UNIVERSITÀ DI PARMA

Dipartimento di Ingegneria e Architettura

Corso di Laurea in Ingegneria Informatica Elettronica e delle Telecomunicazioni

Aggiornamento delle Root Key per Applicazioni IoT Basate su LoRaWAN

Root Keys Update for LoRaWAN-Based IoT Applications

Relatore:
Chiar.mo Prof. Michele Amoretti

Correlatore:
Prof. Luca Veltri

Tesi di Laurea di:
Alessandro Pindozzi

ANNO ACCADEMICO 2020-2021

Ringraziamenti

Vorrei dire “Grazie a tutti” e basta, sarebbe più nel mio stile, ma mi sa che per questa volta forse sarebbe meglio buttare giù due righe.

Innanzitutto mi sembra doveroso ringraziare i miei genitori, per il sostegno, l’affetto e i finanziamenti, soprattutto i finanziamenti. Mia sorella perchè non si offende se non la chiamo, nonna Ada e nonna Giovanna, che nel bene e nel male sono sempre le nonne.

Ringrazio i miei cugini e in particolare mio cugino Francesco, che c’è, ed è sempre pronto ad aiutare, magari con metodi poco ortodossi ma che servono a crescere. Quasi dimenticavo: ringrazio la mia cugina acquisita Gaia, sempre pronta ad ascoltarmi e tirarmi su se serve.

Ringrazio Mattia e Gabriele, perché dalle superiori, tutti i giorni, siamo sempre stati noi tre, a prescindere da tutto.

Ringrazio Pierluigi, con cui sono cresciuto, e continuerò a crescere, tra una chiamata e l’altra.

Ringrazio Gaia T. per essermi stata sempre vicina, soprattutto nei momenti difficili e Aurora, che anche se non risponde al telefono è sempre disponibile (non avrei trovato il completo se non fosse stato per lei).

Infine, non posso che ringraziare i miei compagni di Università. Grazie a Martina, per l’ansia che ci portava quotidianamente e che ci faceva studiare di più, e sopratutto per la piscina. Grazie a Matteo per tutti i suoi aiuti e le canzoni in macchina ogni volta che passava a prendermi. Grazie a Giorgia per la voglia di sentire tutte le mie storie ogni mattina. Grazie a Orazio che è stata la prima persona che ho conosciuto qui, e senza la quale non sarei durato

due mesi, per la sua disponibiltà, simpatia e leggerezza. Grazie a Diletta per avermi sempre sopportato, ma aspetterei a parlare perchè negli anni a venire sarà ancora facile sfiorare l'esaurimento. E grazie a Saverio, per la pazienza ogni qualvolta avevo bisogno di una mano, per la sua collaborazione e per tutti gli allenamenti fatti insieme anche con 0°. Quindi grazie a tutti, per aver reso memorabili questi tre anni, e anche perchè credo, che senza di voi sarebbero stati almeno quattro o cinque. Grazie per la vostra amicizia... e per tutti i passaggi.

Quindi solo ora posso concludere dicendo: grazie a tutti.

Indice

Introduzione **1**

1 Stato dell'Arte **3**

1.1 LoRaWAN 3

1.1.1 LoRa 3

1.1.2 LoRaWAN 5

1.1.3 LoRaWAN: End Device Activation 7

1.1.4 LoRaWAN: Security 11

1.1.5 LoRaWAN: Vulnerabilità 13

1.1.6 LoRaWAN: Possibili Attacchi 15

1.2 The Things Stack 18

1.2.1 TTS 18

2 Architettura Funzionale del Sistema Realizzato **20**

2.1 Nodi della Rete 20

2.2 API 22

2.3 Descrizione dell'Algoritmo 24

2.3.1 Generazione di chiavi 24

2.3.2 Richiesta PUT per l'aggiornamento 26

2.3.3 Re-join 27

2.3.4 Sicurezza 28

3 Implementazione **29**

3.1 Registrazione dei Nodi nella Rete 29

3.2 Algoritmo per l'Aggiornamento dell'AppKey 32
3.2.1 Richieste HTTP 32
3.2.2 Generazione delle Chiavi 34
3.2.3 Aggiornamento dellAppKey dell'ED 35
3.3 Re-Join 39

4 Valutazione Sperimentale **40**
4.1 Test Effettuati 40
4.2 Problematiche di Re-Join 42
4.3 Incremento del Livello di Sicurezza 44

Conclusioni **46**

Bibliografia **48**

Introduzione

Per comprendere l'argomento trattato in questa Tesi è necessario prima spiegare brevemente cosa si intende quando si parla di IoT (Internet of Things). Alla base di Internet of Things, che appunto viene tradotto con Internet delle Cose, ci sono i cosiddetti oggetti intelligenti.

Non si parla di computer, tablet e smartphone, bensì di tutti gli oggetti che ci circondano nelle attività quotidiane. In altre parole l'IoT si riferisce all'interconnessione e allo scambio di dati tra dispositivi/sensori. Attualmente, con la crescita esplosiva delle tecnologie IoT, è possibile trovare un numero crescente di applicazioni pratiche in molti campi, tra cui sicurezza, monitoraggio delle risorse, agricoltura, misurazione, città e case intelligenti [1].

Gli aspetti principali che bisogna considerare quando si parla di IoT sono: comunicazioni a lungo raggio, bassa velocità di trasmissione dei dati e basso consumo energetico, a vantaggio di una riduzione dei costi. Le tecnologie a corto raggio (come il Bluetooth) non sono adatte per le comunicazioni di questo tipo di dispositivi, mentre quelle troppo a lungo raggio (come il 3G) comportano uno spreco di energia eccessivo.

Per questi motivi è nata una nuova branca della comunicazione wireless: LPWAN (Low Power Wide Area Network). Le reti LPWAN permettono di coprire vaste aree, limitando il consumo energetico. Tra questo tipo di reti, una tra le più promettenti è LoRa [2].

LoRa, sviluppato da Semtech Inc., è lo strato fisico utilizzato per creare il collegamento di comunicazione a lungo raggio. Fa uso di una particolare

tecnica di modulazione, chiamata CSS (Chrip Spread Spectrum), la quale è stata utilizzata nelle comunicazioni militari e spaziali per decenni a causa delle lunghe distanze di comunicazione che possono essere raggiunte e della robustezza alle interferenze, ma LoRa è la prima implementazione a basso costo per uso commerciale [3].

