (come specificato nella Sezione B.1.2). Dopo aver fatto ciò, cerca notazioni di richiesta di divisione

del codice: se trova almeno una funzione, attiva il flag `codeSplittingActivated` per poter eseguire successivamente le operazioni di divisione del codice e memorizza anche il numero di funzioni che richiedono lo spostamento sulla nuova applicazione. L'attestazione remota viene riconosciuta dal framework con un altro flag che distingue il tipo di attestazione remota richiesta associando ad un valore intero il tipo richiesto: ricerca nel codice, tramite espressioni regolari, una delle due notazioni di attestazione remota definite, e se trova la direttiva "#pragma attest_enclave " setta il flag `remote_attestation` a 1; se viene trovata la notazione "#pragma server_attestation" il flag è settato a 2, e se nessuna notazione è individuata allora il valore restituito dalla funzione di ricerca è zero. La divisione del codice è implementata in modo tale da essere eseguita con l'attestazione remota, e il flag codeSplittingActivated permette alla fine del processo di conversione di includere tutte le funzioni e le librerie del processo di attestazione remota del client e del server. In questo caso, se non è stata inserita nessuna notazione di attestazione del client, viene indicata come enclave che deve essere attestata la prima, per evitare errori di esecuzione. In base al valore del flag remote_attestation, il framework eseguirà operazioni diverse, distinguendo le operazioni da fare in base alla funzionalità da inserire.
Il processo di generazione del codice relativo all'attestazione remota del client è più complesso di quello analogo per il server, che non necessita di usare un ambiente di esecuzione protetto con SGX per eseguire l'attestazione: è solo il client che per ottenere la Quote deve usare funzioni del SDK offerto da Intel. Dopo aver compiuto le operazioni di conversione e trasformazione del codice in codice compatibile con SGX per ogni enclave da generare per l'applicazione, il processo genera i file delle nuove funzionalità, partendo da template memorizzati nei moduli come `Constants`, e li inserisce insieme alle librerie e ai file dedicati alla compilazione del binario eseguibile nella cartella `generated_trusted`.

B.1.2 cFileWriterReader

Il modulo serve per copiare il codice del file sorgente e poterlo modificare, in modo tale da salvare il contenuto modificato nel nuovo file sorgente. Il contenuto del file sorgente viene infatti salvato all'interno dell'oggetto che questo modulo crea, all'interno dell'attributo stringa di nome `contents`, come una unica stringa chiamata anche *stringa R*[1]. Questo oggetto creato è riferito all'interno del codice come "Dizionario", perché oltre a contenere il codice implementativo, salva anche le strutture dati usate dall'applicazione:

- funzioni;
- struct;
- enumerazioni;

[1]Conosciuta anche come stringa Raw o stringa costante manifesta, vengono indicate antecedendo prima dei caratteri di apici la lettera "r" minuscola o maiuscola. Esse permettono di mantenere la formattazione originale del testo, con gli stessi caratteri di tabulazione, e di poter modificare il contenuto in maniera diretta e letterale.

- librerie incluse;
- variabili definite con la direttiva define;
- variabili di tipo unioni;

I contenuti ottenuti durante la lettura del codice saranno poi salvati all'interno di liste salvate in attributi del dizionario, in modo tale da rendere accessibile ogni dato memorizzato scorrendo le liste o usando chiavi di ricerca (ad esempio, si può usare il nome di una funzione per cercare la funzione con quel determinato nome, e ottenere l'oggetto Funzione). Tutte le operazioni di analisi delle funzionalità da inserire vengono invocate chiamando i metodi di ricerca rispettivi (`findFunctionRequestingClientAttestation` per la ricerca lato client, per la ricerca lato server `findFunctionRequestingServerAttestation` e per la ricerca di più enclavi `searchForEnclaves`). Lo scopo è quello di usare le espressioni regolari per cercare all'interno del codice del sorgente le occorrenze delle notazioni inserite dall'utente usando i pattern segnalati nella Figura B.1.

```
(#\s*pragma\s*attest_enclave_\d+)
(#\s*pragma\s*SERVER_ATTESTATION)
(^\s*#[\s]*define\s*sgx_E[\d]+_(ecall|ocall))
```

Figura B.1: Pattern da usare per la ricerca di notazioni con espressioni regolari

Se lo sviluppatore desidererà modificare le notazioni che bisogna inserire, dovrà anche modificare questi pattern cercati all'interno delle funzioni prima citate. I metodi restituiscono la lista delle funzioni che devono essere modificate trasformandole in funzioni compatibili con la tecnologia SGX, e la lista delle enclavi richieste dal codice. Dato il numero di operazioni che il framework deve eseguire all'interno del codice delle funzioni, sono disponibili metodi per aggiornare il dizionario usato dal processo, sostituendo una funzione con una nuova (utile nella fase di divisione del codice in cui si deve inserire il codice che serializza i parametri da inviare al server), sostituendo il codice implementativo e aggiungendo le strutture dati supportate come le librerie da includere (quando il framework deve aggiungere la libreria dell'attestazione remota).

