POLITECNICO DI TORINO

Corso di Laurea Magistrale in
Ingegneria Informatica

Tesi di Laurea Magistrale

Studio e realizzazione di un'architettura software sicura per applicazioni IoT

Relatore
prof. Gianpiero Cabodi

Candidato
Antonino Rocca

A.A. 2017-2018

Indice

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi anni si è affermato un nuovo neologismo, Internet of Things (IoT). Internet non è più solo legato al mondo digitale, ai calcolatori o alle reti ma viene esteso anche alle persone e agli oggetti. Infatti diventano sempre più comuni gli oggetti connessi ad internet in grado di svolgere diversi compiti diventando *smart*.

Nel 2015 i sistemi IoT hanno superato 15 miliardi di unità con una previsione di 30 miliardi di dispositivi nel 2020 [1].
Tutti i prodotti IoT catturano dati che vengono elaborati e restituiti sotto forma di informazioni utili. Le informazioni sono utilizzate per erogare diversi servizi all'utente [2]. Per esempio, un dispositivo per il fitness colleziona dati che vengono elaborati e restituiti all'utente sotto forma di informazioni su battito cardiaco, frequenza respiratoria o altro. I device IoT possono essere trovati in numerose applicazioni tra cui Smart Health, Smart City, Smart Building, Smart Home, Smart Manufactoring.

A causa dell'aggressivo time to market e dell'alta complessità di progettazione di questi dispositivi è molto facile introdurre errori che si traducono in vulnerabilità utilizzabili per condurre un attacco sul dispositivo o sulla rete alla quale il device è connesso. [3]
Gli attacchi informatici oggi sono uno tra i punti più critici del mondo connesso. Nel 2017 si sono contati più di mille casi di questi attacchi , ognuno dei quali ha sottratto informazioni alle compagnie danneggiando, oltre le stesse società, molti utenti. Questo è costato alle compagnie attaccate diverse centinaia di migliaia di

dollari, danneggiando l'immagine e la credibilità dell'azienda [4]. I bersagli maggiormente attaccati sono stati proprio i device IoT. Perdita di informazioni, accesso non autorizzato a oggetti o dati, perdita del controllo del device sono solo alcune delle problematiche che preoccupano queste nuove tecnologie. In alcuni casi può essere messa a rischio la sicurezza fisica dell'utente. Un caso eclatante è quello della Jeep Cherokee: hackers etici hanno dimostrato che è possibile prendere il controllo dell'automobile da remoto, creando dei reali pericoli sia per il conducente che per i passeggeri. [5] Al giorno d'oggi esistono alcune tecnologie e metodologie di progettazione che aumentano la sicurezza di un device IoT.

Attraverso lo sviluppo di questa tesi si è voluto contribuire alla realizzazione di un'architettura software con funzionalità di sicurezza per sistemi IoT, in modo tale da poter essere utilizzata come base per l'implementazione di applicazioni connesse aumentandone la sicurezza e la resilienza agli attacchi. In particolare, il progetto si è incentrata sullo sviluppo di funzionalità per la protezione del firmware e della memoria al fine di evitare la manomissione o la sottrazione di dati sensibili. Le funzionalità sviluppate che integrano la protezione del firmware e della memoria sono: **Secure Boot** e **Authenticated Firmware Update**. Il secure boot si occupa di accertarsi che il firmware da eseguire non sia stato manomesso. L'authenticated firmware update è utilizzato per l'aggiornamento del firmware da remoto garantendone l'integrità e la confidenzialità attraverso l'utilizzo del protocollo SSL/TLS con autenticazione mutua. Nel seguente elaborato le parole secure e authenticated saranno utilizzate come sinonimi.
Per comprendere meglio i motivi che portano allo sviluppo di questa soluzione è indispensabile un'introduzione approfondita al mondo IoT e ai rischi a cui i dispositivi sono esposti. Dopo quest'analisi viene presentata una soluzione che protegge da un sottoinsieme di minacce appartenenti ai dispositivi IoT. L'elaborato è sviluppato come segue:

- Background: in questo capitolo si analizzano i problemi dei sistemi IoT. Si descrivono nello specifico l'architettura e i problemi che nascono dalla progettazione del sistema e dall'architettura di questi dispositivi. In seguito, vengono forniti concetti tecnici di sicurezza informatica necessari per sviluppare una soluzione;

- Analisi dei requisiti: dopo aver descritto il problema vengono selezionati un sottoinsieme di vulnerabilità. Sulla base di queste vulnerabilità viene presentata una soluzione per la loro risoluzione. La soluzione presentata si preoccupa soprattutto di garantire dell'integrità del firmware, installato o da installare, attraverso le funzionalità di secure boot e secure firmware update.

- Implementazione: in questo capitolo si analizzano le scelte progettuali che sono state effettuate. Sono descritti nel dettaglio il sistema operativo utilizzato, il dispositivo su cui implementare la soluzione e le funzionalità crittografiche da integrare. Una volta scelta la configurazione più adatta per il problema da affrontare, si descrivono le implementazioni delle funzionalità di secure boot e di secure firmware update e i tools sviluppati per l'automatizzazione e il supporto dei processi interconnessi con le funzionalità sviluppate;

- Conclusioni: nella parte finale si discutono i risultati ottenuti, le criticità affrontate e i miglioramenti che si possono apportare alla soluzione sviluppata. Si descrivono, inoltre, come è possibile testare e validare la soluzione implementata.

Capitolo 2

Background

Dato che l'impatto che questi sistemi hanno nella vita quotidiana è elevato, è necessaria un'analisi approfondita su come sia formato un sistema IoT e da quali minacce deve essere protetto affinché si possa avere una visione d'insieme sulle vulnerabilità di questa tipologia di dispositivi. Questo capitolo spiegherà, inoltre, come è possibile utilizzare queste minacce per attaccare i sistemi connessi.

Le vulnerabilità presenti su questa tipologia di dispositivi sono molteplici, per questa ragione è necessario selezionare un sottoinsieme di minacce in modo tale da poter sviluppare una soluzione efficace in grado di proteggere i dispositivi dalle maggiori vulnerabilità [6]. Una volta effettuata questa analisi, verranno descritti dei concetti di sicurezza informatica che sono stati utilizzati per caratterizzare la soluzione proposta.

2.1 Architettura dei dispositivi IoT

L'IoT è considerato l'estensione di Internet che agisce nel mondo fisico attraverso dispositivi connessi. Entità, device, servizi sono concetti chiave nel dominio IoT [7]. Attraverso la definizione dei concetti citati si vuole definire meglio quali sono i punti in comune delle tecnologie IoT e quali difficoltà devono essere affrontate in questo tipo di architettura.

Con entità ci si riferisce a persone o oggetti che hanno la necessità di interagire con Internet. Il dispositivo hardware utilizzato per la connessione alla rete è chiamato device. Questo dispositivo integra sensori e attuatori in grado di collezionare informazioni da utilizzare per l'interazione con Internet. La comunicazione Machine-to-Machine (M2M) è l'applicazione più popolare in ambito IoT ed è largamente usata in tutti gli ambiti in cui l'IoT viene impiegato [8].
Dietro i benefici delle applicazioni IoT, esistono molteplici minacce alla sicurezza dell'architettura e quindi dei dispositivi:

- Dato che molti di questi dispositivi agiscono senza la supervisione umana, averne l'accesso fisico è molto semplice. Con un accesso fisico facilitato è possibile manomettere fisicamente il normale funzionamento del device;
- Utilizzando tecnologie wireless, è facile intercettare informazioni provenienti da questi dispositivi;
- Avendo risorse limitate, i device IoT sono in grado di effettuare operazioni semplici e quindi non vengono quasi mai impiegati meccanismi di protezione.

Un sistema IoT è formato da diversi componenti. Le sottosezioni seguenti illustreranno brevemente alcuni dettagli al fine di chiarire i motivi che portano ad attenzionare alcuni di questi componenti.

IoT device

I dispositivi IoT permettono, a chi ne fa uso, l'interazione con il mondo digitale grazie alla loro connettività.Le tecnologie di interconnessione utilizzate su questi dispositivi sono le seguenti: RFID (Radio Frequency IDentification), NFC (Near Field Communication), Reti cellulari, Bluetooth e tecnologie wireless per la comunicazione tra device e service [2].
I dispositivi IoT si distinguono inoltre per le loro risorse limitate. A causa di quest'ultima caratteristica, i device IoT sono molto vulnerabili ad attacchi informatici poiché si cerca di non implementare funzionalità strettamente utili al tipo di servizio offerto.

Date le limitate prestazioni, è necessario implementare soluzioni di sicurezza a basso impatto sul sistema in termini di memoria e calcoli computazionali.

IoT Service

I service facilitano l'interazione dei dispositivi con il mondo digitale. Essi utilizzano i device IoT per l'interazione fisica cosi da poter ricevere informazioni utili da elaborare. Esistono degli standard nel mondo dell'information technology creati per l'implementazione e uso di questi servizi [9].

Sicurezza nelle architetture IoT

In questa sezione verranno analizzate le tre maggiori problematiche di sicurezza in un'architettura IoT: segretezza, fiducia, autenticazione. Dato che in queste applicazioni l'accesso ai dati non è solo consentito all'utente, ma anche ad oggetti autorizzati, si evidenziano dei problemi dovuti a: gestione degli accessi controllati, meccanismi di autorizzazione e autenticazione di oggetti o utenti che necessitano l'accesso alle informazioni.
Il dispositivo IoT deve accertarsi che la parte interessata abbia l'autorizzazione ad accedere ai dati richiesti prima di rendere disponibili le informazioni.L'accesso autorizzato e autenticato è necessario per stabilire una comunicazione sicura tra le parti. Il maggior problema nasce nel gestire questi aspetti nel modo più semplice possibile.

Dal punto di vista manutentivo, è necessario creare regole d'accesso semplici, comprensibili e controllabili per l'autorizzazione e la gestione delle identità.

Dal punto di vista dell'autenticazione, è necessario avere una procedura che usufruisca di poche risorse computazionali, in modo tale da poter essere facilmente utilizzato e implementato su questi device.

Il mondo IoT è basato principalmente sullo scambio di informazioni tra dispositivi. È necessario che nessuno, se non gli oggetti autorizzati, possano leggere queste informazioni. La privacy è stata già pensata, progettata e implementata per macchine con più risorse, ma non esistono soluzioni efficienti che possano essere utilizzate con i device IoT.

2.2 Minacce, Vulnerabilità e Attacchi

Le vulnerabilità si distinguono in :

- Hardware: molto difficili da scoprire e risolvere;
- Software: appartenenti al sistema operativo, all'applicazione o ai driver delle periferiche.

Un esempio di vulnerabilità software è l'esposizione di dati, sensibili o no, da parte del sistema. Questi dati possono fornire informazione all'attaccante sulle debolezze del sistema. Ad esempio se il sistema espone la versione del sistema operativo utilizzato, è possibile utilizzare i bug di quella versione, se sono presenti, per averne accesso.
Una vulnerabilità può rappresentare una minaccia se viene sfruttata per eseguire un attacco. Con l'avvento dell'IoT esistono minacce diverse alle architetture. In quanto sistemi limitati, spesso i device IoT sono carenti di meccanismi di autenticazione, politiche di autorizzazione e cifratura dati. Questo rende l'architettura del servizio, back-end, insicura.
Esistono molteplici attacchi che sfruttano le limitate risorse e la mancanza di controlli di un dispositivo e si distinguono in attacchi fisici e logici.

Gli attacchi fisici spesso sono basati sulla manomissione del dispositivo. Ad esempio:

- Memory dumping e modifica del firmware. Ovvero riuscire a scaricare il firmware installato sul dispositivo, tradurlo, modificarlo e poi collocare il software modificato sul dispositivo;
- Code injection. Tramite l'accesso alle periferiche si riesce a inserire del codice arbitrario per poi farlo eseguire;
- Utilizzare la porta di debug per la manomissione dei dati, delle periferiche a livello logico.

Tra gli attacchi di tipo logico sono critici quelli che utilizzano la connessione come punto d'accesso e, potenzialmente, potrebbero dare gli stessi risultati di quelli fisici:

- Buffer overflow: utilizzare il buffer che contiene i pacchetti dati ricevuti per immettere codice o sovrascrive informazioni nella memoria del dispositivo;
- Packet sniffing: riuscire a catturare dalla rete pacchetti in transito. E' possibile, se non si utilizza nessun metodo di cifratura, catturare informazioni sensibili;

- Buffer over-read: riuscire a leggere la memoria a causa della cattiva gestione di accesso ad essa. Una vulnerabilità di questo tipo può dare accesso a informazioni sensibili quali chiavi private, password.

La soluzione proposta si pone come obiettivo la protezione dei dispositivi IoT mettendo in sicurezza le maggiori vulnerabilità che possono essere trovate sui device.