LoRaWAN definisce il protocollo di comunicazione e l'architettura di sistema per la rete, mentre il livello fisico consente il collegamento di comunicazione a lungo raggio. Il protocollo e l'architettura di rete hanno maggiore influenza nel determinare la durata della batteria di un nodo, la capacitÃ della rete, la qualità del servizio, la sicurezza e la varietà di applicazioni servite dalla rete [3].

Trattandosi di sistemi IoT, di cui si fa uso tutti i giorni e da cui il lavoro e in generale la vita delle persone dipendono sempre in misura maggiore, è importante che tutti siano sicuri e a prova di attacchi informatici.

In questa Tesi vengono prima spiegati quali sono i principali aspetti che caratterizzano LoRa e LoRaWAN, quindi si può comprendere meglio come è strutturata una rete LoRaWAN, quali sono i nodi che ne prendono parte e come interagiscono. Alla luce di ciò, è possibile comprendere più facilmente in che maniera operano i meccanismi di sicurezza della rete, e di conseguenza quali sono le vulnerabilità. Infine, un excursus sui possibili attacchi informatici che sfruttano tali vulnerabilità conclude il primo Capitolo. Il secondo e il terzo sono incentrati sulla descrizione di come funziona e come è implementato il meccanismo elaborato come progetto di Tesi, volto a migliorare uno degli aspetti critici della sicurezza di una rete LoRaWAN. Più precisamente il Capitolo 2 è incentrato sull'architettura del sistema realizzato, così che il lettore possa capire come questo operi, mentre nel Capitolo 3 si scende più nel dettaglio descrivendone l'implementazione, quindi da quali entità è formato e come esse lavorano e interagiscono tra loro. Nell'ultimo Capitolo vengono infine descritti i test effettuati e i risultati ottenuti, lasciando posto anche a descrizioni e osservazioni sui problemi riscontrati in fase di sviluppo.

Capitolo 1

Stato dell'Arte

1.1 LoRaWAN

1.1.1 LoRa

LoRa (Long Range) [2] è stato sviluppato da Semtech Inc. come livello fisico proprietario per fornire comunicazioni low-power e long-distance utilizzando una tecnologia di comunicazione a spettro espanso. Inoltre fa uso di bande ISM senza licenza, ovvero 868 MHz in Europa, 915 MHz in Nord America e 433 MHz in Asia. La comunicazione bidirezionale è fornita da una specifica tecnica di modulazione detta Chirp Spread Spectrum (CSS), mostrato nella Fig. 1.1, che diffonde un segnale a banda stretta su una larghezza di banda del canale più ampia.

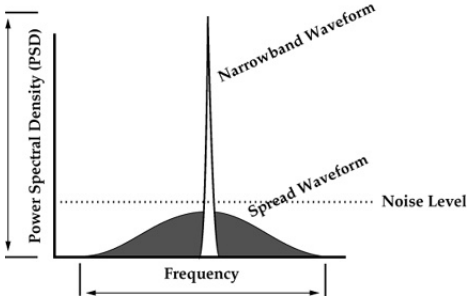

Figura 1.1: Modulazione Spread-Spectrum

Operando quindi su un canale a larghezza di banda fissa (125 KHz o 500 KHz per i canali uplink e 500 KHz per i canali downlink), LoRa offre un compromesso tra sensibilità e velocità dei dati [4]. Inoltre, il segnale risultante dalla modulazione ha bassi livelli di rumore, consentendo un'elevata resilienza alle interferenze.

Un altro importante aspetto da sottolineare è l'utilizzo di spreading factors (fattori di diffusione) ortogonali, i quali indicano i simboli modulati per unità di tempo. In particolare LoRa utilizza sei fattori di diffusione (da SF7 a SF12) per adattare la velocità dei dati sviluppando quindi un meccanismo che viene definito "Adaptive Data Rate mechanism". Un fattore di diffusione più elevato consente un raggio più lungo a scapito di una velocità di trasmissione dati inferiore e viceversa. La velocità dei dati LoRa è compresa tra 300 bps e 50 kbps a seconda del fattore di diffusione e della larghezza di banda del canale [1]. L'utilizzo di questi fattori, quindi, consente alla rete di preservare la durata della batteria dei nodi finali collegati effettuando ottimizzazioni adattive dei livelli di potenza e delle velocità di trasmissione dati di un singolo nodo finale. Per fare un esempio, si immagini un dispositivo finale situato vicino a un gateway. Questo dovrebbe trasmettere dati con un fattore di diffusione basso, poichè è necessario un budget di collegamento molto ridotto. Tuttavia, un dispositivo finale situato a diverse miglia da un gateway deve trasmettere con un fattore di diffusione più elevato, il quale fornisce un maggiore guadagno di elaborazione e una maggiore sensibiltà di ricezione, sebbene la velocità dei dati sarà necessariamente inferiore [4]. Infine, l'utilizzo di diversi fattori di diffusione permette alle stazioni base LoRa di ricevere messaggi contemporaneamente.

Per quanto riguarda LoRa, bisogna anche considerare le restrizioni imposte per ciò che concerne il duty-cylce e il time-on-air. Sono restrizioni che variano a seconda della regione, imposte dalle autorità locali, e devono essere rispettate sia dai device che dai gateway. Oltre alle limitazioni legali appena citate, ci sono delle limitazioni che vengono imposte anche dagli operatori di rete [5]. Le caratteristiche di modulazione LoRa per ciascuna regione sono

definite nel documento LoRaWAN Regional Parameters, disponibile presso LoRa Alliance [4].

1.1.2 LoRaWAN

LoRaWAN è il protocollo di livello superiore (MAC, rete e livelli superiori) basato sul LoRa (livello fisico), in cui vengono definiti in dettaglio il funzionamento e la struttura dell'intero sistema [2]. LoRaWAN è stato standardizzato da LoRa-Alliance e la prima versione risale al 2015.

Molte reti implementate esistenti utilizzano un'architettura di rete mesh. In una rete mesh, i singoli nodi finali inoltrano le informazioni di altri nodi per aumentare il raggio di comunicazione e le dimensioni delle celle della rete. Sebbene ciò aumenti la portata, aggiunge anche complessità, riduce la capacità della rete e riduce la durata della batteria poichè i nodi ricevono e inoltrano informazioni da altri nodi che sono probabilmente irrilevanti per loro.

L'architettura a stella a lungo raggio ha più senso per preservare la durata delle batterie quando è possibile ottenere la connettività a lungo raggio [3].