B.1.3 Function

Dal punto di vista dello sviluppatore, la struttura permette di modificare gli attributi di una funzione in maniera molto diretta. Il modulo Python gestisce infatti gli attributi interni di un oggetto definito con stringhe normali, stringhe R e stringhe F che permettono di concatenare dati diversi in un unica stringa, e di scomporli facilmente in altri dati.

B.1.4 edlFileWriter

Il modulo è usato dal framework per interfacciarsi con il file `enclave.edl`, consentendo la gestione della scrittura del suo contenuto e trasferendo al suo interno i dati delle funzioni pronte per essere convertite. Il modulo gestisce una lista delle funzioni da inserire nel blocco "trusted" e una lista di funzioni per il blocco "untrusted", inizialmente vuota, ma aggiungendovi le funzioni che hanno come attributo interno la definizione SGX rispettiva. Il modulo contiene un metodo che aggiunge le dichiarazioni di queste funzioni all'interno del testo, e riceve anche dal modulo che gestisce la conversione del file il flag di attestazione remota e un flag che indica se l'enclave deve essere attestata. In questo caso, il modulo importa all'interno della memoria protetta le librerie necessarie alla attestazione remota, aggiungendo dichiarazioni di import dentro il blocco relativo all'enclave. Vengono anche inseriti i prototipi delle funzioni di basso livello di attestazione del SDK Intel, usate dalle funzioni inserite dal framework SGX, che devono essere eseguiti dal codice protetto dell'enclave.

B.1.5 framaInterface

Il modulo è stato modificato nei metodi interni rispetto alla precedente versione del framework. La versione di Frama-C supportata precedentemente permetteva di eseguire sul sorgente sia una analisi di tipo sintattico sia una di tipo semantico. Il modulo era stato implementato considerando la presenza dei due tipi di analisi, ma nel corso degli aggiornamenti usciti successivamente, queste funzionalità sono state modificate ed ora è possibile soltanto fare una analisi di tipo semantico: i grafi di chiamate delle funzioni erano perciò generati in un verso opposto, invertendo l'identità della funzione chiamante e della funzione chiamata. Questo effetto non desiderato è ancora più grave nel caso di una applicazione convertita per SGX: una funzione ECall, secondo quel tipo di grafo generato, è quindi la funzione che chiama una funzione presente nell'ambiente di esecuzione non protetto, e questo è per definizione di SGX impossibile, se non tramite le definizioni delle OCall.

B.1.6 Constants

Il framework usa gli attributi di questo modulo per conservare gli scheletri di codice che saranno usati per generare i file di compilazione e le funzioni di SGX. Questi scheletri sono blocchi di codice di dimensioni molto grandi, non gestibili all'interno del codice del modulo principale del framework per motivi di leggibilità e comprensione, e si conservano usando il tipo di dato stringa R, fornito da Python. Le caratteristiche di questo tipo di stringhe sono sfruttate per poter generare codice che sia compatibile con ogni combinazione di funzionalità richieste per una applicazione. Vediamo ad esempio il caso di generazione del Makefile di compilazione, il cui contenuto è salvato nell'attributo `__makefile_project_RA___` nel caso in cui deve generare un Makefile per una applicazione che dovrà includere al suo interno le librerie di attestazione remota del client: la stringa ha al suo interno un template che ha già salvato le regole per i file oggetto delle librerie

di attestazione, e insieme a queste regole sono anche salvati dei valori placeholder `@replace_app_objects@`, `@replace_application_app@` e `@replace_App_name@`. Il framework chiama il metodo `writeMakefileRA` del modulo durante il processo di conversione, passando come argomento il numero di enclavi individuate durante il processo di analisi delle notazioni inserite dallo sviluppatore: per ogni occorrenza trovata, genererà rispettivamente una stringa contenente le regole di compilazione di tutti i file appartenenti alle enclavi richieste, e userà il metodo `replace` per sostituire il valore `@replace_app_objects@` con le regole di compilazione dei file di tutte le enclavi. Successivamente, farà la stessa cosa per il secondo valore placeholder che sarà sostituito dalle regole di compilazione dei file `enclave_manager_X.h` di tutte le enclavi, e per il terzo valore placeholder, che dovrà essere sostituito la regola di complicazione finale che unisce tutti i file oggetto definiti per la generazione dell'eseguibile finale.
L'uso dei template e della costruzione del codice finale è stata implementata con questo meccanismo anche per definire un Makefile che compili automaticamente tutte le enclavi dell'applicazione. Infatti, non è possibile per il framework sapere a priori quante regole dovranno essere inserite, e diventa necessario l'uso di un template completato automaticamente.
Nel modulo sono presenti altri metodi usati con la stessa logica:

writeXMLConf : genera il file `enclave.config.xml` per l'i-esima enclave, usando il contenuto dell'attributo `__enclave_config_xml_code` e sostituendo il parametro di Production ID di default con quello dell'i-esima enclave;

writeTestKey : usa l'attributo `__rsa_private_key__` per creare il file enclave_-private_key__ per ogni enclave;

writeLDS : genera il file di link enclave.lds per ogni enclave, usando il contenuto salvato in `___enclave_lds_code`;

writeEnclaveManager : usa i template salvati negli attributi interni del modulo `__enclave_manager_c_code__` e `__enclave_manager_h_includes__` per costruire un file con i metodi specifici di ogni enclave. Per questo motivo infatti riceve come argomento l'iteratore dell'enclave i-esima che il framework genera nel processo principale.

Anche l'attestazione remota usa Constants per memorizzare il contenuto del testo che dovrà essere inserito nel codice dell'implementazione: per il processo di compilazione delle funzionalità, in cui è necessario distinguere per ogni id dell'enclave possibile quale è l'enclave che va attestata, sono stati definiti il primo attributo `___remoteAttestationFunctions_Header__`, in cui una stringa R contiene la sezione di codice relativa alle librerie che devono essere importate per il funzionamento delle funzioni di attestazione remota (come `sgx_spinlock.h` fornita dal SDK di SGX installato nella Sezione A.1.2) , e il secondo attributo `___remoteAttestationFunctions_Implementations___`, per la sezione di codice di implementazione delle funzioni di attestazione che saranno contenute nel file "enclave.c".

B.1.7 remoteAttestationInterface

Come per il modulo Constants, si usano gli attributi per salvare i blocchi di codice che contengono le implementazioni delle funzioni: in questo caso, non sono salvate le funzioni di attestazione remota, dato che sono appartenenti alle librerie `client_starter` e `server_interface`, ma le chiamate intermedie che importano le librerie e chiamano le funzioni che iniziano il processo.
Si nota che il processo di attestazione si basa sulle funzioni e sulle classi definite da Intel sul suo progetto pubblico[2], e che si cercherà di spiegare solo i punti in cui è più utile implementare modifiche future.

Implementazione client

Il metodo `generateClientCode` viene richiesto dal processo principale del modulo `codeAnalyser` per creare, rispettando la struttura di un oggetto Function, una copia della funzione che contiene la notazione di inserimento di attestazione remota. La notazione viene sostituita dal seguente blocco di codice:

```
sgx_enclave_id_t eidX = create_enclave_X();
attest_enclave(eidX);
```

`create_enclave_X` è incluso nella libreria `enclave_manager_X.h` costruita dal processo principale del framework per ogni enclave, e restituisce come risultato l'id dell'enclave creata nella memoria che serve per identificarla all'interno del codice. La funzione `attest_enclave`, definita nella libreria `enclave_manager_common.h`, è una funzione wrapper che gestisce i parametri sufficienti per rilevare un errore nel processo e indicarlo allo sviluppatore, e chiama la funzione `perform_Attestation` (il cui codice di implementazione è mostrato nella Figura B.2 per iniziare il processo di attestazione. La nuova versione della funzione costruita, una volta terminata

```
int resultAttestation = -1;
printf("Attestation starting, connecting with the server.");
resultAttestation = perform_Attestation(eid);
if (resultAttestation != 0)
{
    printf("Error in attestation, aborting program.");
    abort();
}
printf("Attestation succeded, the enclave is valid. Starting the
   application...\n");
```

Figura B.2: Codice implementativo della funzione wrapper per iniziare il processo di attestazione

[2] https://github.com/intel/sgx-ra-sample

la sua costruzione, sostituisce la vecchia versione memorizzata all'interno del dizionario ottenuto con l'analisi del sorgente. In questo modo, in fase di scrittura del file dell'applicazione convertita, non verrà scritta la versione precedente della funzione che conteneva esclusivamente la notazione per richiedere l'attestazione, ma la versione pronta per chiamare le funzioni del processo di attestazione. In `perform_Attestation`, si inizia il vero processo di attestazione su una nuova porta. La comunicazione tra client e server sarà gestita da un socket, contenuto in una classe `MsgIO`.