2.3 Concetti base di sicurezza informatica

Per comprendere cosa è necessario per la protezione di un device IoT, è doveroso definire alcune proprietà di sicurezza sulle quali si basa l'architettura software sviluppata. Queste proprietà, garantite mediante l'utilizzo di algoritmi o protocolli di sicurezza, sono le seguenti:

- **Integrità dei dati**: garantisce che i dati non siano stati alterati;
- **Autenticazione singola o mutua**: l'autenticazione singola o mutua autentica le parti di una comunicazione accertandone l'identità;
- **Autenticazione dei dati**: garantisce sia l'origine dei dati che la loro integrità;
- **Confidenzialità**: garantisce che i dati non vengano letti da terzi non autorizzati. L'autorizzazione alla lettura dei dati è data dal possesso di un segreto condiviso tra le parti;
- **Non ripudio**: permette di identificare in modo inequivocabile l'autore dell'azione. Può avere valore legale ed è ammissibile come prova giudiziaria.

L'utilità di alcune di queste proprietà è subordinata al tipo di applicazione che si vuole progettare. Ad esempio, si supponga di avere un'applicazione che utilizzi l'orario per effettuare la sincronizzazione tra due dispositivi. In particolare, uno dei dispositivi trasmette l'orario agendo come master della comunicazione. In questo caso non è necessario avere la proprietà di confidenzialità per la trasmissione dell'ora, ma occorre garantire l'integrità in modo tale da rilevare eventuali modifiche sui dati che potrebbero causare la perdita di sincronizzazione tra i dispositivi.

Si procederà spiegando come è possibile garantire le proprietà elencate.

2.4 Confidenzialità

La confidenzialità è quella proprietà che dà segretezza ai messaggi in transito. Essa viene garantita utilizzando la crittografia, che utilizza le operazioni di cifratura e decifratura per nascondere un testo in chiaro oppure per riportare il testo oscurato in chiaro.
La cifratura può essere di due tipi:

- Cifratura simmetrica;
- Cifratura asimmetrica.

Queste due operazioni utilizzano delle assunzioni diverse, quindi è necessario analizzarle separatamente cosi da poter analizzare le caratteristiche di ognuna di esse. Le considerazioni che vengono fatte sono funzionali all'implementazione delle funzionalità. Per ulteriori informazioni si rimanda il lettore alla letteratura sulla sicurezza informatica.

Cifratura simmetrica

Le parti che vogliono condividere le informazioni, per poter utilizzare la cifratura simmetrica, devono essere in possesso di una chiave condivisa. Attraverso la chiave utilizzata in algoritmi di cifratura simmetrica è possibile garantire la **confidenzialità** del messaggio. La chiave condivisa è utilizzata sia per le operazioni di codifica che per quelle di decodifica. Infatti viene definita simmetrica perché le due operazioni, seppur diverse, sono eseguite utilizzando la stessa chiave.

Solitamente questi algoritmi sono classificati in :

- Algoritmi simmetrici a blocchi;
- Algoritmi simmetrici basati su stream di dati;

Le due tipologia differiscono per la dimensione del blocco di dati che viene cifrato per ogni singola operazione. Gli algoritmi simmetrici a blocchi utilizzano per ogni operazione un blocco del messaggio di dimensione fissa e formato da un gruppo di byte. Mentre gli algoritmi basati su stream utilizzano per singola operazione il byte

o, in alcuni casi, anche il bit.
Dovendo cifrare blocchi lunghi più di byte, sono necessari degli strumenti sia per dividere un messaggio più grande in blocchi base, sia per aumentare la lunghezza fino al raggiungimento della lunghezza del blocco base. L'operazione di aggiunta di testo viene chiamata **padding**. Esistono diverse tecniche per l'utilizzo del padding, per ulteriori informazioni ci si riferisca alla letteratura.

Gli algoritmi simmetrici a blocchi utilizzano diversi modi operazionali per rendere più efficace la cifratura. Ai fini del progetto sviluppato, verrà analizzato soltanto la modalità CBC (Cipher Block Chain). **CBC** viene utilizzato per suddividere il messaggio in diversi blocchi base e per legare ogni blocco a quello precedente . Per evitare attacchi di knowing-plaintext [10], è sconsigliato suddividere il messaggio in blocchi e cifrare singolarmente ciascun blocco. Invece, è più efficace contestualizzare la parte cifrata, ovvero legare il singolo blocco al resto del messaggio, ottenendo così una maggiore protezione.

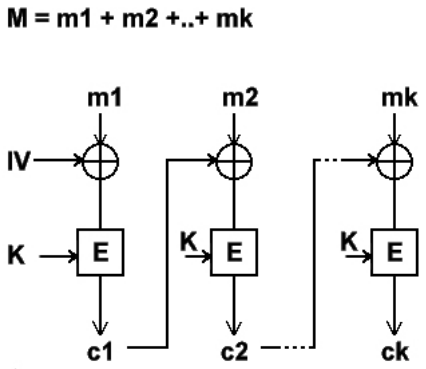

Figura 2.1: Architettura cbc

Al fine di garantire che anche il primo blocco venga processato utilizzando lo stesso procedimento, nella modalità CBC viene impiegato il vettore di inizializzazione (IV). Dopo aver suddiviso il messaggio in N blocchi, viene cifrato il *m1* dopo essere stato messo in $\bigoplus$ (operatore xor) con IV. Il risultato messo in $\bigoplus$ con il secondo blocco e poi cifrato proseguendo fino alla fine (figura 2.1).

La cifratura simmetrica ampiamente documentata in letteratura. Ai fini della soluzione progettata e realizzata si fa uso di AES (Advanced Encryption Standard)

per via delle alte prestazioni operazionali e dell'alto livello di sicurezza. L'algoritmo è basato su operazioni di spostamento, rotazione e mescolamento. Inoltre, può essere facilmente implementato sia in hardware che software. La configurazione scelta è AES 256-CBC, ovvero l'algoritmo AES utilizza una chiave di lunghezza 256 bit con modalità CBC. Quest'ultima configurazione garantisce la sicurezza della cifratura oltre l'anno 2031[11].

Cifratura asimmetrica

La cifratura asimmetrica utilizza una coppia di chiavi per le operazioni di cifratura e decifratura e per questo differisce dalla cifratura simmetrica.
Le chiavi accoppiate sono associate ad una stessa risorsa o identità. Entità, identità e risorsa sono utilizzati come sinonimi nell'elaborato. Una delle chiavi è mantenuta pubblica mentre l'altra non deve essere accessibile a nessuno se non al proprietario. La chiave pubblica può e deve essere distribuita al pubblico in modo tale da poter svolgere le operazioni di cifratura asimmetrica associate a questa chiave da parte di tutti. La cifratura asimmetrica garantisce le proprietà di **confidenzialità**, **autenticazione** e opzionalmente di **non ripudio**.

Un messaggio cifrato con cifratura asimmetrica può essere decifrato solo con la chiave accoppiata alla chiave di cifratura. Grazie a questa proprietà, la confidenzialità è garantita cifrando il messaggio con la chiave pubblica del destinatario cosi che il ricevente sia l'unico a poter decifrare il messaggio poiché è necessario l'utilizzo della chiave privata. L'autenticazione è garantita utilizzando, come chiave di cifratura del messaggio, la chiave privata, quindi questa operazione può essere esclusivamente eseguita dal possessore della chiave privata che ha necessità di dimostrare la sua identità.

In generale la chiave pubblica è contenuta all'interno di un certificato digitale. Quest'ultimo attesta che un'identità possiede la chiave privata associata alla chiave pubblica certificata.
In presenza di un certificato, gli algoritmi che utilizzano la coppia di chiavi certificate garantiscono la proprietà di **non ripudio**.
Gli algoritmi di cifratura asimmetrica più conosciuti e utilizzati sono DSA, Digital Signature Algorithm, e RSA, che prende il nome dai suoi creatori Rivest, Shamir,

Adleman. Il primo algoritmo può essere utilizzato esclusivamente per la firma digitale, operazione che garantisce integrità e autenticazione dei dati. Invece, RSA può essere utilizzato sia per la firma digitale che per la cifratura, quindi può garantire anche la confidenzialità. Date le proprietà garantite, per lo sviluppo del software è stato impiegato l'utilizzo dell'algoritmo RSA che viene analizzato nella sottosezione seguente in dettaglio.

RSA

L'algoritmo RSA si basa su un procedimento che utilizza i numeri primi e funzioni matematiche, che è quasi impossibile invertire. Dati due numeri primi, è molto facile stabilire il loro prodotto, mentre è molto più difficile determinare, a partire da un determinato numero, quali numeri primi abbiano prodotto quel risultato dopo essere stati moltiplicati tra loro. In questo modo si garantisce quel principio di sicurezza alla base della crittografia a chiave pubblica; infatti l'operazione di derivare la chiave segreta da quella pubblica è troppo complessa per venire eseguita in pratica. I passi da seguire sono:

1. Calcolare il valore di n, prodotto di p e q, due numeri primi molto elevati (sono consigliati valori maggiori di $10^{100}s$. Negli esempi in seguito si utilizzeranno numeri più piccoli per facilitare la lettura. Esempio: $n = p * q = 7 * 5 = 35$;

2. Calcolare il valore di $z = (p-1) * (q-1)$. $z = (p-1) * (q-1) = 24$;

3. Scegliere un intero D tale che D che sia primo rispetto a z, il che significa che i due numeri non devono avere fattori primi in comune. Esempio: $z = 24$ $D = 7$;

4. Trovare un numero E tale che $E * D \mod z = 1$, cioè che il resto della divisione tra $E * D$ e z sia 1. $E = 7 \rightarrow 49\%24 = 1$;

5. Dopo aver calcolato questi parametri inizia la cifratura. Il testo in chiaro viene visto come una stringa di bit e viene diviso in blocchi costituiti da kbit, dove k è il più grande intero che soddisfa la disequazione $2^k < n$.

A questo punto per ogni blocco M si procede con la cifratura calcolando $Mc = M^E \pmod n$. In ricezione, invece, per decifrare Mc si calcola$Mc^D \pmod n$. Come si può capire da quanto appena detto per la cifratura si devono conoscere E ed n che

quindi costituiranno in qualche modo la chiave pubblica, mentre per decifrare è necessario conoscere D ed n che quindi faranno parte della chiave segreta.

RSA viene considerato sicuro perché non è ancora stato trovato il modo per fattorizzare numeri primi molto grandi, che significa riuscire a trovare p e q conoscendo n. Nel corso degli anni l'algoritmo RSA ha più volte dimostrato la sua robustezza: per riuscire a derivare dalla chiave pubblica quella privata con 129 cifre, svelando il meccanismo con cui quella chiave generava messaggi crittografati, sono necessari 8 mesi di lavoro coordinato tra 1600 macchine da calcolo, facendole lavorare in parallelo collegate tra loro attraverso Internet. [12]

Data la mole delle risorse necessarie per abbattere la sicurezza dell'algoritmo RSA, è chiaro come un attacco alla privacy di un sistema a doppia chiave non sia praticamente realizzabile. Inoltre, l'esempio fatto è su una chiave di 129 cifre mentre i programmi di crittografia attualmente a disposizione prevedono chiavi private con lunghezza minima di 2048 bit. Questo rende praticamente infattibile la fattorizzazione, visto anche che l'ordine di grandezza dei tempi necessari per operazioni di questo tipo è esponenziale passando in fretta da qualche giorno a centinaia di anni.

2.5 Integrità dei dati

Per garantire l'integrità dei dati si definisce un nuovo tipo di messaggio chiamato **digest**. Il digest non è altro che un *riassunto* di dimensione fissa del messaggio iniziale. Esso deve avere le seguenti caratteristiche:

- Facile e veloce da calcolare;
- Difficilmente invertibile;
- Resistente alle **collisioni**.

Si ha una collisione quando, dati due messaggi diversi, il digest calcolato è identico. Questo può scaturire dei problemi poiché il digest è utilizzato per la firma digitale. Si supponga di calcolare un digest su un documento che riporta *Versare 1.000 euro a ...* e poi si supponga di calcolare un digest su un documento diverso

con riportato *Versare 10.000 euro a ...* ed esso sia identico. L'utente che firmerà il documento sarà ingannato poiché potrebbe essere facilmente sostituito con l'altro documento che ha lo stesso digest ma con contenuto differente. Quindi è necessario che l'algoritmo utilizzato per il calcolo del digest difficilmente dia risultati uguali. Il digest viene calcolato utilizzando funzioni di hash crittografiche come SHA1, SHA2 che hanno una probabilità di collisione veramente minima.Il digest, inoltre, viene utilizzato per evitare operazioni onerose sull'intero messaggio come operazioni di cifratura asimmetrica.

2.6 Autenticazione

L'autenticazione è utilizzata per garantire che i dati provengano con assoluta certezza da una determinata fonte. L'integrità è una proprietà necessaria per garantire che i dati non vengano manomessi, ma potrebbe non essere sufficiente nella maggior parte dei casi. Spesso è necessario accertarsi che la fonte del messaggio sia autentica. Ciò è possibile utilizzando due diverse modalità, basate sui metodidi cifratura descritti precedentemente. Entrambi i metodi utilizzano delle operazione che elaborano il digest per poter garantire sia l'integrità del messaggio che l'autenticità della fonte.