In una rete LoRaWAN, mostrata nella Fig. 1.2, i nodi non sono associati a un gateway specifico. Invece, i dati trasmessi da un nodo vengono generalmente ricevuti da più gateway. Ogni gateway inoltra il pacchetto ricevuto dal nodo finale al server di rete basato su cloud tramite un backhaul (cellulare, Ethernet, satellite o Wi-Fi). Quindi ogni messaggio trasmesso da un dispositivo finale viene ricevuto da tutte le stazioni base nel range. Sfruttando questa ricezione ridondante, LoRaWAN migliora il rapporto dei messaggi ricevuti con successo. Tuttavia, per ottenere questa funzionalità sono necessarie più stazioni base nelle vicinanze, il che può aumentare i costi di implementazione della rete. Le ricezioni duplicate risultanti vengono filtrate nel sistema di backend (server di rete) che dispone anche dell'intelligenza necessaria per il controllo della sicurezza, l'invio di conferme al dispositivo finale e l'invio del messaggio all'application server corrispondente. Inoltre, LoRaWAN sfrutta più ricezioni dello stesso messaggio da diverse stazioni ba-

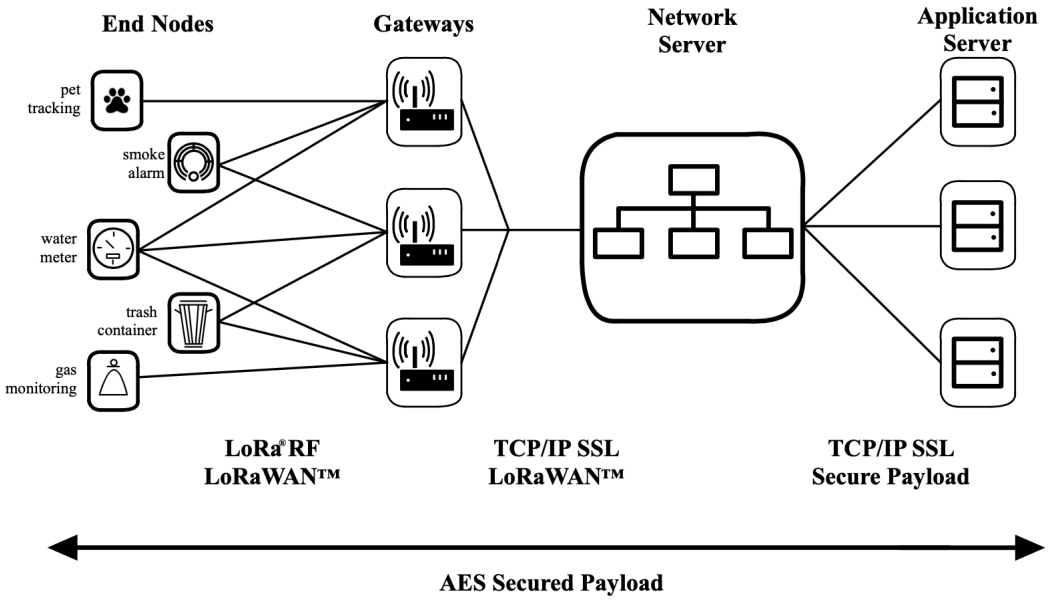

Figura 1.2: Architettura di rete LoRaWAN

se per la localizzazione dei dispositivi finali. A tale scopo, viene utilizzata la tecnica di localizzazione basata sulla differenza di tempo di arrivo (TDOA) supportata da una sincronizzazione temporale molto accurata tra più stazioni base. Inoltre, più ricezioni dello stesso messaggio su stazioni base diverse evitano l'handover nella rete LoRaWAN (ovvero, se un nodo è mobile o in movimento, non è necessario l'handover tra le stazioni base) [1].

LoRaWAN fornisce varie classi di dispositivi finali per soddisfare i diversi requisiti di un'ampia gamma di applicazioni IoT, ad esempio i requisiti di latenza [1].

- *Bidirectional end devices* (classe A): i dispositivi finali di classe A consentono comunicazioni bidirezionali in cui la trasmissione uplink di ciascun dispositivo finale è seguita da due brevi finestre di ricezione downlink. Il lotto di trasmissioni pianificato dal dispositivo finale si basa sulle proprie esigenze di comunicazione con una piccola variazione derivata da una base temporale casuale. Questa operazione di classe A è il meccanismo dei dispositivi finali a minor potenza per le applicazioni

che richiedono solo una breve comunicazione in downlink dopo che il dispositivo finale ha inviato un messaggio in uplink. Le comunicazioni di downlink in qualsiasi altro momento dovranno attendere fino al successivo messaggio di uplink del dispositivo finale.

- *Bidirectional end devices with scheduled receives lots* (classe B): oltre alle finestre di ricezione casuali di classe A, i dispositivi di classe B aprono finestre di ricezione aggiuntive ad orari pianificati. Per aprire le finestre di ricezione all'ora programmata, i terminali ricevono un beacon sincronizzato nel tempo dalla stazione base. Ciò consente al server di rete di sapere quando il dispositivo finale è in ascolto.

- *Bidirectional end devices with maximal receive slots* (classe C): i dispositivi terminali di classe C hanno finestre di ricezione quasi continuamente aperte e si chiudono solo durante la trasmissione a scapito di un consumo energetico eccessivo.

Nell'ottobre del 2017 è stata rilasciata una nuova versione di LoRaWAN, denominata LoRaWAN 1.1. Le principali differenze con la versione 1.0 sono l'introduzione del Join Server (JS), che consente il roaming dei dispositivi finali, e di diversi miglioramenti della sicurezza. Come nel caso di qualsiasi sistema informatico, la sicurezza è una delle maggiori preoccupazioni in LoRaWAN. La proliferazione di dispositivi IoT dipende dall'accettazione pubblica di questi dispositivi come parte di un sistema affidabile. Quindi, migliorarne il livello di sicurezza molto importante per acquisire supporto e accettazione da parte del pubblico. [2]. Diversi studi e analisi effettuate hanno portato alla luce numerosi punti deboli e vulnerabilità di LoRaWAN v1.0. Nei paragrafi successivi, quando si parla di LoRaWAN, si fa riferimento alla versione 1.1, a meno che non venga specificato diversamente.

1.1.3 LoRaWAN: End Device Activation

La Fig. 1.3 mostra l'architettura di LoRaWAN v1.1, per cui è presente il Join Server, e vi sono tre Network Server piuttosto che uno: "home", "forwarding"

e "serving". La logica alla base di questa inclusione è quella di consentire il roaming dei dispositivi in tutta la città, in tutto il paese o, da una prospettiva più elevata, in tutto il mondo. Come nel caso di LoRaWAN v1.0, i nodi finali e i gateway sono collegati con topologia a "stella", e il resto della rete, inclusi GW e server, sono collegati tra loro tramite topologie dette "fully-connected mesh" o "partially-connected mesh" [2].