Implementazione server

Il modulo contiene metodi per inserire i dati utili alla connessione dell'applicazione leggendo il file di configurazione alle librerie (`configSPSettings` per i dati ottenuti con la registrazione all'IAS). `updateMakefileServerProvider` serve per aggiornare il Makefile con le librerie usate nel processo, e il funzionamento è simile a quello usato per l'inserimento delle regole di compilazione dei file dell'enclave: vengono sostituiti alcuni valori placeholder con stringhe R che contengono le regole che devono essere inserite per la corretta compilazione finale, rispettando la formattazione dello script grazie alle proprietà delle stringhe R.
In maniera analoga, viene anche sostituita la funzione in cui è inserita la notazione di attestazione del server: le funzioni `insertFunctionOfClientAttestation` e `generateServerCode` sono usate rispettivamente per creare la funzione `requestClientAttestation` e per inserire all'interno della funzione che contiene l'attestazione il codice per chiamarla. La funzione `requestClientAttestation` serve a chiamare dalla libreria di attestazione del server `server_interface` la funzione `startServerIAS()`: questa funzione carica le informazioni ottenute (chiavi di sottoscrizione, certificato, misure della whitelist adottata e altre) nella memoria dell'applicazione e inizia l'ascolto sulla porta selezionata di richieste di connessione da parte di client che vogliono attestarsi.

B.1.8 codeSplittingInterface

Il processo di divisione del codice dell'applicazione su un codice eseguito remotamente è gestito dal processo principale di conversione e da questo modulo. Oltre ad usare metodi che effettuano operazioni di verifica sulle condizioni di entrocontenibilità che ogni funzione da trasferire deve rispettare, sono definiti anche i metodi che analizzano i parametri usati dalla funzione, e generano codice fondamentale per la serializzazione dei dati, generando per ogni tipo di dato relativo ricevuto in input all'argomento della funzione trasferita l'opportuno output, come lo specificatore di formato da usare (ad esempio, se deve serializzare un intero, restituirà un `"%d"`).
Il modulo contiene in un attributo il template del codice del server di esecuzione remota, salvato sempre sotto forma di stringa R dal nome `__serverCSEntireCode__`.
Il server, per ogni richiesta di connessione ricevuta dal client, durante un ciclo che itera un numero infinito di volte alloca un nuovo thread `pthread_routine`, incaricato di occuparsi della ricezione dei parametri che vengono inviati in byte e ricomposti. Quando termina la ricezione dei parametri, il thread sceglie in base

all'id ricevuto la funzione selezionata, la esegue e restituisce il risultato riconvertendolo in byte, terminando.
All'interno di questo template, sono presenti due placeholder `[CODE_FUNCTIONS]` e `[SWITCH_CASE_FUNCTIONS]`: il processo di conversione sostituirà quei valori con le funzioni da trasferire secondo le richieste ricevute e il costrutto switch per riconoscere la funzione da invocare in base all'id ricevuto dal client. Nel codice implementativo, troviamo la definizione della struttura dati `param_t` scelta per rappresentare i parametri serializzati, come mostrato in Figura B.3. Con questa struttura, si rap-

```
typedef struct{
char type;
int value;
float value_f;
char* buf;
double value_d;
unsigned int value_u;
} param_t;
```

Figura B.3: Struttura dati di un parametro usata nella divisione del codice

presenta il tipo di dato che verrà ricevuto dal server e l'informazione che il dato contiene. La variabile `type` serve per indicare il tipo di dato con un singolo carattere, e per ognuno di questi dati è definita una variabile che memorizzerà il valore ricevuto dal client:

- "u" per i dati interi senza segno salvati nella variabile `value_u`;
- "i" per i dati interi salvati nella variabile `value`;
- "f" per i dati di tipo float salvati nella variabile `value_f`;
- "c" per i dati composti da più caratteri consecutivi salvati nello spazio di memoria indicato dal puntatore `buf`;
- "d" per i dati di tipo double salvati nella variabile `value_d`.

Il resto dei dati non è supportato dalla funzionalità, ma può essere aggiunto in futuro.
Il primo dato che viene inviato al server è l'id della funzione, associato dal processo principale in fase di analisi. Questo dato è un valore intero, rappresentato quindi su 4 Byte: il server legge i primi quattro Byte con la funzione di sistema `read`, e deserializza i dati ricevuti usando la funzione di sistema `atoi` per convertire in intero il carattere ricevuto. A questo punto, crea un vettore di parametri di dimensione "MAX_PARAMETERS", ottenuto analizzando tutte le funzioni e cercando il valore di argomenti di una funzione il più grande possibile, e aspetta la ricezione di ogni parametro. I parametri saranno inviati nel formato "Type:X Value:Y" dal client: si eseguono una serie di operazioni di manipolazione della stringhe con i token con la funzione standard `strtok`. In questo modo, si riesce a sapere in quale

variabile della struttura dati, rappresentata in Figura B.3, si deve memorizzare il valore ricevuto.