Autenticazione dati usando la cifratura simmetrica

Utilizzando la cifratura simmetrica, è possibile garantire anche l'autenticazione grazie all'impiego del MAC (Message Authentication Code).
Il MAC necessita in ingresso una chiave simmetrica dando un valore aggiunto al digest poiché per il suo calcolo viene utilizzata la chiave simmetrica. Questo meccanismo garantisce sia l'integrità che l'autenticazione, qualora il messaggio venisse modificato in transito, poiché il calcolo del MAC diviene impossibile se non si è in possesso della chiave. La letteratura degli algoritmi per il calcolo del MAC è vastissima. In particolare, uno tra i più importanti e utilizzati è l'**HMAC** (Hashing Message Authentication Code) dovuta alla sua facilità di integrazione e la sua velocità di computazione. L'HMAC che rientra nella categoria dei *keyed-digest*.

L'HMAC viene calcolato utilizzando le funzioni di hash crittografiche come SHA256 ed è impiegato nel protocollo SSL/TLS. L' HMAC risulta essere più resistente alle collisioni rispetto alla funzione di hash associata poiché l'attaccante non conosce la chiave utilizzata e non risulta possibile recuperarla dal MAC generato. Inoltre, viene consigliato il suo impiego per la facilità di integrazione con qualunque algoritmo di hash. Il carico computazionale che l'HMAC introduce rispetto all'algoritmo di hash scelto è minimo poiché una delle caratteristiche degli algoritmi di hash è la velocità computazionale.

Scelta la funzione di hash **H**, il messaggio M viene suddiviso in blocchi da n bits pari alla dimensione del risultato dell'algoritmo di hash. Se la chiave simmetrica differisce in lunghezza rispetto al blocco base, deve essere trasformata. In particolare:

- $K' = K + zeropadding \iff |K| < n$;
- $K' = H(K) \iff |K| > n$.

Ai fini del calcolo si introducono i valori $ipad = 0x36$ e $opad = 0x5c$ Una volta effettuate le operazioni precedenti l'HMAC viene calcolato

$$HMAC(M) = H((K' \oplus opad)||((K' \oplus ipad)||M'))$$

Dove M' è il messaggio suddiviso in blocchi e $||$ descrive la concatenazione tra le variabili.

Autenticazione dati usando la cifratura asimmetrica

La cifratura asimmetrica, in generale, non viene utilizzata per cifrare grandi moli di dati poiché le operazioni di cifratura richiedono molte risorse e sono onerose a livello computazionale. Per evitare queste operazioni, si utilizza il digest del messaggio e su quest'ultimo vengono applicate le operazioni di cifratura garantendo oltre l'integrità dei dati, anche la autenticazione. Se le chiavi asimmetriche sono certificate si ottiene anche la proprietà di non ripudio dell'azione e l'operazione descritta si chiama firma digitale.

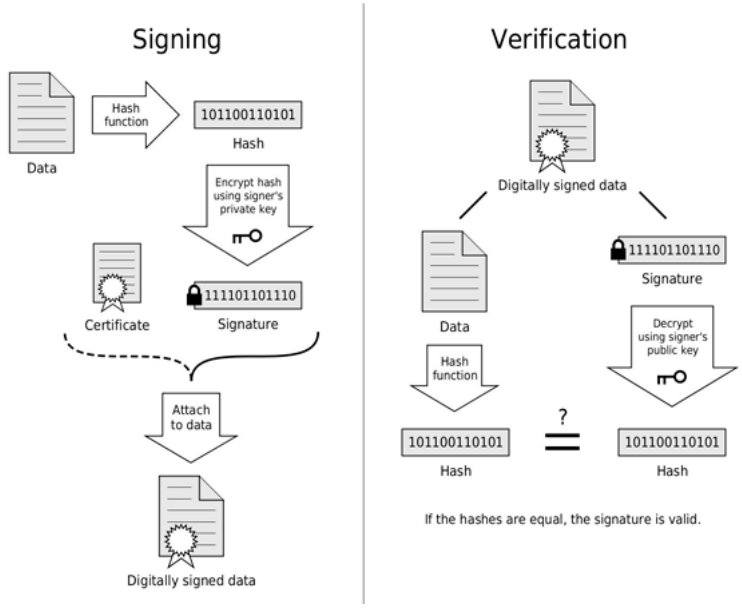

Figura 2.2: Operazioni di Firma e della sua Verifica

Per effettuare l'operazione di firma è necessario che il proprietario cifri il digest calcolato su un messaggio con la propria chiave privata e allegando il risultato, ovvero la firma, trovato al messaggio originale. In questo modo, il ricevente può decifrare la firma utilizzando la chiave pubblica, ottenendo il digest per poter verificare che il messaggio sia arrivato senza aver subito modifiche.

2.7 Non Ripudio, Certificati e Public Key Infastructure

In questa sezione vengono descritti i meccanismi che permettono l'associazione delle chiavi ad una risorsa attraverso i certificati. Vengono inoltre descritti i meccanismi di rilascio e di revoca dei certificati digitali.
Un certificato digitale, tipicamente, lega un'entità, fisica o digitale, ad una coppia di chiavi. Il certificato contiene diverse informazioni come: chiave pubblica, l'autorità che ne ha autorizzato l'emissione, data di rilascio, il periodo di validità. Il certificato viene rilasciato e firmato da una **Certification Authority** (CA). Una CA è un soggetto terzo di fiducia, pubblico o privato, che è stato abilitato ad emettere certificati

digitali. Un certificato, inoltre, può essere revocato in qualunque momento sia dalla CA che dall'identità a cui appartiene.

Public Key Infrastructure

La PKI è un insieme di ruoli, politiche e procedure necessari per creare, gestire, distribuire, utilizzare, archiviare e revocare i certificati digitali e gestire la crittografia a chiave pubblica. L'associazione è stabilita attraverso un processo di registrazione e rilascio di certificati presso e da un'autorità di certificazione (CA) [13]. Esiste una struttura gerarchica internazionale formata da tutte le CA chiamata **Chain of Trust**.

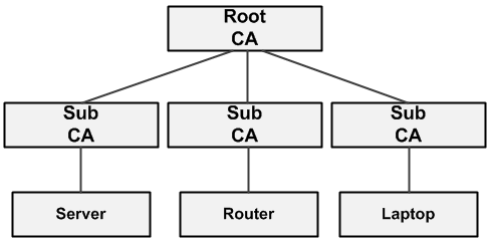

Figura 2.3: Chain of Trust

La verifica di un certificato viene attuata risalendo la catena di certificazione fino a trovare una CA *trusted*. La Root CA possiede un certificato *self-signed*.

Rilascio Certificato

La PKI mette a disposizione dei protocolli per il rilascio di certificati. I protocolli utilizzati sottopongono il soggetto a dimostrare il possesso di una coppia di chiavi e la propria l'identità per la richiesta del certificato

Un certificato per essere valido necessita della firma di una CA.
La procedura per richiedere la firma è descritta secondo lo standard PKCS #10 L'operazione in questione viene chiamata **Certificate Signing Request** (CSR).
La CA per validare l'identità fisica del soggetto ne richiede il riconoscimento a seguito della richiesta di firma del certificato.

In seguito, viene richiesta la prova del possesso della chiave privata utilizzando una modalità di autenticazione chiamata **challenge**. Essa consiste nell'invio di un messaggio univoco chiamato *nonce* da parte di chi vuole verificare l'identità. Il nonce viene cifrato con la chiave pubblica. La parte che vuole provare la propria identità, avrà il compito di decifrare il messaggio e rispedirlo indietro.

Certificato X.509

Il formato X.509 è lo standard più utilizzato per la descrizione di un certificato digitale. In un certificato X.509 si trovano diversi campi con dettagli sul certificato. Alcuni campi del seguente certificato sono:

- Subject Name: nome digitale dell'entità a cui è stato rilasciato il certificato;
- Issuer: quale CA ha rilasciato il certificato;
- Validity;
- Algorithm: algoritmo usato per la firma e la sua verifica;
- Issuer Signature: firma da parte della CA.

L'analisi di questi campi può essere sufficiente per verificare la validità e l'accettabilità del certificato all'interno del progetto sviluppato. La validità deve essere accertata verificando la firma del certificato e risalendo la catena di certificazione

Revoca certificato

Ogni certificato può essere revocato prima della data di scadenza. La revoca può essere effettuata sia dalla CA che dall'utente a cui è stato lasciato il certificato. L'annullamento del certificato segue una procedura bene precisa.

Dopo essere stato revocato il certificato viene messo all'interno di una lista. La Certificate Revocation List **(CRL)** è un elenco, emesso dalla CA, che attesta quali certificati sono stati dismessi prima della data di scadenza e, quindi, da non considerare più attendibili. Questa lista viene generata periodicamente ed è mantenuta aggiornata. All'interno della CRL viene indicato anche quale sarà la prossima data di rilascio. La CRL, però, crea un overhead nelle applicazioni connesse che non può

essere sottovalutato poiché ad ogni richiesta viene scaricata tutta la lista .
In alternativa, per applicazioni connesse, può essere utilizzato un protocollo chiamato **OCSP** (Online Certificate Status Protocol) che verifica, tramite una richiesta, la validità di un singolo certificato eliminando il problema dell'overhead e velocizzando le operazioni di verifica.

2.8 Protocollo di comunicazione sicura (SSL/TLS)

Precedentemente è stato ampiamente descritto quanto sia importante un protocollo di comunicazione sicura per i device IoT. Ad oggi la tecnologia più utilizzata per questo tipo di operazioni è il protocollo SSL/TLS. Esso può essere posizionato subito sopra il layer trasporto ed è indipendente rispetto all'applicazione che lo utilizza. Il protocollo SSL è stato inventato nel 1994 da Netscape per rendere sicure le transazioni via Internet. Nel 1990 SSL 3.0 è stato standardizzato da IETF sotto il nome di TLS 1.0. Oggigiorno la versione più utilizzata è la versione TLS 1.2.
Il protocollo garantisce diverse proprietà di sicurezza alla comunicazione tra cui:

- Confidenzialità;
- Autenticazione dati;
- Autenticazione singola;
- Autenticazione mutua (non obbligatoria);
- Integrità;
- Non-ripudio.

Attraverso la descrizione dei sotto protocolli sui quali si basa TLS, si comprenderà come vengono garantite le proprietà presentate.

Sotto protocolli SSL/TLS

Il protocollo SSL/TLS include quattro protocolli:

- Handshaking Protocol;

- Record Protocol;
- Alert Protocol;
- Change Cipher Spec Protocol.

Questi protocolli sono inquadrati nel modo seguente:

Figura 2.4: Sottoprotocolli SSL/TLS

Tra questi quattro protocolli, quelli più interessanti e utilizzati nelle sessioni sono l'handshaking procotol e il record protocol. Verranno descritti nel dettaglio poiché le proprietà garantite da SSL/TLS possono essere ottenute utilizzando solo questi due protocolli.

Handshaking Protocol

L'Handshaking protocol è utilizzato per la gestione della sessione SSL/TLS. Il protocollo in analisi permette di negoziare gli algoritmi di cifratura, autenticare una o entrambe le parti, scambiarsi le chiavi di cifratura e iniziare la sessione di comunicazione sicura. Questo viene fatto attraverso dei messaggi ben precisi schematizzati di seguito

1. **Client Hello**: Nel messaggio sono contenuti la versione tls preferita, bytes random per la generazione del segreto, session id, lista di cipher suite supportate, lista di metodi di compressione;
2. **Server Hello**: nel pacchetto sono contenute le scelte effettuate dal server tra le opzioni date dal client. Quindi, si avranno la versione di TLS, la cipher

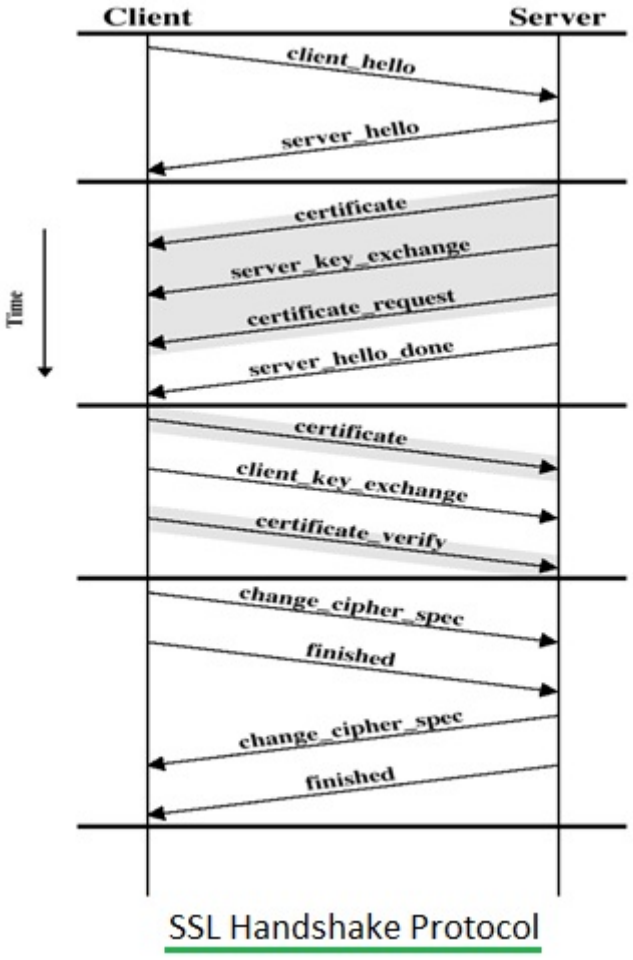

Figura 2.5: Handshaking protocol

suite, l'algoritmo di compressione e ancora il session id e bytes random per la generazione del segreto condiviso;

3. **Server Certificate**: Il Server invia il certificato al client;

4. **Server Certificate Request**: Opzionale. Richiesta al client di un certificato emesso da una particolare CA;

5. **Server Key Exchange**: Opzionale. Il server invia una chiave pubblica temporanea atta all'uso di meccanismi effimeri;

6. *Client Certificate*: Opzionale. Il client invia il proprio certificato al server;

7. **Client Key Exchange**: utilizzato per lo scambio di informazioni tra client e serve per il calcolo della chiave simmetrica condivisa;

8. **Change Cipher Spec (Client/Server)**:Questo messaggio ha lo scopo di segnalare i cambiamenti delle strategie di cifratura;

9. **Finished (Client/Server)**: Questo è il primo messaggio che utilizza gli algoritmi di sicurezza concordati e serve per verificare l'integrità dello scambio dei messaggi precedentemente effettuato.