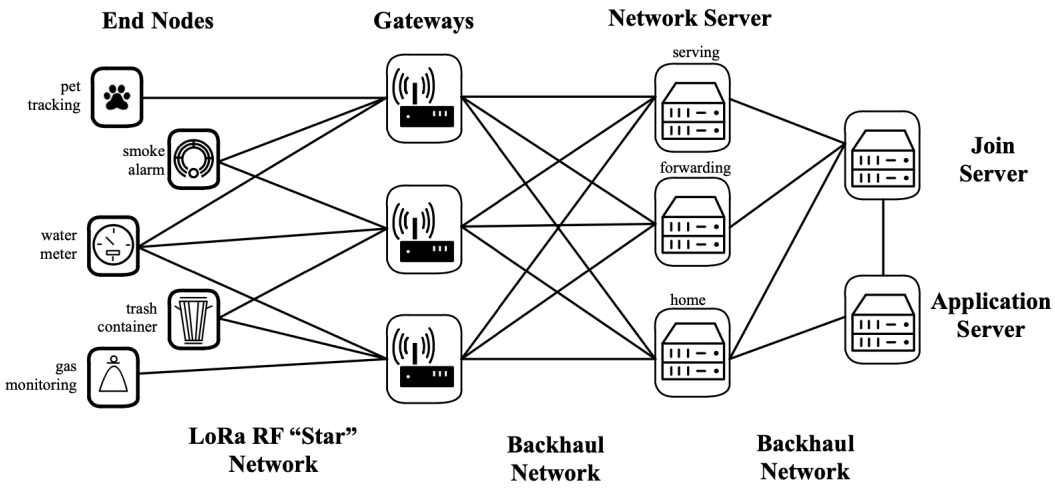

Figura 1.3: Architettura di rete di LoRaWAN v1.1

Ogni end device (ED) deve essere registrato con una rete prima di poter inviare e ricevere messaggi. Esistono due diversi tipi di registrazione:

- *Over-The-Air-Activation* (OTAA): questo è il metodo di attivazione più sicuro. I dispositivi eseguono una procedura di collegamento con la rete, durante la quale viene assegnato un indirizzo di dispositivo dinamico e le chiavi di sicurezza vengono negoziate con il dispositivo.

- *Activation By Personalization* (ABP): richiede l'hardcoding dell'indirizzo del dispositivo e delle chiavi di sicurezza nel dispositivo. ABP è meno sicuro di OTAA e ha anche lo svantaggio che i dispositivi non possono cambiare provider di rete senza cambiare manualmente le chiavi nel dispositivo [6].

Nell'OTAA, la procedura di join prevede lo scambio di due messaggi MAC tra l'ED e il JS: Join-request e Join-accept. Prima dell'attivazione occorre che alcuni valori siano memorizzati nel dispositivo, ovvero:

- AppKey e NwkKey: l'Application key e la Network key sono due chiavi segrete AES-128, conosciute come root key.
- DevEUI: ID dispositivo finale globale a 64 bit nello spazio degli indirizzi IEEE EUI64 che identifica in modo univoco il dispositivo finale.
- JoinEUI: ID applicazione globale a 64 bit nello spazio degli indirizzi IEEE EUI64 che identifica in modo univoco il JS che può assistere nell'elaborazione della procedura di join e nella derivazione delle chiavi di sessione.

Le AppKey, NwkKey e DevEUI corrispondenti devono essere fornite sul Join Server, che assisterà nell'elaborazione della procedura di join e nella derivazione delle chiavi di sessione.

Il messaggio di Join-request viene trasmesso dal dispositivo e consiste in tre campi, che sono appunto il DevEUI, il JoinEUI e una DevNonce, ovvero un 2-byte counter, che inizia a 0 quando il dispositivo viene connesso per la prima volta e incrementa a ogni Join-request. Quest'ultimo valore viene utilizzato per prevenire i replay-attack. Inoltre all'inizio del messagio di Join-request vi è un byte di MHDR e alla fine 4 bytes di MIC, che viene calcolato utilizzando la NwkKey. Il frame risultante da questi campi viene detto PHYPayload. Il messaggio di Join-request può essere trasmesso utilizzando un qualsiasi data-rate e uno qualsiasi dei join channel specifici, e viaggia attraverso uno o più gateway, che semplicemente ritrasmettono il messaggio al Network Server. Quest'ultimo utilizza il DNS per cercare l'indirizzo IP del JS basato sul JoinEUI del messaggio di Join-request ricevuto. Solo se questa ricerca ha esito positivo, allora il Network Server può mandare il messaggio al JS, il quale, a sua volta, processa il messaggio di Join-request e manda un messaggio di JoinAns al Network Server.

Il messaggio di Join Ans contiene:

- PHYPayload with Join-accept message
- Network session keys
- Serving Network session integrity key (SNwkSIntKey)
- Forwarding Network session integrity key (FNwkSIntKey)
- Network session encryption key (NwkSEncKey)
- Encrypted AppSKey

Il Network Server prepara il messaggio di Join-accept e lo cripta con la NwkKey e lo manda come normale downlink.

L'ED riceve il messaggio, calcola il MIC e genera le network session keys e l'AppSkey. Le altre chiavi (FNwkSIntKey, SNwkSIntKey, and NwkSEncKey) sono derivate dalla NwkKey.

Quando il Network riceve un pacchetto in uplink dal dispositivo finale, il server di rete invia il DevEUI e l'AppSKey crittografata insieme al payload dell'applicazione all'Application Server (AS). Quest'ultimo, decripta prima l'AppSKey, grazie ad una chiave condivisa con il JS, e poi, tramite l'AppSKey è in grado di decifrare il payload ricevuto.

Dopo l'attivazione, alcune informazioni addizionali devono essere salvate nell'ED, ovvero:

- DevAddr: il DevAddr viene allocato dal server di rete del dispositivo finale. È un indirizzo del dispositivo a 32 bit che identifica il dispositivo finale all'interno della rete corrente.
- Tripletta di chiavi di sessione di rete: FNwkSIntKey, SNwkSIntKey e NwkSEncKey
- Chiave di sessione dell'applicazione: AppSKey

Il compito di queste chiavi di sessione viene descritto nella sottosezione 1.1.4.