Bibliografia

[1] N. I. of Standards and Technology, "Minimum Security Requirements for Federal Information and Information Systems." FIPS PUB 200, Marzo 2006, DOI 10.6028/NIST.FIPS.200

[2] V.C.Hu, D.Ferraiolo, and D.Kuhn, "Assessment of Access Control Systems." NISTIR 7316, Settembre 2006, DOI https://doi.org/10.6028/NIST.IR.7316

[3] M.Sabt, M.Achemlal, and A.Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not", 2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki (Finlandia), 20-22 Agosto 2015, pp. 57–64, DOI 10.1109/Trustcom.2015.357

[4] R.Ross, V.Pillitteri, G.Guissanie, R.Wagner, R.Graubart, and D.Bodeau, "Enhanced Security Requirements for Protecting Controlled Unclassified Information: A Supplement to NIST Special Publication 800-171." NIST SP 800-172, Febbraio 2021, DOI https://doi.org/10.6028/NIST.SP.800-172

[5] "ISO/IEC/IEEE International Standard - Systems and software engineering–Measurement process", ISO/IEC/IEEE 15939:2017(E), Aprile 2017, pp. 1–49, DOI 10.1109/IEEESTD.2017.7907158

[6] D. E. 3rd and P.Jones, "US Secure Hash Algorithm 1 (SHA1)." RFC-3174, Settembre 2001, DOI 10.17487/RFC3174

[7] S.Berger, R.Cáceres, K.Goldman, R.Perez, R.Sailer, and L. Doorn, "VTPM: Virtualizing the trusted platform module", 15th USENIX Security Symposium (USENIX Security 06), Vancouver (Canada), 31 Luglio - 4 Agosto 2006, pp. 1–16. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.2124&rep=rep1&type=pdf

[8] V.Costan and S.Devadas, "Intel SGX Explained", IACR Cryptology ePrint Archive, vol. 2016, Gennaio 2016, pp. 1–86. http://eprint.iacr.org/2016/086

[9] J.Reid and W.Caelli, "DRM, Trusted Computing and Operating System Architecture", Proceedings of the 2005 Australasian Workshop on Grid Computing and E-Research, vol. 44, Gennaio 2005, pp. 127—-136. ISBN:1920682260,https://web2.qatar.cmu.edu/cs/15349/dl/DRM-TC.pdf

[10] M.Dworkin, E.Barker, J.Nechvatal, J.Foti, L.Bassham, E. Roback, and J.Dray, "Advanced Encryption Standard (AES)", Ottobre 2001, DOI https://doi.org/10.6028/NIST.FIPS.197

[11] L.Chen, "Recommendation for Key Derivation Using Pseudorandom Functions." NIST SP 800-108, Ottobre 2009, DOI 10.6028/NIST.SP.800-108

[12] S. Fei, Z.Yan, W.Ding, and H.Xie, "Security Vulnerabilities of SGX and Countermeasures: A Survey", ACM Computing Surveys, vol. 54, Luglio 2021, pp. 1–36, DOI 10.1145/3456631

[13] A.Nilsson, P. N. Bideh, and J. Brorsson, "A Survey of Published Attacks on Intel SGX", CoRR, vol. abs/2006.13598, Giugno 2020. https://arxiv.org/abs/2006.13598

[14] H.Shacham, "The Geometry of Innocent Flesh on the Bone: Return-into-Libc without Function Calls (on the X86)", Proceedings of the 14th ACM Conference on Computer and Communications Security, New York (NY, USA), Ottobre 2007, pp. 552—561, DOI 10.1145/1315245.1315313

[15] J. Lee, J.Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang, "Hacking in darkness: Return-oriented programming against secure enclaves", 26th USENIX Security Symposium (USENIX Security 17), Vancouver (Canada), 16-18 Agosto 2017, pp. 523–539. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-jaehyuk

[16] M.Schwarz, S.Weiser, and D.Gruss, "Practical Enclave Malware with Intel SGX", Detection of Intrusions and Malware, and Vulnerability Assessment, Giugno 2019, pp. 177–196, DOI 10.1007/978-3-030-22038-9_9

[17] J.Wang, Y.Cheng, Q.Li, and Y.Jiang, "Interface-Based Side Channel Attack Against Intel SGX", ArXiv, vol. abs/1811.05378, Ottobre 2018, pp. 1–13, DOI 10.48550/ARXIV.1811.05378