Client e Server si scambiano informazioni servendosi del Record Protocol.

Record Protocol

Il sotto protocollo Record è usato per l'incapsulamento dei dati provenienti dai livelli superiori ed è basato su un protocollo di trasporto affidabile come il TCP.

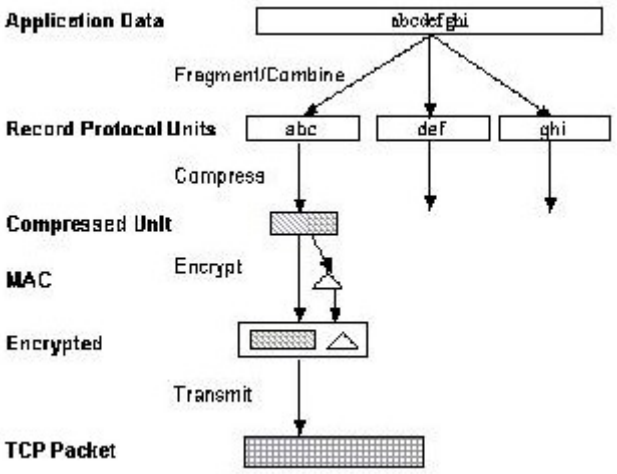

Figura 2.6: Record protocol

Il Record protocol (Figura 2.6) utilizza i messaggi trasmessi dal livello superiore, li frammenta in blocchi di dati (record di 214 byte o meno) e opzionalmente li comprime. Dopo, applica un algoritmo per il calcolo del MAC, li cifra, e trasmette il risultato ottenuto. I dati ricevuti vengono decifrati, verificati, decompressi, e riassemblati, quindi vengono trasmessi al livello più alto. Tutti gli SSLPlaintext record sono compressi.

2.9 Hardware Sicuri

Gli Hardware sicuri, anche chiamati Hardware Trust Anchor (HTA), sono componenti primari per costruire un sistema con proprietà di sicurezza. Questi sono impiegati principalmente per:

- Protezione di dati sensibili in modo da non poter essere manipolati;
- Esecuzione di funzioni crittografiche per alleggerire il carico sulla CPU.

L'utilizzo dell'HTA è un requisito necessario per garantire la protezioni di informazioni quali: certificati a chiave pubblica, chiavi private e funzioni crittografiche. Essa viene utilizzata come Root of Trust del dispositivo, quindi questo device è fidato per definizione.
Esistono diversi standard:

- Secure Hardware Extension (SHE);
- Hardware Security Module (HSM);
- Trusted Platform Module (TPM).

Qualora questi dispositivi non fossero disponibili, esistono altre periferiche hardware che aiutano, se gestiti correttamente, nella protezione di informazioni sensibili. Le periferiche in questione in particolare, gestiscono la memoria e sono: la **Memory Management Unit** (MMU) e la **Memory Protection Unit** (MPU). La MMU può avere vari compiti tra cui la traduzione degli indirizzi logici e simbolici in indirizzi fisici, necessaria per la gestione della memoria virtuale. La memoria viene suddivisa in indirizzi virtuali e possono essere gestiti in modo diverso. Ogni sezione può avere dei permessi specifici come: lettura o scrittura e lettura. La sezione di memoria inoltre può essere configurata per l'accesso in modalità privilegiata o in modalità utente. Grazie a queste caratteristiche, protegge memoria individuando accessi illeciti.

L'MPU è una MMU però di dimensioni ridotte, cioè gestisce solo otto sezioni di memoria ed è facilmente reperibile in dispositivi IoT con prestazioni limitate.

L'impiego dell'MPU in un sistema che non integra HTA aiuta a proteggere le informazioni sensibili grazie alle caratteristiche dette prima.
All'interno della tesi l'MPU è stata utilizzata per effettuare protezione dati e isolamento della memoria in base al processo in esecuzione. Maggiori dettagli verranno discussi nel capitolo 4

Capitolo 3

Studio delle Vulnerabilità e Analisi della soluzione

Dopo aver compreso come l'IoT si sta sviluppando e quali rischi questa tipologia di dispositivi deve affrontare, in questo capitolo vengono descritte le metodologie adottate per rispondere in modo efficace alla risoluzione delle maggiori problematiche dei device IoT.

I problemi di sicurezza in un sistema IoT sono ad ogni livello, a partire da quello fisico, con la manomissione del dispositivo, fino a quello applicativo, attraverso errori sia di implementazione sia di architettura software. Quindi è necessario definire quali problematiche creano le maggiori criticità cosi da progettare una soluzione efficace contro queste minacce.

3.1 Threat Model

Dato che una soluzione software non riesce a difendere il dispositivo da tutti i tipi d'attacchi, è necessario definire quali sono le minacce più critiche risolvibili via software. La soluzione poi può essere estesa per creare una protezione più efficace da ulteriori vulnerabilità.

Questa sezione si occupa di definire quali sono le minacce da affrontare attraverso l'analisi legata alle vulnerabilità dei dispositivi in esame.

3.1.1 Superficie d'attacco sul dispositivo IoT

In un sistema IoT esistono diversi rischi. In particolare, i maggiori attacchi ai dispositivi IoT mirano alle debolezze della memoria eseguendo memory corruption, memory over-read, memory over-write o corruzione del firmware. Le minacce elencate si manifestano maggiormente su questa tipologia di dispositivi a causa della mancanza di meccanismi di sicurezza delle connessioni wireless che permettono di raggiungere il dispositivo pur non avendone diretto l'accesso fisico. La tabella 3.1 riporta tre macro-vulnerabilità su cui si vuole porre particolarmente l'attenzione [6].

Esecuzione di codice arbitrario	Buffer Overflow, Autenticazione debole
Perdita informazioni Sensibili	Buffer over-read, Autenticazione debole, packet sniffing
Corruzione Firmware	Aggiornamento del firmware da remoto non sicuro

Tabella 3.1: Rischi

3.1.2 Mitigazione dei rischi

Per poter far fronte alle minacce elencate nella tabella 3.1 si deve implementare una logica software che protegga il dispositivo, il firmware al suo interno e la sua operabilità. Si cerca di sviluppare una soluzione lightweight e pronta all'uso che venga utilizzata come base dalle piattaforme IoT presenti sul mercato per lo sviluppo di applicazioni connesse. Dato che è possibile utilizzare soluzioni software per la risoluzione dei rischi sopraelencati, la protezione fisica non è stata ritenuta necessaria per questo progetto.

Connessione alla rete

I servizi che un device IoT offre sono basati su connessioni wireless come Bluetooth, Wi-fi, NFC o altri tipi di tecnologie. Inoltre, sempre più dispositivi utilizzano la connessione Internet per offrire più servizi all'utente. La rete espone un dispositivo

IoT a molti rischi quindi proteggere la comunicazione tra device è una priorità dato che vengono trasmessi dati sensibili attraverso Internet.
Una connessione sicura evita principalmente:

1. Manomissione dei dati;

2. Perdita di informazioni sensibili.

Anche se la connessione sicura garantisce l'integrità dei dati in transito, non si può escludere che una delle due parti sia un utente malevolo. è necessario, quindi, creare un meccanismo per poter stabilire se il soggetto che vuole accedere alle informazioni ne abbia l'autorizzazione. In questo modo si assicura sia che il dispositivo non riceva (invii) dati da (a) fonti non attendibili sia che il server stia comunicando con il vero device.

Nell'architettura progettata e sviluppata, la connessione sicura e l'accesso autenticato sono gestiti con l'aiuto di certificati. Questi sono utilizzati sia dal dispositivo, per potersi autenticare nei confronti del servizio, che dal servizio che vuole aver accesso ai dati del device.

Il dispositivo deve gestire informazioni sensibili utilizzati per poter stabilire la connessione sicura. Qualora queste informazioni venissero corrotte, manipolate o sottratte la connessione sicura potrebbe essere manomessa. Tra le informazioni critiche si hanno chiavi private e certificati digitali delle terze parti fidate. Queste informazioni devono essere protette e quindi è necessario prevedere ulteriori meccanismi di protezione per evitare che la connessione sia compromessa.

La sicurezza in questa tipologia di dispositivi si deve propagare a più livelli per poter essere efficace. In particolare per proteggere informazioni sensibili si utilizzano dei meccanismi di protezione della memoria sia hardware, come i device elencati nella sezione 2.9, che software attraverso offuscamento della memoria[14].

Integrità del firmware

Si assuma che un attaccante riesca ad inserire nel dispositivo del codice arbitrario e quest'ultimo venga eseguito. Qualora succedesse è possibile che l'attaccante entri in possesso di informazioni sensibili o utilizzi il device in modo anomalo, mettendo a rischio la sicurezza dell'utente che ne ha accesso, qualora i dispositivi fossero

integrati in macchinari industriali. Quindi è necessario avere dei meccanismi per verificare che il firmware da eseguire non sia stato manomesso da terzi. Qualora la verifica venisse fatta all'accensione del dispositivo si parla di **secure boot**. Il secure boot deve essere considerato dal dispositivo come una terza parte fidata, ovvero una funzionalità che non può essere manomessa. L'integrità del secure boot può essere garantita utilizzando una memoria di sola lettura o un hardware sicuro poiché queste due tipologie di periferiche non possono essere manomesse. Dentro questi supporti dovranno essere memorizzati:

- Dati sensibili come certificati a chiave pubblica del dispositivo;
- Libreria crittografica;
- Logica di verifica del firmware.

L'integrità di queste informazioni devono essere garantite a livello hardware poiché la soluzione progettata si baserà su queste assunzioni. Inoltre i dati presenti all'interno del dispositivo hardware sicuro sono utilizzati dal secure boot per la sua esecuzione.

Applicazioni

Le applicazioni IoT utilizzano la connessione internet per estendere le proprie potenzialità utilizzando dei servizi messi a disposizione. Si supponga che, a causa di una cattiva implementazione, venga trovata una vulnerabilità nel protocollo di comunicazione sicura e che un attaccante riesca a sfruttarla per ottenere l'accesso al dispositivo. Dato che la connessione ad internet espone il device ad attacchi di vario tipo, esso stesso deve proteggersi dalle applicazioni che ricevono dati dall'esterno. Proprio per questa ragione, l'applicazione che ha diretto accesso ad internet verrà considerata non fidata. In quanto *untrusted*, la funzionalità che utilizza la connessione deve essere isolato dal sistema. Si introducono meccanismi di controllo d'accesso alla memoria, per evitare che codice malevolo possa essere eseguito, danneggiando sottraendo dati sensibili.

3.2 Metodologia

Il tipo di approccio utilizzato per il design di un sistema o un'applicazione è descritto dalla metodologia **TARA** (threat assessment risks analysis), che utilizza una tabella di valutazione del rischio per focalizzarsi sulla protezione delle vulnerabilità categorizzate più critiche del progettista. La metodologia TARA è descritta dallo standard SAE J3061. In questa tesi, l'analisi citata è stata ampiamente sviluppata sia nel capito 2 che nella sezione 3.1.
Dopo aver definito i rischi da cui si vuole proteggere, è necessario adottare una strategia efficace per la progettazione di una soluzione idonea. Diversi articoli spiegano quali sono le*best practices* da adottare nei sistemi IoT per renderli più sicuri. L'NHTSA, ente statunitense responsabile della sicurezza autostradale, ha rilasciato nel 2016 un documento chiamato *Cybersecurity Best Practices for Modern Vehicle* [15] nel quale vengono spiegate le pratiche da mettere in atto per evitare cyber attack nei veicoli.
In particolare, questo documento dichiara che una buona metodologia per proteggersi da attacchi è prevedere dei meccanismi di sicurezza a partire dalla progettazione. Questo tipo di pratica è descritta sotto il nome *secure by design*. Oggigiorno questa è una teoria molto conosciuta ma non sempre adottata a causa dei costi che comporta in termini di risorse.
L'NHTSA consiglia di utilizzare la metodologia di secure by design per diverse ragioni.

- **Risparmio economico**: rendere sicuro un dispositivo già prodotto è più costoso rispetto alla progettazione sicura del sistema.

- **Rischio di mettere in produzione un prodotto non sicuro**. Se il dispositivo evidenzia falle in fase di test, si è troppo vicini alla fase di produzione per poter intervenire.

Per rendere fruibile ai produttori di dispositivi IoT la metodologia secure by design, lo stack software è già stato pensato per essere resiliente ad alcuni attacchi noti.