Tutto ciò, come già detto riguarda l'OTAA. L'Activation By Personalization (ABP), invece, collega direttamente un dispositivo finale a una rete

preselezionata, ignorando la procedura di unione. Un ED attivato utilizzando il metodo ABP può funzionare solo con una singola rete e mantiene la stessa sessione di sicurezza per tutta la sua durata.

1.1.4 LoRaWAN: Security

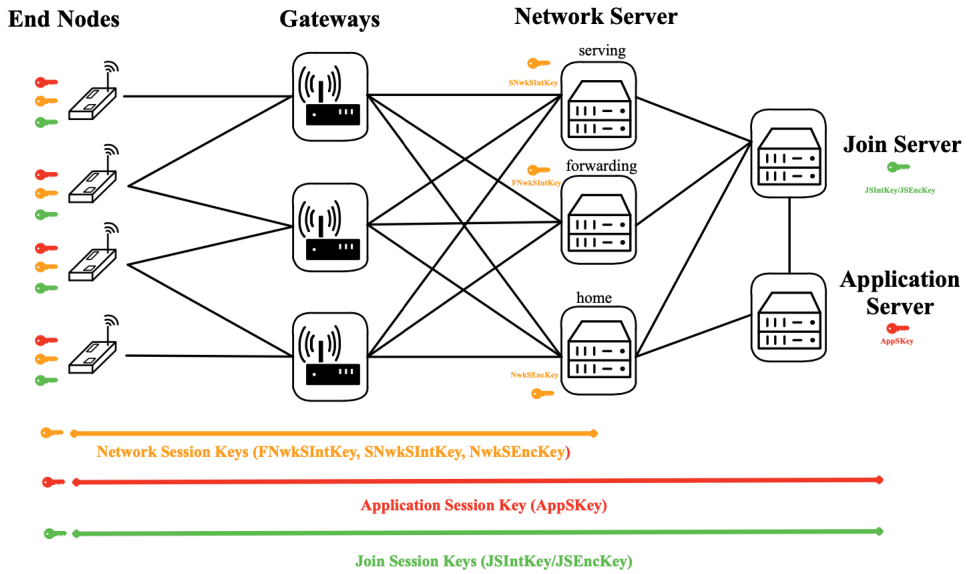

Figura 1.4: Architettura di rete di LoRaWAN v1.1

Come descritto precedentemente, con LoRaWAN v1.1 sono stati introdotti diversi meccanismi che ne aumentano la sicurezza. Le varie chiavi di sicurezza, la cui distribuzione è mostrata nella Fig. 1.4 , vengono analizzate più nel dettaglio in questa sottosezione [2].

AppKey e NwkKey sono le root key AES-128 bit da cui vengono generate le chiavi di sessione. Queste root key sono specifiche per ogni dispositivo finale e quindi sono incorporate in ognuno di essi durante la fabbricazione. In base alle specifiche, queste root key devono essere archiviate in modo sicuro dall'hardware, ad esempio utilizzando elementi di memoria resistenti e a prova di manomissione, come Secure Elements (SE) o Hardware Security Modules (HSM).

AppSKey, generata dalla root key AppKey, è la chiave di sessione utilizzata tra un end device e l'Application Server. Questa chiave viene utilizzata per crittografare/decrittografare i payload a livello di applicazione ed è specifica per ogni end device. I Network Server sono considerati parti fidate per trasmettere onestamente questi messaggi crittografati a ciascuna estremità. L'AppSKey deve essere tenuta segreta agli estranei archiviandola in modo sicuro.

Dalla root key NwkKey vengono generate diverse chiavi di sessione e di durata, che vengono ora descritte più nello specifico.

Per garantire l'integrità dei messaggi provenienti dai Network Server, a tutti i data message in uplink-downlink viene aggiunto il MIC (Message Integrity Code). Inoltre, per ogni dispositivo finale, esiste una network session key specifica. Tutte le chiavi di sessione sono derivate utilizzando il valore NwkKey, JoinNonce, JoinEUI e DevNonce, e sono state introdotte nella v1.1. Ovviamente devono essere conservate in modo sicuro per evitare che possano essere estratte e riutilizzate da soggetti malintenzionati.

La FNwkSIntKey è una chiave di sessione di rete specifica per ciascun dispositivo finale che viene utilizzata per calcolare il MIC di tutti i data message in uplink per garantire l'integrità dei dati. Questa chiave è considerata "pubblica" e può essere condivisa con un roaming forwarding-network server (fNs).

La SNwkSIntKey è una chiave di sessione di rete specifica per ciascun dispositivo finale, utilizzata per calcolare il MIC di tutti i data message in downlink e per garantire l'integrità dei dati. Questa chiave è considerata "privata" e non deve essere condivisa con un fN di roaming.

La NwkSEncKey è anch'essa una chiave di sessione di rete specifica per ciascun dispositivo finale e viene utilizzata per la crittografia/decrittografia dei comandi del livello MAC (payload).

Affinchè un dispositivo possa riconnettersi alla rete a seguito di una disconnessione, vengono utilizzate le join session key. Durante la generazione di queste chiavi, oltre alla root key NwkKey, viene utilizzato anche il DevEUI.

Esse sono la JSIntKey, utilizzata per il MIC dei messaggi Rejoin-Request e relative risposte Join-Accept, e la JSEncKey, che al contrario, è utilizzata per la crittografia dei messaggi Join-Accept in risposta ai messaggi Rejoin-Request.

1.1.5 LoRaWAN: Vulnerabilità

Sebbene la versione 1.1 abbia vantaggi in termini di roaming e mobilità per i dispositivi finali, sono numerosi i problemi e i rischi per la sicurezza.

Per quanto riguarda le entità di rete collegate, si può affermare che i gateway sono sicuramente l'anello debole della catena, in quanto nella maggior parte degli scenari di implementazione, ne vengono implementati pochi (a volte anche 1 o 2) e qualsiasi tipo di attacco di cattura o attacco fisico distruggerebbe la comunicazione tra gli ED e il resto della rete. Inoltre la presenza di numerosi server (almeno tre NS, AS, JS) rende sicuramente difficile la gestione della rete e comporta che tutte le chiavi debbano essere installate a priori nei dispositivi, rendendo difficoltoso l'utilizzo di un meccanismo di re-keying delle root key, che renderebbe il sistema più sicuro. Infine, per quanto concerne la sicurezza delle entità di rete, occorre che gli ED siano dotati di un hardware a prova di manomissione.