[18] Y.Jang, J.Lee, S.Lee, and T.Kim, "SGX-Bomb: Locking Down the Processor via Rowhammer Attack", SysTEX'17: Proceedings of the 2nd Workshop on System Software for Trusted Execution, New York (NY, USA), 28 Ottobre 2017, DOI 10.1145/3152701.3152709

[19] F.Koeune and F.X.Standaert, "A Tutorial on Physical Security and Side-Channel Attacks", Foundations of Security Analysis and Design III : FOSAD 2004/2005, Bertinoro (Italia), 19-24 Settembre 2005, pp. 78–108, DOI 10.1007/11554578_3

[20] Y.Xu, W.Cui, and M.Peinado, "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems", 2015 IEEE Symposium on Security and Privacy, San Jose (CA, USA), 17-21 Maggio 2015, pp. 640–656, DOI 10.1109/SP.2015.45

[21] J. Bulck, F.Piessens, and R.Strackx, "SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control", SysTEX'17: Proceedings of the 2nd Workshop on System Software for Trusted Execution, New York (NY, USA), 28 Ottobre 2017, pp. 1–6, DOI 10.1145/3152701.3152706

[22] D.A.Osvik, A.Shamir, and E.Tromer, "Cache Attacks and Countermeasures: The Case of AES", Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology, San Jose (CA, USA), 13-17 Febbraio 2005, pp. 1—-20, DOI 10.1007/11605805_1

[23] D.Gullasch, E.Bangerter, and S.Krenn, "Cache Games – Bringing Access-Based Cache Attacks on AES to Practice", 2011 IEEE Symposium on Security and Privacy, Oakland (CA, USA), 22-25 Maggio 2011, pp. 490–505, DOI 10.1109/SP.2011.22

[24] T.Kim and Y.Shin, "Reinforcing Meltdown Attack by Using a Return Stack Buffer", IEEE Access, vol. 7, Dicembre 2019, pp. 186065–186077, DOI 10.1109/ACCESS.2019.2961158

[25] A.Moghimi, G.Irazoqui, and T.Eisenbarth, "CacheZoom: How SGX Amplifies The Power of Cache Attacks", Cryptographic Hardware and Embedded Systems (CHES '17), Taipei (Taiwan), 25-28 Settembre 2017, pp. 69–90, DOI 10.1007/978-3-319-66787-4_46

[26] F.Dall, G. Micheli, T.Eisenbarth, D.Genkin, N.Heninger, A.Moghimi, and Y.Yarom, "CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks", IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2018, Maggio 2018, pp. 171—191, DOI 10.13154/tches.v2018.i2.171-191

[27] F.Brasser, U.Müller, A.Dmitrienko, K.Kostiainen, S.Capkun, and A.R.Sadeghi, "Software Grand Exposure: SGX Cache Attacks Are Practical", WOOT'17: Proceedings of the 11th USENIX Conference on Offensive Technologies, Vancouver (Canada), 14-15 Agosto 2017, pp. 1–12. http://arxiv.org/abs/1702.07521

[28] J.Götzfried, M.Eckert, S. Schinzel, and T.Müller, "Cache Attacks on Intel SGX", EuroSec'17:Proceedings of the 10th European Workshop on Systems Security, New York (NY, USA), 23-26 Aprile 2017, pp. 1–6, DOI 10.1145/3065913.3065915

[29] S. Schaik, M.Minkin, A.Kwong, D.Genkin, and Y.Yarom, "CacheOut: Leaking Data on Intel CPUs via Cache Evictions", 2021 IEEE Symposium on Security and Privacy (SP), San Francisco (CA, USA), 24-27 Maggio 2021, pp. 339–354, DOI 10.1109/SP40001.2021.00064

[30] O.Aciiçmez, Ç.K.Koç, and J.P.Seifert, "Predicting Secret Keys via Branch Prediction", CT-RSA'07: Proceedings of the 7th Cryptographers' track at the RSA conference on Topics in Cryptology, San Francisco (CA, USA), 5-9 Febbraio 2007, pp. 225–242, DOI 10.1007/11967668_15

[31] D. Evtyushkin, R. Riley, N.Abu-Ghazaleh, D. Ponomarev, and ECE, "BranchScope: A New Side-Channel Attack on Directional Branch Predictor", ACM SIGPLAN Notices, vol. 53, Febbraio 2018, pp. 693—-707, DOI 10.1145/3173162.3173204

[32] T.Huo, X.Meng, W.Wang, C.Hao, P.Zhao, J.Zhai, and M.Li, "Bluethunder: A 2-level Directional Predictor Based Side-Channel Attack against SGX", IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2020, Novembre 2019, pp. 321—347, DOI 10.13154/tches.v2020.i1.321-347

[33] Q. Dang, "Secure Hash Standard (SHS)", Marzo 2012, DOI 10.6028/NIST.FIPS.180-4