3.3 Requisiti

Le funzionalità di sicurezza da implementare per proteggere il dispositivo dai rischi analizzati nella sezione 3.1 possono essere sintetizzate secondo il seguente elenco.

- Assicurarsi che il firmware da eseguire non sia stato manomesso;
- Isolamento delle applicazioni connesse dal resto del sistema;
- Aggiornare in modo sicuro il firmware del dispositivo da remoto.

La soluzione sviluppata è pensata per ospitare ulteriori applicazioni rispetto a quelle implementate per la sicurezza del dispositivo, quindi all'interno del firmware è utilizzato un sistema operativo in modo tale da integrare facilmente ulteriori funzionalità.

3.3.1 Integrità del firmware

Un modo possibile per garantire l'integrità del firmware è utilizzare la firma digitale. Attraverso all'impiego della firma digitale si rende semplice l'operazione di verifica dell'integrità del firmware. Il firmware presente sul dispositivo è accoppiato ad una firma digitale calcolata su di esso o una sua parte.
La funzionalità che prevede la verifica del firmware è descritta sotto il nome di **secure boot**. Il secure boot può essere implementato in diversi modi. In particolare, si prendono in esame le seguenti tipologie:

- One shot secure boot;
- Just-In-Time (or Lazy) secure boot;
- Parallel secure boot.

Anche se diversi, questi tipi di secure boot sono basati su un set di operazioni in comune.

1. Lettura del binario in memoria;
2. Calcolo del digest con un algoritmo di hash sui dati letti;

3. Lettura e decifratura firma digitale;

4. Confronto tra il valore calcolato e quello decifrato.

Tra le tre tipologie varia il modo in cui queste operazioni vengono effettuate. **One Shot Secure Boot** utilizza una singola firma per la verifica dell'integrità del software del dispositivo. Esso è il più semplice delle tipologie sia dal punto di vista implementativo che da quello teorico. L'operazione, in questo caso, diventa onerosa in termini di tempo e risorse. Il secure boot potrebbe impiegare qualche secondo per il completamento della verifica, difficile da gestire in sistemi hard real-time.
Dato che esistono contesti in cui il tempo di accensione è cruciale, per questa tipologia di applicazioni può essere utile l'utilizzo del **Just-In-Time Secure Boot**.

In questa tipologia di secure boot si hanno tante firme quante le sezioni di binario da verificare in modo tale da utilizzare meno risorse computazionali in fase di verifica. La verifica viene effettuata su una parte del firmware alla volta avendo come beneficio un tempo di avvio minore. Il firmware viene verificato solo nel momento in cui sta per essere utilizzato. Ad esempio si supponga che il binario sia suddiviso in 3 sezioni: una per il bootloader, una per il sistema operativo e una per le applicazione. Prima di avviare la configurazione di sistema, ovvero la fase di bootloader, verrà verificata la sua integrità prima di procedere al suo avvio. Anche se il tempo di avvio della sezione da verificare aumenta, questo non è critico in quanto il tempo in più è ancora sostenibile dal sistema. Se come vantaggio si ottiene un tempo di accensione minore, lo svantaggio di questa tipologia è la maggiore memoria occupata dalle firme digitali. Se in sistemi hard-real time la memoria può non essere un problema, nei sistemi IoT essa diventa un aspetto fondamentale. In questa tipologia di device si preferisce utilizzare maggiori risorse per le operazioni computazionali ottimizzando così l'occupazione della memoria.
La terza tipologia è una fusione delle prime due.
All'avvio del sistema vengono eseguite n verifiche della firma digitale in parallelo. Se tutte le verifiche sono state positive allora si procede con l'avvio del sistema. I contro di questa tipologia sono principalmente due:

- Memory Footprint cioè la memoria occupata per l'esecuzione e la memorizzazione del firmware;
- I device IoT solitamente non hanno processori multi-core.

Nelle tre le tipologie l'unica parte esclusa per il calcolo della firma è la parte relative al secure boot, librerie crittografiche, la firma digitale e i certificati a chiave pubblica poiché queste informazioni sono memorizzate in hardware sicuri.

Data la memoria limitata e la presenza di un unico core, si ricava che la prima tipologia, anche se più lenta, è quella più appropriata per un sistema IoT che non ha vincoli sul tempo di avvio.
Il secure boot necessita di alcuni componenti per il suo funzionamento tra cui:

- Gestione dei certificati;
- Libreria di funzioni crittografiche.

3.3.2 Isolamento delle applicazioni

Molti attacchi su questa tipologia di device sfruttano la cattiva gestione dei buffer, come buffer overflow e buffer over-read, per sottrarre o immettere informazioni all'interno del dispositivo bersaglio. Dato che la tipologia di dispositivi, su cui si vuole implementare la soluzione, hanno risorse limitate, è necessario utilizzare soluzioni semplici e leggere per gestire l'accesso alla memoria, cioè un meccanismo che permetta di controllare quali applicazioni hanno accesso alle diverse sezioni di memoria e con quali permessi tale accesso è consentito. Esistono delle periferiche che integrano delle modalità di controllo degli accessi in memoria (sezione 2.9). Per la gestione della memoria si impiega l'utilizzo di un dispositivo hardware dedicato così da alleviare il carico computazionale del device. In particolare, si utilizza l'MPU per l'implementazione della soluzione scelta, poiché è integrata nella maggior parte dei device IoT. L'MPU permette di suddividere la memoria in sezioni e, per ogni sezione, selezionare un sottoinsieme di permessi d'accesso per il codice in esecuzione e assegnare quali privilegi l'applicazione dispone in quel momento (sezione 2.9).
Per rendere efficace l'impiego dell'MPU, per ogni task o applicazione è necessario gestire le modalità d'accesso alla memoria. Dato che ogni applicazione avrà un compito diverso da svolgere, è necessario comprendere a quali periferiche e variabili deve accedere. Si supponga si avere un task che utilizza il Bluetooth ma non la porta USB. È necessario fornire accesso alla prima periferica ma è sconsigliato dare accesso alla seconda poiché si aumenterebbe la superficie di attacco del dispositivo. L'MPU

inoltre ha la facoltà di limitare l'accesso alle funzioni o dati sensibili e può essere utilizzata per offuscare dati, codice e periferiche finché il dispositivo è in stato di esecuzione.
Per poter utilizzare efficacemente l'MPU, si devono cercare dei modi semplici per la sua gestione durante il context-switch. Inoltre, è utile rendere la configurazione di questa periferica semplice mentre un'applicazione è in esecuzione.

3.3.3 Aggiornamento sicuro del firmware

Nella sezione 3.1 si è sottolineato come la connessione sia un punto critico in un dispositivo IoT e come questa possa essere utilizzata per compromettere la sicurezza del device. Una connessione sicura è necessaria se si vuole effettuare l'aggiornamento sicuro del firmware, ed è efficace per evitare:

- La perdita intellettuale del firmware;
- La manomissione del binario in transito.

Il secure, o authenticated, firmware update è composto da diverse funzionalità, alcune di esse rendono più sicuro il device e la sua connessione alla rete. I componenti necessari alla sua implementazione sono descritti dalla seguente lista:

1. Connessione sicura;
2. Accesso Autenticato;
3. Protezione la memoria;
4. Verifica il firmware ricevuto;
5. Installazione del binario.

Si sono ampiamente spiegati i motivi per cui è necessaria una connessione sicura, viceversa alcuni punti della lista sono stati poco discussi. L'accesso autenticato è necessario dal momento in cui il device deve assicurarsi che la sorgente delle informazioni sia attendibile e che quest'ultima abbia le autorizzazioni necessaria per poter fornire il nuovo firmware al dispositivo. Una volta autenticata la sorgente e scaricato il firmware si effettua la verifica del firmware poiché non è detto che il firmware non

sia stato manomesso.

L'accesso autenticato e la connessione sicura elevano il livello di sicurezza del dispositivo, tuttavia questi meccanismi offrono protezione solo al canale di comunicazione instaurato dal task. Per garantire una maggiore resilienza agli attacchi è necessario implementare meccanismi di protezione di memoria e di verifica del firmware scaricato, con le modalità descritte nelle sezioni precedenti. Il firmware aggiornato sarà, dunque, accoppiato ad un firma digitale che ne garantisce l'integrità. Nel capitolo seguente si spiegano le scelte implementative per la realizzazione delle funzionalità obiettivo della tesi: **Secure Boot e Authenticated Firmware Update** e come queste si scostano dai requisiti analizzati.

Capitolo 4

Implementazione

L'architettura software proposta per lo sviluppo del sistema è quella mostrata nella figura 4.1. Questa architettura è la base che viene utilizzata per lo sviluppo di applicazioni IoT poiché integra delle funzionalità di sicurezza utili capaci di creare un ambiente protetto attraverso l'impiego di una connessione sicura e dell'accesso autenticato. Il tipo di architettura proposta può essere utilizzata in modo indipendente dal dispositivo scelto. In base al device si dovranno configurare le interfacce con le periferiche del dispositivo.

Una volta definiti i componenti necessari per le funzionalità descritte, è importante comprendere se è possibile utilizzare moduli esistenti. Nel caso di elementi già disponibili si analizza quali vantaggi essi offrono e se esiste una configurazione adatta per le esigenze progettuali.

Il capitolo si concentra su la scelta, configurazioni o implementazioni di:

1. Device target;
2. Sistema Operativo;
3. Libreria Crittografica;
4. Protocollo comunicazione sicura;
5. Organizzazione memoria;
6. Secure Boot;
7. Authenticated Firmware Update.

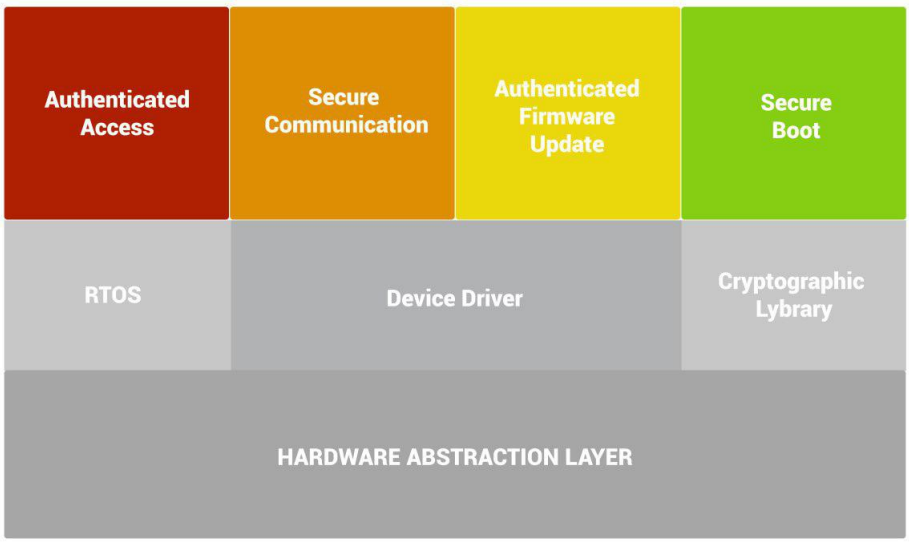

Figura 4.1: Architettura Software

4.1 Device Target

Le caratteristiche che differenziano i device IoT sono:

- Il basso costo;
- Le prestazioni limitate;
- Il consumo ridotto;
- L'utilizzo di sensori per catturare ed elaborare informazioni.

Il costo contenuto e le prestazioni limitate sono dovute all'utilizzo di sensori che alleggeriscono il carico computazionale del dispositivo.

A fronte di queste caratteristiche, è necessario individuare un dispositivo che rispetti questi requisiti e che abbia almeno una delle periferiche di sicurezza presentate nella sezione 2.9.
ARM mette a disposizione la famiglia di processori **Cortex-M**. I dispositivi basati su Cortex-M sono stati progettati per rispondere alla richiesta del mondo IoT.

Questi integrano un buon numero di periferiche utili per l'integrazione di sensori e di dispositivi esterni. In particolare, il processore scelto per l'implementazione della soluzione progettata è ARM Cortex-M4 poiché ha integrata l'MPU, ampiamente discussa nell'elaborato e utilizzata per la gestione della memoria, ed è ampiamente utilizzato nelle applicazioni industriali.

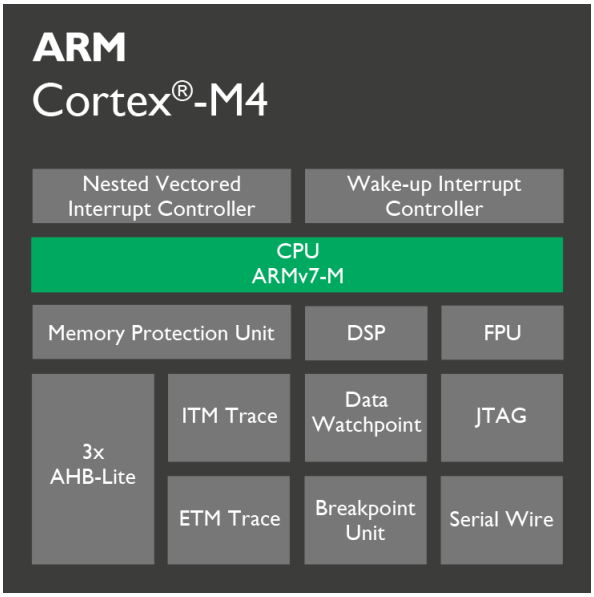

Figura 4.2: Architettura ARM Cortex-M4

Particolare attenzione si pone sulle seguenti periferiche:

- MPU (Memory Protection Unit) presente in figura;
- Acceleratore crittografico AES, non presente nell'architettura ma integrato;
- TRNG (True Random Number Generator), non presente in figura ma integrato.