Un altro problema d considerare è quello legato alla distribuzione delle chiavi, che ovviamente dipende dal meccanismo di attivazione (OTAA e ABP). ABP, come già anticipato è meno sicuro rispetto all'OTAA, in quanto i dispositivi utilizzano le stesse chiavi di sessione per tutta la loro vita (ovvero, non è possibile il re-keying). Per giunta i vari contatori sono mantenuti nella memoria non volatile, e se questa dovesse presentare dei problemi, il dispositivo andrebbe fuori sincronizzazione, diventando inutilizzabile. Nel caso di OTAA il problema principale riguarda le root key, che vengono installate in fase di fabbricazione e dalle quali vengono derivate tutte le altre chiavi, motivo per il quale devono essere conservate nel modo più sicuro possibile. Proprio per questa ragione si suggerisce l'uso di una PKI che sia in grado di distribuire le chiavi ai dispositivi finali piuttosto che installarle in fase di fabbricazione.

Alcune vulnerabilità sono dovute all'implementazione. Tra i vari problemi ad essa legati, occorre prestare attenzione alla procedura di uscita, la quale viene utilizzata per la disattivazione dei dispositivi finali dopo la scadenza della licenza o se si ritiene che siano compromessi e si desidera che vengano tagliati fuori dalla rete. Tuttavia, manca una definizione standard relativa alla procedura di uscita e ciò potrebbe causare complicazioni, in quanto ad esempio, una procedura di uscita di un dispositivo finale dovrebbe comportare la chiusura permanente di tutti gli ID, password, contatori e nonce relativi a quel dispositivo specifico. Un'altra possibile vulnerabilità legata all'implementazione riguarda il DevEUI, che viene salvato nel dispositivo in fase di fabbricazione, mentre i NS memorizzano una lista di DevEUI, ma non vi è alcuna procedura per revocare o rinnovare questo valore. Anche i contatori costituiscono un'area di implementazione delicata. Nella specifica di LoRaWAN v1.1 si afferma quanto segue: "Per i dispositivi OTAA, i contatori di frame non devono essere riutilizzati per una determinata chiave, pertanto il nuovo contesto di sessione deve essere stabilito molto prima della saturazione di un contatore di frame. Si consiglia di mantenere lo stato della sessione durante il ciclo di alimentazione di un dispositivo finale. In caso contrario per i dispositivi OTAA, la procedura di attivazione dovrà essere eseguita ad ogni accensione e spegnimento di un dispositivo" [2]. Infine, anche il JoinEUI costituisce un problema, in quanto viene memorizzato nei dispositivi prima di avviare la procedura di OTAA, motivo per cui una modifica del JS comporterebbe un cambiamento in tutti gli ED.

Un altro possibile problema è relativo alla fiducia. I NS sono considerati parti fidate per trasmettere onestamente messaggi crittografati. Sebbene i payload dell'applicazione sono crittografati end-to-end tra l'ED e l'AS, l'integrità è garantita solo in modo hop-by-hop: uno hop tra il dispositivo finale e il NS e l'altro hop tra il NS e l'AS. Ciò comporta che un NS malintenzionato può essere in grado di alterare il contenuto dei messaggi e di dedurre alcune informazioni sui dati. In conclusione a questa affermazione, si può affermare che la riservatezza end-to-end e la protezione dell'integrità tra i server non

sono garantite o supportate da LoRaWAN v1.1 [2].

Infine ci sono problemi che riguardano proprio la v1.1, ovvero il roaming, che è stato appunto introdotto con questa versione, in quanto suscettibile ad attacchi di bit, o MITM, e infine, ma non meno importante, la compatibilità di LoRaWAN 1.1 con dispositivi della v1.0 comporta alcuni rischi.

1.1.6 LoRaWAN: Possibili Attacchi

In linea con le vulnerabilità descritte nella precedente sottosezione, vengono elencati, in questa, quelli che sono i possibili attacchi informatici che possono essere apportati a LoRaWAN 1.1.

Nelle vulnerabilità relative all'implementazione, il rinnovamento di DevEUI e JoinEUI è stato segnalato come problematico, per cui è possibile la creazione di falsi pacchetti di Join, sebbene sia un attacco poco probabile.

Altri possibili attacchi sono gli attacchi man-in-the-middle (MITM). I bit-flipping attack consistono nel cambiamento del contenuto di un messaggio tra il NS e l'AS, mentre i frame playload attack sono possibili in quanto l'handover-roaming consente maggiori possibilità per un attacco MITM, poichè i FRMPayload non protetti vengono prima trasportati da sNS (serving-NS) a hNS (homing-NS) e da lì all'AS.

Sfruttando un dispositivo finale è possibile effettuare attacchi contro il resto della rete, ad esempio degradarla inondandola di pacchetti.

Una network traffic analysis è un attacco di tipo passivo, nel quale un utente malintenzionato può configurare un Rogue-GW per ricevere pacchetti e dedurre alcune conoscenze su dati trasmessi o sul materiale utilizzato per le chiavi.

Ovviamente possono essere effettuati degli attacchi fisici a una rete LoRaWAN. Un ED può senz'altro essere rubato o distrutto, sebbene la generazione delle root key avviene in modo univoco per ogni dispositivo, per cui la rivelazione di una root key non permette di scoprire informazioni su tutta la rete, ma solo sul singolo dispositivo. La specifica di LoRaWAN richiede, inoltre, la protezione del materiale della chiave contro il riutilizzo, per cui i nodi devono

essere adeguatamente protetti contro la modifica del firmware, che potrebbe indirettamente portare al riutilizzo.

Per giunta, se non sono usati sistemi di archiviazione sicura e le chiavi sono salvate in semplici file, allora è possibile che venga effettuato un Plaintext Key Capture, minaccia per la riservatezza e l'integrità della rete.

È stato anche dimostrato come, utilizzando hardware a basso costo e di largo consumo, il jamming di segnali RF sia possibile, e bisogna tener conto che un RF jamming attack può portare ad un Denial of Service (DoS), che è difficile da rivelare.

Come già accennato prima, un dispositivo finale può essere catturato e, se ne viene sostituito il firmware può essere utilizzato per effettuare un Rogue-ED attack. Tuttavia, gli ED non sono autorizzati ad accedere ad informazioni sul Server, quindi inviare informazioni false sulla rete è il danno maggiore che può essere apportato. Inoltre, un dispositivo finale, può essere utilizzato come jammer nella rete, portando così, come già accennato prima, a un DoS, sebbene questo sia circoscritto ad un'area nella vicinanze del dispositivo in questione, mentre il resto della rete continua a funzionare regolarmente. Infine, gli ED non autorizzati possono essere utilizzati per eseguire replay attack. I pacchetti trasmessi dai vicini possono essere catturati e ritrasmessi successivamente. Sebbene ciò sia comunque rilevabile a causa dell'uso di diversi valori di nonce (DevNonce, JoinNonce ecc), potrebbe comunque portare ad uno spreco di risorse, e ad una minore disponibilità dei GW ai nodi legittimi.