[34] M.Stipčević and C.Koç, "True Random Number Generators", pp. 275–315. Springer International Publishing, Novembre 2014. DOI:10.1007/978-3-319-10683-0_12

[35] A.Lee, M.Smid, and S.Snouffer, "Security Requirements for Cryptographic Modules [includes Change Notices as of 12/3/2002]." FIPS 140-2, Maggio 2001, DOI 10.6028/NIST.FIPS.140-2

[36] B.Beurdouche, K.Bhargavan, A.Delignat-Lavaud, C.Fournet, M.Kohlweiss, A.Pironti, P.Y.Strub, and J.K.Zinzindohoue, "A Messy State of the Union: Taming the Composite State Machines of TLS", Communication from ACM, vol. 60, Febbraio 2017, pp. 99—107, DOI 10.1145/3023357

[37] T.Polk and S.Turner, "Prohibiting Secure Sockets Layer (SSL) Version 2.0." RFC 6176, Marzo 2011, DOI 10.17487/RFC6176

[38] E.Barker, "Guideline for using cryptographic standards in the federal government: Cryptographic mechanisms", Agosto 2016, DOI https://doi.org/10.6028/NIST.SP.800-175B

[39] M.McFadden, "Request to Move RFC 2754 to Historic Status." RFC 6254, Maggio 2011, DOI 10.17487/RFC6254

[40] P.Kyzivat, v. Gurbani, H. Schulzrinne, and J. Rosenberg, "RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF)." RFC-4480, Luglio 2006, DOI 10.17487/RFC4480

[41] E.Barker, L.Chen, A.Roginsky, A.Vassilev, and R.Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography", Aprile 2018, DOI 10.6028/NIST.SP.800-56Ar3

[42] J.Jonsson and B.Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1." RFC 3447, Febbraio 2003, DOI 10.17487/RFC3447

[43] S.Gueron, "Quick Verification of RSA Signatures", Proceedings of the 2011 Eighth International Conference on Information Technology: New Generations, Las Vegas (NV, USA), 11-13 Aprile 2011, pp. 382—-386, DOI 10.1109/ITNG.2011.74

[44] E.Brickell, J.Camenisch, and L.Chen, "Direct Anonymous Attestation", Cryptology ePrint Archive, Paper 2004/205, Ottobre 2004, pp. 132–144, DOI 10.1145/1030083.1030103

[45] H.Krawczyk, "SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols", Advances in Cryptology - CRYPTO 2003, Santa Barbara (CA, USA), 17-21 Agosto 2003, pp. 400–425, DOI 10.1007/978-3-540-45146-4_24

[46] G.Chen, S.Chen, Y.Xiao, Y.Zhang, Z.Lin, and T.H.Lai, "SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution", 2019 IEEE European Symposium on Security and Privacy (EuroS&P), Stoccolma (Svezia), 17-19 Giugno 2019, pp. 142–157, DOI 10.1109/EuroSP.2019.00020

[47] M.Ceccato and M. Dalla Preda and J.Nagra and C.Collberg and P.Tonella, "Trading-off security and performance in barrier slicing for remote software entrusting", Automated Software Engineering, vol. 16, Giugno 2009, pp. 235–261, DOI 10.1007/s10515-009-0047-y

[48] X. Zhang and Gupta, R., "Hiding program slices for software security", International Symposium on Code Generation and Optimization (CGO) 2003., San Francisco (CA, USA), 23-26 Marzo 2003, pp. 325–336, DOI 10.1109/CGO.2003.1191556

[49] Y. Swami, "Intel SGX Remote Attestation is not sufficient", 26-27 Luglio, 2017, BlackHat USA 2017, Las Vegas (NV, USA), https://eprint.iacr.org/2017/736.pdf

[50] M.Bellare and O.Goldreich, "On Defining Proofs of Knowledge", Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara (CA, USA), Agosto 16-20, 1992, pp. 390—420, DOI 10.1007/3-540-48071-4_28

[51] M.Conti, N.Dragoni, and V.Lesyk, "A Survey of Man In The Middle Attacks", IEEE Communication Surveys and Tutorials, vol. 18, Luglio 2016, pp. 2027––2051, DOI 10.1109/COMST.2016.2548426

[52] J.Ménétrey, C.Göttel, M.Pasin, P.Felber, and V.Schiavoni, "An Exploratory Study of Attestation Mechanisms for Trusted Execution Environments", SysTEX'22: the 5th Workshop on System Software for Trusted Execution, Aprile 2022, pp. 1–7, DOI 10.48550/ARXIV.2204.06790