Il TRNG e l'acceleratore crittografico sono implementazioni hardware di algoritmi che spesso vengono effettuate via software. L'operazione di generazione di numeri random, implementata via software, è una delle operazione che utilizza più risorse poiché impiega diversi algoritmi per la sua esecuzione. Dal momento in cui esiste l'implementazione hardware, l'operazione è sia più veloce sia meno costosa in termini di memoria e calcolo computazionale. Il TRNG non è stato utilizzato all'interno del

progetto sviluppato ma è molto utile qualora si volessero aumentare le prestazioni delle funzionalità sviluppate.

4.2 Sistema Operativo

Avere una buona flessibilità software permette di poter utilizzare le funzionalità di sicurezza sia su un maggior numero di dispositivi sia su un maggior numero di applicazioni. Per rendere il sistema configurabile è doveroso introdurre un sistema operativo cosi da rendere semplice sia la gestione di processi multipli sia l'integrazione di periferiche diverse. Dato che i sistemi IoT si declinano in ambiti differenti ed alcuni di questi hanno dei requisiti specifici sulla disponibilità delle risorse, una caratteristica altrettanto importante è operare su un sistema operativo real-time. Un Real-Time Operating System (RTOS) è un sistema operativo che, fissate delle soglie, riesca a garantire che una risorsa richiesta da un processo sia utilizzabile entro i limiti fissati.
Esistono diversi sistemi operativi real-time per applicazioni IoT[16]. Tra tutti è stata effettuata una scrematura tenendo conto di:

- Memoria utilizzata;
- Supporto tecnico;
- Facilità d'utilizzo;
- Tipologia di licenza;
- Periferiche supportate nativamente.

Utilizzando questi filtri, i sistemi operativi che soddisfano le necessità sottolineate sono: **FreeRTOS, mbed-RTOS**. Dato che FreeRTOS utilizza meno memoria, supporta più periferica in modo nativo e ha un'organizzazione più modulare rispetto a mbed-RTOS, è stato scelto per lo sviluppo del progetto.

FreeRTOS

Il sistema operativo FreeRTOS è sviluppato dal MIT ed è stato utilizzato la prima volta in piattaforme embedded nel 2011. FreeRTOS è un sistema operativo open-source senza nessuna particolare licenza commerciale di utilizzo.
FreeRTOS è un sistema real-time che offre buone prestazioni, meccanismi per la modalità di risparmio energetico del dispositivo, facilità di utilizzo e supporta nativamente diverse architetture e periferiche. Il suo impiego in ambito IoT garantisce la compatibilità con dispositivi basati su differenti architetture e integranti diverse periferiche.
Esistono molte configurazioni possibili di FreeRTOS. Nella configurazione base si presenta con soli tre file sorgente.

- Un modulo per i task, dove vengono definite le primitive da utilizzare per la loro gestione;
- Un'altro modulo, che dipende dal compilatore e dal core utilizzato. Nel sorgente in questione sono implementate le istruzioni specifiche per il tipo di architettura utilizzata.
- Il terzo modulo è utilizzato dallo scheduler per gestire le code in cui sono inseriti i task pronti per essere eseguiti.

Con questa configurazione e con l'utilizzo di 13 task, 2 queue e 4 timer software si stima un memory footprint (occupazione di memoria) minore di 4KB.
La configurazione utilizzata nel progetto è composta da otto file sorgente. In particolare, si utilizza un file che implementa nativamente la gestione dell'MPU. La gestione dei task è effettuata diversamente dal momento in cui è presente l'MPU. All'interno del file sono definite funzioni specifiche da utilizzare per migliorare le prestazioni durante il context-switch. Inoltre, la sincronizzazione e la comunicazione tra i task avviene attraverso l'utilizzo di queue e semafori.

La configurazione scelta utilizza gestione statica della memoria cosi da evitare sovra-allocazione o mancanza di spazio per l'allocazione delle variabili. La configurazione statica della memoria comporta:

- Maggiore facilità della gestione della RAM e della FLASH;

- Resilienza maggiore ad attacchi che utilizzano come vettore le allocazioni di memoria;
- Una stima precisa della massima memoria utilizzata dal device;
- Maggiore controllo sul flusso del codice e sulle variabili utilizzate.

s

Lo Scheduler

Lo scheduler è utilizzato per scegliere quale task deve essere eseguito e per capire quando è necessario un contex-switch. Per avere dei task con priorità diversa è stato configurato lo scheduler con la modalità *preemption*. Ad ogni task viene data una priorità e i task vengono eseguiti secondo questa, nel caso di FreeRTOS è crescente ovvero con 0 si intende priorità di esecuzione minore rispetto a 1. È possibile utilizzare come algoritmo di scheduling la modalità *Round-Robin*, ovvero i task vengono eseguiti per ordine di arrivo. In questo modo è possibile eseguire sempre tutti i task evitando che alcuni non vengano mai serviti. Le primitive di sistema forniscono meccanismi per poter interagire direttamente con lo scheduler, potendo richiedere esplicitamente il context switch e la sospensione dello scheduler.
La configurazione da utilizzare per l'implementazione del software dipende dal sistema utilizzato e dalle necessità dell'applicazione IoT.

Memory Protection Unit

L'MPU dell'ARM Cortex-M4 offre una tabella con otto zone di memoria completamente configurabili. L'MPU offre inoltre due modalità di esecuzione del codice: privilegiata e non privilegiata. Per maggiori informazioni ci si riferisca alla sezione 2.9. I permessi a disposizione di ognuna delle modalità sono i seguenti:

- Read Only
- Read and Write
- Execute never

Per l'utilizzo della periferica MPU, FreeRTOS dispone di un modulo apposito che ne gestisce la configurazione. Per il corretto funzionamento, è necessaria la definizione di alcune variabili utilizzate dal kernel del sistema operativo per la configurazione della periferica. Le variabili da definire sono:

- Indirizzo di inizio e fine della memoria FLASH;
- Indirizzo di inizio e fine della memoria RAM;
- Indirizzo di inizio e fine della memoria che ospita le funzioni privilegiate;
- Indirizzo di inizio e fine della memoria che ospita i dati privilegiati.

FreeRTOS gestisce alcune delle 8 zone in modo autonomo configurate attraverso l'utilizzo di queste variabili. In particolare, l'MPU viene configurata nel seguente modo:

1. La flash ha permessi di sola lettura sia in modalità privilegiata che non
2. La RAM ha permessi di lettura e scrittura sia in modalità privilegiata che non
3. La zona di memoria con funzioni privilegiate ha permessi di sola lettura solo in modalità privilegiata
4. La zona di memoria con dati privilegiati ha permessi di lettura e scrittura solo in modalità privilegiata

Le restanti zone sono configurate dinamicamente ad ogni context-switch. Tra le quattro zone una viene sempre utilizzata dal kernel per rendere lo stack del task accessibile dallo stesso task. Le altre tre zone sono gestite utilizzando le configurazione del task, quindi lo sviluppatore può scegliere come utilizzarle. L'MPU permette anche sovrascrittura dei record quindi se una riga successiva descrive con permessi diversi una zona precedente allora verranno considerati validi solo quelli dell'ultimo record. Questo è utile quando si vuole rendere accessibile una porzione di memoria privilegiata ad un task non privilegiato ad esempio accesso alle periferiche del microprocessore.

Task

I task nella configurazione scelta vengono eseguiti secondo priorità. Quando non è presente nessun task in coda si può scegliere se mettere il dispositivo in sleep oppure far eseguire un task di idle, sfruttabile per operazioni come il backup, analisi memoria, integrità del firmware. Questo task è configurato direttamente da FreeRTOS e le operazioni da eseguire durante l'esecuzione di questo task possono essere definite dall'utente.
Ogni task necessita di una struttura che ha all'interno i parametri per la sua creazione. In particolare, nella configurazione scelta la struttura del task avrà:

- Il nome del task;
- Il puntatore alla funzione che il task dovrà eseguire;
- Dimensione dello stack;
- Parametri della funzione;
- Priorità e privilegio;
- Puntatore ala struttura che rappresenta lo stack;
- Definizioni delle zone di memoria per la configurazione dell'MPU;
- Puntatore alla struttura di controllo del Task chiamata TCB (Task Control Block). Necessario poiché la configurazione prevede l'utilizzo di memoria esclusivamente statica.

Un particolare su cui porre attenzione è lo stack. Esso è fondamentale per l'esecuzione del task poiché tutte le variabili dichiarate all'interno della funzione vengono allocate in questa variabile. FreeRTOS utilizza dei meccanismi per la sostituzione del puntatore allo stack di default con quello del task che deve essere eseguito. Con l'utilizzo dell'MPU, la gestione delle zone di memoria diventa più complessa.
L'MPU gestisce zone di memoria con allineamento in potenze di 2 con un minimo di 32 Byte. Per esempio, se è necessario uno stack di 250 byte, allora esso dovrà essere allineato in 256 byte cioè la potenza di 2 più grande e vicina alla memoria necessaria. Quindi è necessario che lo stack sia allineato in memoria. In questi casi,

l'ambiente di sviluppo mette a disposizione dei metodi per la gestione dell'allineamento delle variabili e della collocazione di dati in determinate porzioni di memoria. Le direttive date vengono utilizzati dal linker per organizzare la memoria in modo da poter essere conforme alle disposizione descritte attraverso il codice.

Strutture dati di sincronizzazione e comunicazione

I mutex servono per garantire l'accesso esclusivo alla sezione critica. L'utilizzo dei mutex all'interno del progetto sviluppato è giustificato dal momento in cui i task necessitano di comunicare e di condividere informazioni. I mutex utilizzati sono implementati attraverso l'utilizzo di semafori binari. La condivisione delle informazioni viene eseguita attraverso l'utilizzo delle queue, integrate nativamente in FreeRTOS. Dato che le queue offrono un meccanismo di blocco qualora un task che cerca di leggere da una coda vuota, sono state utilizzate anche come mezzo di coordinazione tra i task. Il task che attende su una coda vuota si dice essere in stato **bloccato**.

4.3 Libreria Crittografica

Le operazioni crittografiche utilizzano molte risorse computazionali del dispositivo. Definire quale fosse il miglior modo per l'integrazione e la scelta di una libreria crittografica che non usasse molte risorse di sistema ma comunque avesse prestazioni elevate, è stata un'operazione che ha richiesto notevole studio. Inoltre, è necessario utilizzare una libreria che non solo garantisse un memory footprint sostenibile dal sistema ma anche che integrasse meccanismi di gestione della memoria.
Dopo aver analizzato le varie librerie disponibili in rete ne sono state selezionate inizialmente due: mbedTLS e wolfSSL. In seguito, la scelta effettuata si è basata sulla gestione della memoria, sulle prestazioni e sul memory footprint. MbedTLS, sviluppata da ARM, ha delle prestazioni minori e necessita di più memoria rispetto a wolfSSL. Inoltre wolfssl prevede delle configurazioni con memoria totalmente statica. Dato che si vuole mantenere come scelta progettuale la configurazione statica della memoria e date le prestazioni offerte, è stata utilizzata la libreria wolfSSL per l'integrazione di funzioni crittografiche.

WolfSSL è una libreria multi piattaforma, leggera e con alte prestazioni. Questa può essere configurate per utilizzare un ampio numero di hardware sicuri e sistemi operativi. In una fase iniziale è stata utilizzata solo la crypto-engine di wolfSSL, wolfCrypt. Il core crittografico offre molteplici algoritmi di cifratura simmetrica e asimmetrica e algoritmi di hash crittografici.

La configurazione scelta per wolfCrypt prevede l'utilizzo di memoria statica e l'uso di maggiore potenza computazionale degli algoritmi salvaguardando l'utilizzo della memoria. Gli algoritmi scelti inizialmente sono:

- Cifratura simmetrica: AES 256-CBC;
- Cifratura asimmetrica: RSA con chiave lunga 2048 bit;
- Algoritmo di hash crittografico: SHA-256.

Questi non sono gli unici algoritmi utilizzati e configurati all'interno del dispositivo, ma sono usati per l'implementazione del secure boot.

4.4 Protocollo comunicazione sicura

La crypto engine introdotta nella sezione precedente è parte di una libreria più estesa che implementa il protocollo SSL/TLS, wolfSSL.Il protocollo SSL/TLS è utilizzato in tutte le operazioni che necessitano di una connessione sicura. In particolare WolfSSL implementa le versioni TLS v1.2 e TLS v1.3. Dato che la versione 1.3 è poco utilizzata, il dispositivo è stato configurato come client e utilizza TLS 1.2.
Le ciphersuites usate sono:

- TLS_RSA_WITH_AES_256_CBC_SHA256;
- TLS_RSA_WITH_AES_128_CBC_SHA256;
- TLS_RSA_WITH_NULL_SHA256.