Anche i gateway, sebbene siano considerati parti fidate, possono essere hackerati, catturati o replicati, motivo per cui anche un Rogue-Gateway attack è possibile. In particolare, in questo campo, due attacchi possono essere realizzati. Il primo, il Beacon Synchronization DoS Attack, prevede che un malintenzionato utilizzi un rogue-gw per inviare beacon falsi. Questo tipo di attacco colpisce sopratutto le sessioni di Classe B, in quanto i beacon di tale classe non sono protetti in alcun modo, e fa sì che i dispositivi ricevano messaggi in finestre non sincronizzate, oppure può portare a un aumento della probabilità di collisione dei pacchetti. Il secondo, l'Impersonation Attack,

avviene quando un gateway viene impersonato per attaccare gli ED, ascoltandone la comunicazione, determinandone gli indirizzi di rete, o, localizzandoli tramite operazioni di triangolazione.

Gli attacchi di routing sono possibili, sebbene la probabilità che avvengano è molto bassa, in quanto è necessario che un malintenzionato si impadronisca di un gateway o di un server, non basta avere a disposizione un ED, in quanto questo non partecipa al routing. Anche in questo caso è possibile fare la distinzione tra due diversi attacchi. Il primo viene detto Selective Forwarding Attack, nel quale un utente malintenzionato può inoltrare i pacchetti in modo selettivo e può causare il blocco totale o il sovraccarico di uno o più nodi della rete. Il secondo, Sinkhole o Blackhole Attack, prevede che un utente attragga il traffico attraverso se stesso pubblicizzando, in maniera falsa, informazioni di routing modificate. Di conseguenza, l'intero traffico di rete potrebbe crollare.

Infine, un attacco che può essere apportato e che mette in serio pericolo la sicurezza di LoRaWAN è il Self-Replay Attack. Esso sfrutta la procedura di Join di LoRaWAN v1.1, bloccando selettivamente i segnali che vengono utilizzati per una sessione di OTAA. In questo tipo di attacco, viene osservata con successo la trasmissione di un segnale di Join-request da ED a NS, e, il corrispondente messaggio di Join-accept, da NS a ED viene bloccato utilizzando tecniche di selective-jamming. Dopo aver atteso un timeout per ricevere il messaggio Join-accept dal NS, l'ED tenta di accedere nuovamente alla rete e invia lo stesso messaggio Join-request con lo stesso valore DevNonce richiesto dalla specifica. Questo fa sì che il NS risponda alla richiesta di join, perché ancora la procedura di join non è soddisfatta e tutto è legittimo e in ordine secondo le specifiche. Questo attacco continuerà fino all'esaurimento della quota giornaliera di messaggi dell'ED [2]. Nelle sessioni di OTAA vi è un limite massimo di pacchetti che un ED può trasmettere al giorno, cioè 14, motivo per cui questo tipo di attacco è particolarmente valido per i dispositivi OTAA.

1.2 The Things Stack

1.2.1 TTS

The Things Stack (TTS), sviluppato e gestito da The Thing Industries (TTI), è un LoRaWAN Network Server robusto e flessibile, che soddisfa tutte le esigenze impegnative di implementazione di una rete LoRaWAN, dalla copertura degli elementi essenziali alle configurazioni di sicurezza avanzate e alla gestione del ciclo di vita dei dispositivi. In altre parole, TTS, costruito su un core open source, consente di creare e gestire reti LoRaWAN sul proprio hardware o cloud [7].

Sono diverse le distribuzioni disponibili, che consentono di soddisfare numerose esigenze, ma le più comuni sono:

- Cloud: server di rete con supporto SLA ospitato da TTI. Ciò vuol dire che TTI Cloud è un ambiente multi-tenant, per cui mentre ogni cliente ha la propria rete isolata, l'infrastruttura sottostante è condivisa con altri clienti.
- Enterprise: permette di installare il Network Server sul proprio hardware.
- Community Edition: consente di creare la propria rete LoRaWAN sfruttando il Network Server gratuito fornito da The Things Network (TTN).

Quindi, quando si parla di TTS, si fa riferimento allo stack del Network Server di LoRaWAN. Esso è attualmente alla versione 3 di un'implementazione di Network Server, ed è quindi anche informalmente noto come V3 [7].

Invece, TTN è un ecosistema collaborativo globale di Internet of Things che crea reti, dispositivi e soluzioni utilizzando LoRaWAN. TTN esegue TTS Community Edition [7].

La società responsabile dello sviluppo di The Things Stack e della stesura della documentazione è TTI.

Sebbene TTS fornisca una Console grazie alla quale è possibile gestire applicazioni e dispositivi, vengono comunque offerte delle API HTTP e GRPC con cui interagire [8].

Capitolo 2

Architettura Funzionale del Sistema Realizzato

2.1 Nodi della Rete

Come già ampiamente descritto nelle sottosezioni 1.1.5 e 1.1.6, uno dei principali problemi di sicurezza di LoRaWAN è costituito dalla mancanza di un meccanismo che permetta di effettuare il re-keying delle root key, le quali vengono installate nel dispositivo in fase di fabbricazione e grazie alle quali vengono generate tutte le chiavi di sessione.

Per far ciò sono stati utilizzati, per lo meno in fase di sperimentazione, diversi strumenti. In particolare in questa sezione vengono descritti gli elementi che formano la rete LoRaWAN.

- End Device Virtuali: sono degli ED non fisici, ma solo software (realizzati tramite Java) che emulano il comportamento di dispositivi fisici. Quindi, connettendosi con un gateway permettono di scambiare messaggi con i vari NS, AS e JS. In particolare ci sono sei dispositivi virtuali che possono essere utilizzati:
  - CountDevice: semplice dispositivo con un intero (leggibile e scrivibile) che viene incrementato a ogni lettura.

  - CurrentTimeDevice: semplice dispositivo che restituisce una data (per la sola lettura) in formato stringa YYYY-MM-dd HH:mm:ss.
  - DataDevice: dispositivo che restituisce un dato (leggibile e scrivibile) salvato in RAM, che deve essere passato come parametro.
  - FileDevice: anche questo dispositivo restituisce un dato (leggibile e scrivibile), il quale però è salvato in un file passato come parametro.
  - Dispositivi Draghino: in particolare LHT65 (con valori artificiali di temperatura e umidità) e LSE01 (con valori artificiali di temperatura e umidità del suolo).