[53] G.Chen and Y.Zhang, "MAGE: Mutual Attestation for a Group of Enclaves without Trusted Third Parties", arXiv, Agosto 2020, pp. 1–15, DOI 10.48550/ARXIV.2008.09501

[54] G.Chen, Y.Zhang, and T.Lai, "OPERA: Open Remote Attestation for Intel's Secure Enclaves", CCS '19: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, Londra (Regno Unito), 11-15 Novembre 2019, pp. 2317—2331, DOI 10.1145/3319535.3354220

[55] R. Yeluri and E.Castro-Leon, "Building the Infrastructure for Cloud Security", Apress Berkeley (CA, USA), 2014, ISBN: 978-1-4302-6145-2. 10.1007/978-1-4302-6146-9

[56] N.Schear, P.T.Cable, T.M.Moyer, B.Richard, and R.Rudd, "Bootstrapping and Maintaining Trust in the Cloud", Proceedings of the 32nd Annual Conference on Computer Security Applications, Los Angeles (CA, USA), 5-8 Dicembre 2016, pp. 65—77, DOI 10.1145/2991079.2991104

[57] M.Costa, "Protezione automatica del codice tramite enclavi SGX = Automatic code protection via SGX enclaves." pp.1–150, http://webthesis.biblio.polito.it/15956/, Ottobre 2020

[58] ISO, "ISO/IEC 9899:2011 Information technology — Programming languages — C", International Organization for Standardization, Dicembre 2011

[59] ISO, "ISO/IEC 9899:2018 Information technology — Programming languages — C", International Organization for Standardization, Giugno 2018

[60] S.Josefsson and S.Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures." RFC 7468, Aprile 2015, DOI 10.17487/RFC7468

[61] J.Lind, C.Priebe, D.Muthukumaran, D. O'Keeffe, P. Aublin, F. Kelbert, T.Reiher, D.Goltzsche, D.Eyers, R.Kapitza, C.Fetzer, and P.Pietzuch, "Glamdring: Automatic Application Partitioning for Intel SGX", 2017 USENIX Annual Technical Conference (USENIX ATC 17), Santa Clara (Canada), 12 - 14 Luglio 2017, pp. 285–298. https://www.usenix.org/conference/atc17/technical-sessions/presentation/lind

[62] E.Barker and W.C.Barker, "Recommendation for Key Management: Part 2 – Best Practices for Key Management Organizations." NIST SP 800-57, Maggio 2019, DOI 10.6028/NIST.SP.800-57pt2r1

[63] S.Boeyen, S.Santesson, T.Polk, R.Housley, S.Farrell, and D.Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." RFC 5280, Maggio 2008, DOI 10.17487/RFC5280

[64] E.Rescorla and T.Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2." RFC 5246, Agosto 2008, DOI 10.17487/RFC5246

[65] Y.Shen, H.Tian, Y.Chen, K.Chen, R.Wang, Y.Xu, Y.Xia, and S.Yan, "Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX", Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, New York (NY, USA), Marzo 2020, pp. 955—-970, DOI 10.1145/3373376.3378469

[66] G.Tan, "Principles and Implementation Techniques of Software-Based Fault Isolation", Foundations and Trends® in Privacy and Security, vol. 1, no. 3, 2017, pp. 137–198, DOI 10.1561/3300000013

[67] H.Birkholz, D.Thaler, M.Richardson, N.Smith, and W.Pan, "Remote Attestation Procedures Architecture", Internet-Draft draft-ietf-rats-architecture-22, Internet Engineering Task Force, Settembre 2022. Work in Progress

[68] J.Cui, J.Z.Yu, S.Shinde, P.Saxena, and Z.Cai, "SmashEx: Smashing SGX Enclaves Using Exceptions", Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, Novembre 2021, pp. 1–15, DOI 10.1145/3460120.3484821

[69] L.Regano, "An expert system for automatic software protection", Luglio 2019, DOI 11583/2751495

[70] A.Viticchié, C.Basile, A.Avancini, M.Ceccato, B.Kessler, B.Abrath, and B.Coppens, "Reactive Attestation: Automatic Detection and Reaction to Software Tampering Attacks", Proceedings of the 2016 ACM Workshop on Software PROtection, Vienna (Austria), 24-28 Ottobre 2016, pp. 73—84, DOI 10.1145/2995306.2995315

[71] M.-J. O. Saarinen and J.-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)." RFC 7693, Novembre 2015, DOI 10.17487/RFC7693

[72] R. Rivest, "The MD5 Message-Digest Algorithm." RFC 1321, Aprile 1992, DOI 10.17487/RFC1321

[73] Q.Dang, "The Keyed-Hash Message Authentication Code (HMAC)", Luglio 2008, DOI https://doi.org/10.6028/NIST.FIPS.198-1