La scelta è stata effettuata analizzando le esigenza progettuali per evitare un utilizzo maggiore di memoria. Ciononostante è possibile estendere la lista configurando ulteriori ciphersuites.

4.5 Organizzazione memoria

Dato che il device utilizzato non integra Hardware Sicuri è necessario ricorrere a degli stratagemmi per impiegare l'MPU come strumento per la protezione delle informazioni sensibili. Ogni microcontrollore utilizza degli indirizzi per la mappatura delle periferiche. Gli indirizzi adoperati cambiano al variare del tipo di dispositivo utilizzato. Per mantenere la descrizione più generale possibile e per poter implementare il firmware per device con una mappatura degli indirizzi diversa, ci si riferirà alle aree di memoria con il nome del suo contenuto. Per l'implementazione delle funzionalità definite si ripartisce la memoria nelle aree seguenti:

- Flash;
- RAM;
- Periferiche interne come Real Time Clock (RTC), Nested Vectored Interrupt Controller (NVIC);
- Periferiche esterne come EEPROM, HTA, Can-Bus.

La flash deve essere ulteriormente suddivisa cosi da poter rendere facilmente comprensibile quale logica è stata utilizzata per l'implementazione delle funzionalità discusse. Inoltre, potrebbe essere utile per sviluppi futuri del secure boot nella modalità Just-In-Time. In particolare, la zona codice è stato suddiviso nelle seguenti sezioni:

- Secure Boot;
- Bootloader;
- Kernel;
- Applicazioni.

Secure Boot

La prima sezione è quella che si occupa del secure boot e della memorizzazione di dati critici come certificati digitali, chiavi private e funzioni crittografiche. Il secure

boot deve avere accesso, oltre alle informazioni e alle funzioni della propria regione di memoria, al codice che deve essere eseguito dal dispositivo. Questa sezione, teoricamente collocata in una zona di memoria protetta dove l'integrità è garantita per costruzione del dispositivo, nell'implementazione sviluppata è posta nella prima parte di memoria. Questa scelta non garantisce l'integrità del dispositivo in caso di manomissione fisica, ma attraverso l'utilizzo dell'MPU è possibile proteggere questa zona a livello hardware.

L'integrità è garantita finché l'MPU è attiva, cioè solo se il dispositivo è in stato di esecuzione. Quindi è ancora possibile manomettere le informazioni presenti al suo interno attraverso manomissione fisica della memoria flash. L'utilizzo dell'MPU per la protezione della memoria richiede la creazione di alcune direttive per il linker che imponga la collocazione di questi componenti nella parte di memoria scelta.

Bootloader

La seconda sezione è il Bootloader che si occupa della preparazione del dispositivo. Questo implica due diverse opzioni. La prima è l'avvio normale del device, che si occupa di configurare tutte le funzionalità e periferiche che verranno utilizzate nella sua normale esecuzione. La seconda opzione fa si che il dispositivo avvii una MODALITà da utilizzare per l'installazione di un nuovo firmware in memoria qualora la verifica effettuata dal secure boot fallisse.

Kernel

L'utilizzo di un sistema operativo implica la presenza di un kernel, il quale gestisce le primitive di sistema. L'accesso a quest'ultimo deve essere limitato e controllato. Per poter utilizzare l'MPU come dispositivo di gestione degli accessi in memoria, è necessario definire una porzione dove possano essere memorizzate le informazioni e le funzioni che il sistema operativo classifica come PRIVILEGED DATA e PRIVILEGED FUNCTION ovvero quelle informazioni quelle che rappresentano le primitive di sistema e che appartengono strettamente al kernel. Con PRIVILEGED si intende la memoria non accessibile dai task che non possiedono permessi privilegiati. Nella soluzione proposta fanno parte di questa categoria:

- Secure Boot;
- Primitive di sistema;
- Certificati a chiave pubblica;
- Chiave privata del dispositivo;
- Funzioni crittografiche;
- Periferiche di sistema.

In particolare, i dati e la logica definite come privileged verranno rese fruibili alle applicazioni, seppur in modo limitato, attraverso meccanismi sia software che hardware .

Sezione Applicativa

La quarta sezione è dedicata al codice utente. I task definiti nella zona di memoria in questione necessitano di un controllo più stringente sull'accesso alla memoria nonché di meccanismi particolari sia per l'accesso alle periferiche di sistema che per l'utilizzo delle primitive di FreeRTOS. All'interno della sezione applicativa dovranno essere inserite tutte le funzionalità che utilizzano come sorgente dati il mondo esterno, come l'Authenticated Firmware Update, e tutte le applicazioni basate sul sistema operativo.

4.6 Secure Boot

Dopo aver definito il posizionamento in memoria e le caratteristiche del secure boot, in questa sezione viene descritto nel dettaglio la sua implementazione.
Il secure boot viene eseguito all'avvio e quindi non prevede l'utilizzo del sistema operativo. L'integrazione all'interno del dispositivo è indipendente dal sistema operativo utilizzato e dalle applicazioni in cui il device è usato. Inoltre, in questa sezione si descrivono i processi sviluppati per la generazione dei dati complementari alla funzionalità in analisi. In particolare si descrivono i processi per la creazione delle chiavi e i certificati digitali associati e i tools sviluppati per la generazione della

firma digitale del firmware. Attraverso il loro impiego è possibile automatizzare le operazioni da compiere sul codice sorgente per poi essere installato sul device.

Verifica del firmware

Per l'implementazione del secure boot è stato necessario comprendere cosa potesse compromettere la buona riuscita dell'operazione. L'analisi effettuata è stata necessaria poiché durante lo sviluppo dei requisiti non si è tenuto conto dei meccanismi che il dispositivo utilizza per la gestione delle periferiche.
Qualora un interrupt venisse scatenato mentre il secure boot è in esecuzione, la routine di gestione dell'interrupt potrebbe compromettere la buona riuscita dell'operazione. Dato che la soluzione è stata pensata per essere la base su cui creare applicazioni IoT si deve prevedere che l'utilizzatore possa implementare routine che utilizzino la memoria sia in lettura che in scrittura. Se la routine descritta venisse eseguita durante il secure boot, la verifica potrebbe essere falsato. Quindi è necessario implementare il secure boot come operazione atomica, disattivando tutto ciò che potrebbe compromettere la sua esecuzione.

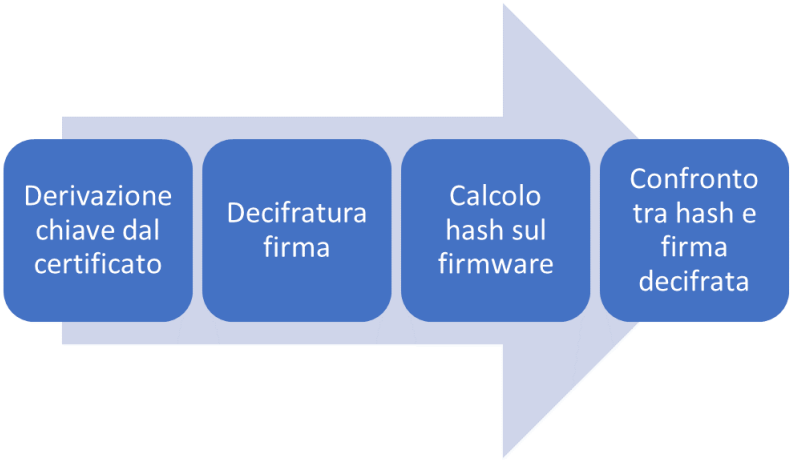

Figura 4.3: Architettura Secure Boot

Come rappresentato dalla figura 4.3, una volta aver disattivato gli interrupt, vengono eseguite le operazioni descritte di seguito:

1. Il parse del certificato digitale è utilizzato per la verifica della validità del certificato e derivazione della chiave pubblica;

2. La decifratura della firma digitale presente in memoria viene effettuata attraverso l'utilizzo dell'algoritmo RSA. La chiave utilizzata, lunga 2048 bits, è ricavata dal certificato digitale presente in memoria;

3. Il calcolo del digest è basato su algoritmo di hash crittografico SHA-256 ed è calcolato sul firmware presente in memoria dalla sezione kernel fino a quella applicativa;

4. Il confronto tra il valore calcolato e quello decifrato è propedeutico per l'avvio del dispositivo. Qualora la verifica fallisse, il device entrerà in modalità di *recovery* dove è necessaria l'installazione di un nuovo firmware integro.

Questo primo livello di sicurezza offre una buona resistenza agli attacchi di manomissione del firmware. L'algoritmo può essere riutilizzato nell'Authenticated Firmware Update per verificare che il firmware ricevuto non sia stato manomesso.

Processo generazioni chiavi, certificati e firma digitale

Per derivare dalla firma digitale il digest appartenente al firmware, è necessario che il device abbia in memoria il certificato digitale dell'entità che ha firmato i dati presenti in memoria. Dal certificato si ricava il periodo di validità e la chiave pubblica. Quest'ultima verrà impiegata dal dispositivo per l'utilizzo di RSA.
Per la generazione delle chiavi e del certificato digitale è stato utilizzato openSSL. Questo tool integra molti algoritmi e protocolli di sicurezza e attraverso il suo utilizzo è stato possibile automatizzare i processi di generazione di chiavi, certificato e firma. OpenSSL è stato utilizzato all'interno di uno script che modifica il firmware da installare sul dispositivo, inserendo il certificato digitale o la firma digitale. Per la generazione della firma digitale viene utilizzato l'algoritmo SHA256 con una chiave RSA lunga 2048 bits, lunghezza minima consigliata per applicazioni non critiche.

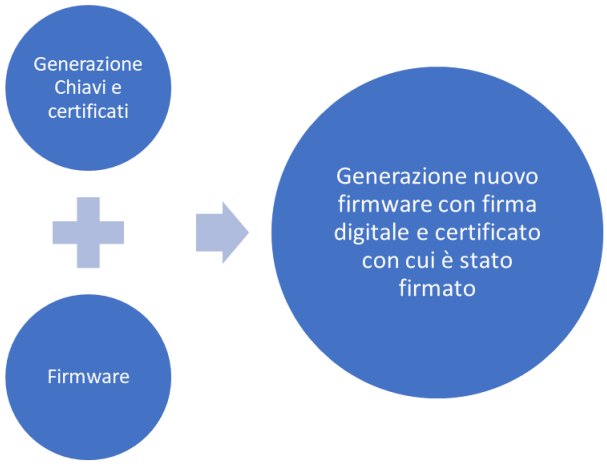

Figura 4.4: Script per la firma del firmware

Lo script descritto dalla figura 4.4 richiede in ingresso almeno il file .hex generato dall'ambiente di sviluppo a seguito della compilazione. Questo file descrive come vengo mappati i dati e le funzioni in memoria. Lo script utilizza il file per leggere le informazioni e calcolare la firma partendo dalla sezione kernel. L'indirizzo da cui partire per il calcolo della firma, per semplicità, è costante all'interno dallo script. Qualora fosse necessario modificare l'indirizzamento è indispensabile correggere l'indirizzo. L'utility creata è modulare e i componenti possono essere utilizzati separatamente. Infatti, si può eseguire esclusivamente la generazione della firma fornendo in ingresso allo script il parametro **-f**.Il tool inserisce i dati generati all'interno binario dato in ingresso.

4.7 Authenticated Firmware Update

Il protocollo SSL/TLS impone l'utilizzo di chiavi, certificati digitali e un protocollo di livello trasporto affidabile, come TCP. Nell'implementazione i certificati digitali impiegati sono due: uno appartenente al dispositivo, cosi da potersi autenticare nei confronti del server, e un altro appartenente alla CA fidata che è incaricata ad emettere certificati autorizzati all'aggiornamento del firmware. Per semplicità il certificato presente in memoria e utilizzato per il secure boot coincide con quello della CA appena citata. Inoltre per verificare il funzionamento dell'algoritmo implementato sul dispositivo è stato necessario creare un'applicazione che emulasse un server SSL/TLS atto ad effettuare l'aggiornamento del firmware.

Comunicazione sicura

Il modulo di comunicazione che utilizza SSL/TLS è implementato all'interno di un task indipendente dalla funzionalità di autheticated firmware update cosi da rendere indipendente la connessione sicura. Per poter usufruire della connessione è necessario che il dispositivo sia entrato nella modalità di funzionamento normale, ovvero deve aver effettuato la verifica del firmware, inizializzato le periferiche necessarie e avviato lo scheduler. Il task designato alla comunicazione sicura impiega le librerie implementate sul dispositivo per l'avvio della comunicazione. Sono state configurate una serie di ciphersuites basate sopratutto su RSA, AES, SHAx. Questo non esclude l'aggiunta di ulteriori algoritmi.

Dopo aver autenticato la sorgente, negoziato la ciphersuite e avviato la comunicazione, il task di connessione sicura riceve i dati. I dati sono resi disponibili ad altre operazioni attraverso l'utilizzo di queue. Per garantire un maggiore isolamento tra i task è stata utilizzata una queue per ciascuna applicazione che utilizza la comunicazione sicura. Nel firmware sviluppato l'unico task che ricorre al protocollo SSL/TLS è quello che si occupa dell'aggiornamento sicuro del firmware, di conseguenza è stata impiegata un'unica queue per la comunicazione dei task.
Dato che non è stato possibile configurare un protocollo di rete su cui basarsi, wolfSSL è stato configurato per ricevere e inviare dati con delle funzioni appositamente create, basate su un protocollo seriale, l'UART.

Prima di avviare la sessione, durante il protocollo di handshake, il task si preoccupa di verificare l'identità del server per autenticarne l'accesso. Una volta avviata la comunicazione, il task riceve i dati inviati dal server nella forma TLV (Type Length Value) così da poter effettuare dei check sui dati ricevuti. Se il tipo di pacchetto ricevuto corrisponde all'aggiornamento del firmware, i dati vengono memorizzati in una flash esterna finché non saranno verificati e installati sul dispositivo. Terminata la ricezione dei dati, viene notificato, al task che si occupa dell'aggiornamento del firmware, la terminazione della comunicazione. È stata utilizzata la forma TLV cosi da poter rendere versatile la soluzione qualora fossero presenti altre applicazioni connesse.

Processo di verifica e dell'installazione dell'aggiornamento

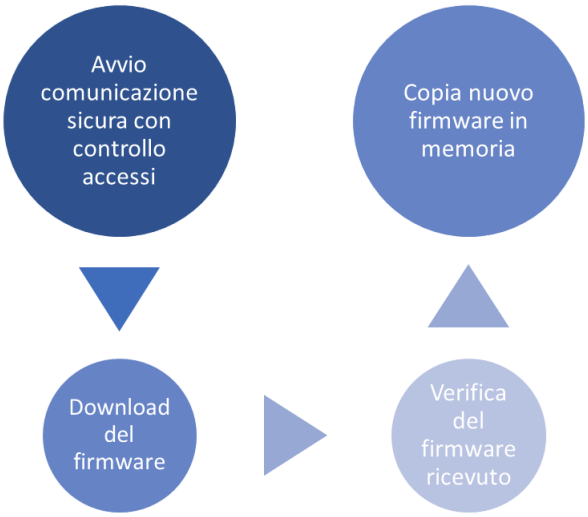

Figura 4.5: Architettura Authenticated Firmware Update

Il task di **authenticated firmware update** attende la terminazione della comunicazione aspettando che un dato sia immesso in una queue. Una volta terminata la connessione e viene notificata l'avvenuta ricezione del firmware attraverso l'inserimento di un valore nella coda che definisce lo stato di terminazione della comunicazione. Il task, in stato di attesa su una coda vuota, riceve l'informazione dalla coda e inizia con la verifica del firmware memorizzato nella flash esterna. Per necessità l'indirizzo di inizio del firmware memorizzato nella memoria esterna è fissato dalle configurazioni di sistema.
Per la verifica del firmware da installare viene utilizzato lo stesso meccanismo del secure boot. La funzione utilizzata è la medesima impiegata all'avvio per il secure boot ma in questo caso si utilizzano indirizzi diversi per poter far riferimento alla memoria esterna. In questo caso, come per la comunicazione sicura, si devono elevare i privilegi per l'utilizzo della funzione di verifica e dei dati necessari al suo funzionamento. La chiamata delle funzioni privilegiate viene gestita dal sistema operative. Una volta verificata l'integrità del firmware con le modalità descritte per il secure boot, viene effettuato l'aggiornamento del firmware spostando il contenuto

della memoria esterna nella memoria flash.

Processi off-board

Come detto precedentemente, è necessario un server per verificare che il codice implementato funzioni correttamente. Per stabilire una connessione client-server, oltre al supporto di openSSL, è stato creato un tool atto a ricevere ed inviare su seriale dati ricevuti da openSSL e dal dispositivo. OpenSSL non supporta altri tipi di trasmissione se non il protocollo TCP/IP che nell'ambiente utilizzato utilizza la scheda di rete per la trasmissione e ricezione dei dati. Il tool deve essere un mezzo per poter trasferire queste informazioni dalla scheda di rete alla porta seriale. Il programma riceve le informazioni attraverso un socket per poi trasferirle dispositivo utilizzando il protocollo UART. Le informazioni inviate dal device, vengono ricevute e inoltrate sul socket utilizzato per la ricezione. Il programma creato, poi, viene integrato all'interno di uno script che esegue le seguenti operazioni:

1. Firmare il firmware da inviare, con il tool descritto nella sezione riguardante il secure boot;
2. Avviare openSSL come server;
3. Avviare il programma che trasferisce da scheda di rete a seriale e viceversa;
4. Instaurare la connessione sicura;
5. Trasferire il firmware firmato.

Capitolo 5

Conclusioni

Attraverso lo sviluppo di questo progetto si è voluto contribuire alla realizzazione di un'architettura software con funzionalità di sicurezza per sistemi IoT, in modo tale da poter essere utilizzata come base per l'implementazione di applicazioni connesse aumentandone la sicurezza e la resilienza agli attacchi. In particolare la tesi si è incentrata sullo sviluppo di funzionalità per la protezione del firmware e della memoria al fine di evitare la manomissione o la sottrazione di dati sensibili. Le funzionalità sviluppate sono: **Secure Boot** e **Authenticated Firmware Update**. Il secure boot si occupa di accertarsi che il firmware da eseguire non sia stato manomesso, mentre l'authenticated firmware update è utilizzato per l'aggiornamento del firmware da remoto garantendone l'integrità e la confidenzialità.

Attualmente esistono alcune tecnologie e metodologie di progettazione atte ad aumentare la sicurezza dei device IoT. In particolare la metodologia adottata utilizza una tabella di valutazione del rischio al fine di determinare le vulnerabilità più critiche.

La maggior parte delle vulnerabilità sono dovute alla cattiva gestione della memoria e alla mancanza di forti meccanismi di cifratura o autenticazione [6]. Queste vulnerabilità possono introdurre codice eseguibile in memoria alterando il normale funzionamento del dispositivo. Dato che questo progetto di tesi è stato pensato per essere utilizzato anche in ambiti dove la safety è di primaria importanza, è necessario implementare delle funzionalità che verifichino l'integrità del firmware e che proteggano la memoria del dispositivo da eventuali accessi anomali al fine di evitare

comportamenti inusuali.

Gli attacchi a questi dispositivi possono essere condotti in due modi diversi: attraverso le periferiche di comunicazione del dispositivo o condotti attraverso accesso fisico al device. In questo progetto sono stati presi in esame soltanto gli attacchi attraverso le periferiche di comunicazione poiché le vulnerabilità più rilevanti possono essere risolte attraverso soluzioni software [6].

Dal sottoinsieme di rischi selezionati emerge che è necessario proteggere:

- Le informazioni scambiate attraverso la rete;
- La memoria;
- Il firmware.

Si è preferito utilizzare una soluzione software per la protezione del firmware così da rendere la soluzione versatile e utilizzabile su diversi dispositivi. La logica di verifica del firmware è stata progettata in modo tale da ottenere un basso impatto sul sistema (in termini di memoria e di calcolo computazionale) dato che questa tipologia di dispositivi hanno risorse limitate. In particolare, è stata utilizzata la firma digitale. Quest'ultima è associata al firmware presente sul dispositivo per garantirne l'integrità.

Il secure boot proposto utilizza la firma digitale e il certificato digitale, precedentemente caricati sul dispositivo, per poter effettuare la verifica sul firmware. In particolare la funzionalità sviluppata effettua le operazioni seguenti:

1. Disattivazione degli interrupt per rendere l'operazione atomica;
2. Analisi del certificato in memoria per verificarne la validità e per ricavarne la chiave pubblica;
3. Decifratura della firma digitale, ottenuta con gli algoritmi SHA-256 e RSA con chiave di lunghezza 2048 bit ricavata dal passo precedente;
4. Calcolo dell'hash del firmware presente in memoria attraverso l'utilizzo dell'algoritmo SHA-256;
5. Confronto tra valore calcolato e valore decifrato. Se i valori corrispondono si prosegue con il normale funzionamento del dispositivo.

L'authenticated firmware update è utilizzato per effettuare l'aggiornamento del firmware da remoto. Questa funzionalità è supportata da un sistema operativo [17] e da una libreria che implementa sia funzioni crittografiche sia il protocollo SSL/TLS [18]. Questi due componenti supportano diverse architetture e periferiche. In particolare:

- Per garantire protezione alla memoria, FreeRTOS è stato configurato con l'integrazione dell'MPU;
- A causa della mancanza di una periferica di rete non è stato ritenuto necessario implementare il protocollo TCP/IP e quindi wolfSSL è stato configurato per essere utilizzato con il protocollo UART.

Terminata la fase di mutua autenticazione vengono scaricate le informazioni relative all'aggiornamento del firmware. Una volta scaricato il nuovo firmware, questo viene verificato e installato. La verifica del firmware è stata implementata secondo le modalità descritte per il secure boot. I benefici derivanti dalla soluzione sviluppata sono diversi:

- **Basso impatto sul sistema in termini di memoria.** Le configurazioni delle librerie sono state scelte in modo da occupare meno memoria possibile;
- **Integrazione su diversi dispositivi o architetture.** Attraverso le il sistema operativo utilizzato è facile adattare la soluzione proposta su una maggiore tipologia di dispositivi;
- **Firmware sicuro grazie a meccanismi forti di verifica.** La verifica basata su firma digitale generata attraverso gli algoritmi RSA e SHA-256 garantisce l'anti-manomissione del firmware;
- **Cifratura dei dati in transito attraverso l'integrazione del protocollo SSL/TLS.** La connessione sicura garantisce un livello di protezione elevato attraverso l'utilizzo di ciphersuite basate su AES, SHA e RSA;
- **Protezione della memoria attraverso l'impiego dell'MPU.** Attraverso la configurazione utilizzata diventa difficoltoso accedere ai dati in memoria attraverso buffer overflow o buffer over-read;

- **Meccanismi forti di autenticazione attraverso SSL/TLS**. La mutua autenticazione effettuata con SSL/TLS garantisce ad entrambe le parti di scambiare informazioni con fonti autenticate;

Questa soluzione può essere facilmente implementata in device IoT per via del memory footprint limitato, aumentando sensibilmente il livello di sicurezza. Inoltre il lavoro svolto riesce a risolvere più della metà delle *Top 10 IoT vulnerabilities* [6].

Gli sviluppi futuri per migliorare la soluzione saranno:

- Integrazione un Hardware Trust Anchor (HTA). In questo caso è necessario trasferire le informazioni sensibili nell'HTA, così da garantire l'anti-manomissione delle informazioni. Con informazioni sensibili si intendono certificati, funzioni crittografiche e chiavi pubbliche o private;
- Integrazione di una periferica di comunicazione, il protocollo IP e il protocollo TCP cosi da connettere il dispositivo ad una rete con accesso Internet;
- Testing attraverso l'utilizzo di tool per la verifica statica del codice sorgente;
- Testing attraverso l'emulazione di attacchi noti.

In conclusione, la tematica affrontata per lo svolgimento della tesi è molto recente e ha dato la possibilità di portare un vero contribuito a quest'area dell'informatica. Grazie a quest'esperienza è stato possibile approfondire alcune metodologie di analisi e di progettazione sicura per questa tipologia di prodotti.

Bibliografia

[1] K. Zhao, L. Ge in Computational Intelligence and Security (CIS), 2013 9th International Conference on, IEEE, **2013**, pp. 663–667.

[2] R. Ratasuk, B. Vejlgaard, N. Mangalvedhe, A. Ghosh in Wireless Communications and Networking Conference (WCNC), 2016 IEEE, IEEE, **2016**, pp. 1–5.

[3] J. Pacheco, S. Hariri in 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W), **2016**, pp. 242–247.

[4] report panorama degli attacchi informatici nel 2017 fino ad oggi, `https://it.safeandsavvy.f-secure.com/2017/08/29/report-panorama-degli-attacchi-informatici-nel-2017-fino-ad-oggi/`.

[5] Remote Exploitation of an Unaltered Passenger Vehicle, **2015**, `http://illmatics.com/Remote%20Car%20Hacking.pdf`.

[6] Top 10 IoT vulnerabilities, **2017**, `https://owasp.org/index.php/Top_IoT_Vulnerabilities`.

[7] M. Abomhara, G. M. Køien, *Journal of Cyber Security* **2015**, *4*, 65–88.

[8] P. Flood, M. Schukat in The 10th International Conference on Digital Technologies 2014, **2014**, pp. 68–72.

[9] Network service, `https://en.wikipedia.org/wiki/Network_service`.

[10] Knowing-Plaintext attack, `https://en.wikipedia.org/wiki/Known-plaintext_attack`.

[11] Key Strength, `https://keylength.com/en/4/`.

[12] RSA, http://www.html.it/pag/16477/la-crittografia-a-chiave-pubblica-e-rsa.

[13] Public Key Infastructure, https://en.wikipedia.org/wiki/Public_key_infrastructure.

[14] Code Obfuscation, https://en.wikipedia.org/wiki/Obfuscation_(software).

[15] Cybersecurity Best Practices for Modern Vehicle, **2016**, https://www.nhtsa.gov/staticfiles/nvs/\pdf/812333_CybersecurityForModernVehicles.pdf.

[16] Comparison of real -time operating systems, https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems.

[17] FreeRTOS Operating System, **2017**, https://freertos.org.

[18] WolfSSL CryptoLibrary, **2017**, https://wolfssl.com.