- Gateway Virtuale: anch'esso non è un elemento fisico, e viene utilizzato per connettersi ad un NS LoRaWAN remoto utilizzando il protocollo Semtech e per trasmettere i dati da alcuni dispositivi virtuali al NS.

- TTN: è stato presentato nella sottosezione 1.2.1, per cui viene utilizzato come piattaforma di backend LoRaWAN a cui collegare il gateway virtuale per poterlo eseguire. Inoltre si fa uso dell'API fornita da TTS di TTN per poter gestire i dispositivi.

Come prima cosa viene registrato il gateway e i dispositivi finali virtuali su TTN, procedura che può essere effettuata sia tramite API, che tramite la Console di TTS. Occorre quindi specificare prima di tutto la regione, poi i vari parametri del Gateway (ID, EUI e Frequency plan) e degli ED (metodo di attivazione, versione di LoRaWAN, ID, JoinEUI, DevEUI, oltre a Fequency plan, e ovviamente un'AppKey). Anche la gestione può avvenire sia tramite Console che API, ma ai fini del sistema viene utilizzata l'API, la quale è fornita direttamente dalla piattaforma di backend.

Per attivare i dispositivi e il gateway vengono utilizzati dei file di configurazione, che contengono i vari identificatori e chiavi necessari per avviare la comunicazione.

Affinchè questi elementi possano interagire, e quindi formare una rete LoRaWAN, occorre seguire diversi passaggi, che sono rappresentati nel sequence diagram mostrato nella Fig. 2.1.

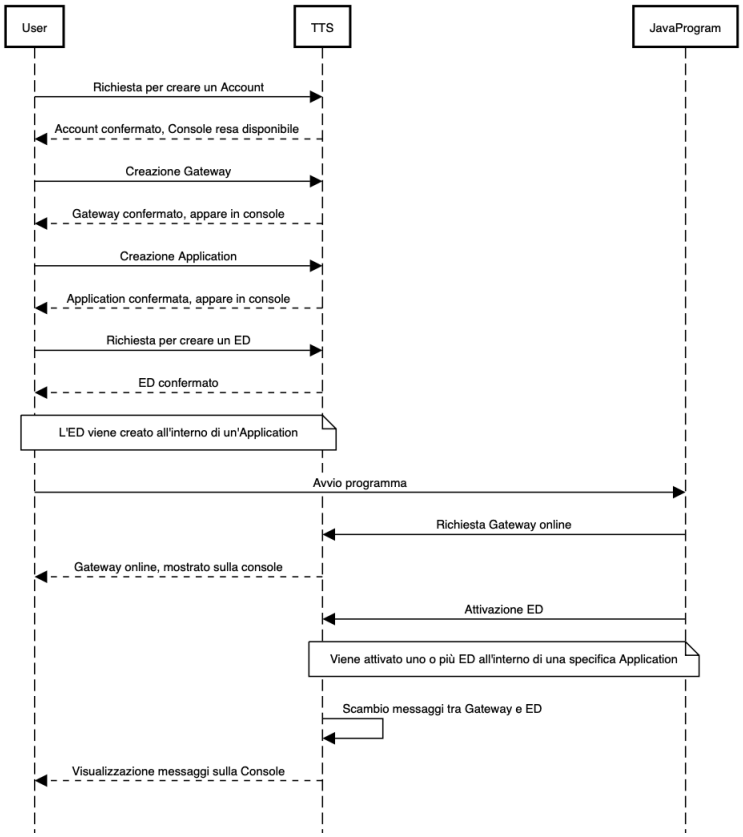

Figura 2.1: Sequence diagram della registrazione di gateway e dispositivi finali su TTS

2.2 API

L'API che TTS espone utilizza HTTP come protocollo di comunicazione, per cui è possibile fare richieste di tipo GET, POST, PUT e DELETE. Ovviamente, trattandosi di richieste HTTP, le richieste di tipo GET permettono di ottenere una risorsa, le POST di aggiungerne una nuova, le PUT di aggior-

narla e quelle di tipo DELETE di eliminarla. Gli oggetti che devono essere forniti tramite le richieste, devono essere dati in formato JSON.

Tutte le chiamate API devono essere autorizzate o da una API key, o da un OAuth access token o da Session cookie. Le API key, utilizzate in questa architettura, sono il metodo di autorizzazione più semplice, in quanto non scadono, sono revocabili e hanno come ambito l'entità da cui sono state generate [9]. Quindi ci sono diversi tipi di API key.

- User API key
- Application API key
- Gateway API key
- Organization API key

L'API di TTS permette di specificare un subset di campi che devono essere restituiti da una richiesta di lettura, o un subset di campi che devono essere aggiornati in una richiesta di scrittura, tramite i cosiddetti field-mask, come mostrato nella Fig. 2.2.

```
{
  "field_mask": {
      "paths": [
        "ids.device_id",
        "name",
        "description"
      ]
    }
}
```

Figura 2.2: Formato del field-mask estratto dal body di una richiesta

Quando si parla di richieste GET, ciò che si vuole ottenere può essere specificato direttamente nell'URL, inserendoli come stringa, mentre nelle altre richieste devono essere inseriti nel body del messaggio. Quindi tutti i campi che non sono specificati nei field-mask non vengono restituiti o aggiornati.

Quindi, ad esempio, una richiesta PUT, il cui body presenta una sezione field-mask di questo tipo, aggiorna l'id, il nome e la descrizione di un determinato ED.

2.3 Descrizione dell'Algoritmo

2.3.1 Generazione di chiavi

Algoritmo 2.1 Update Keys

1: $newKey \leftarrow null$
2: $apiKey \leftarrow BearerNNSXS.XXXXXXXXXX$
3: $url \leftarrow https://tts.com/api/v3/js/applications/app1/devices/dev1$
4: $timer \leftarrow 0$
5: $period \leftarrow 50000$
6: $timer.Start()$
7: **while** $timer$ **do**
8: $\quad newKey \leftarrow KeyGenerator()$
9: $\quad SendPutRequest(url, apiKey, newKey)$
10: $\quad fileDiConfigurazione \leftarrow write(newKey)$
11: $\quad wait(period)$
12: **end while**

Il linguaggio di programmazione utilizzato è Java, che fornisce di per sè delle librerie per il protocollo HTTP e per la generazione di chiavi AES-128.

Tramite l'API, e in particolare tramite richieste di tipo PUT è possibile aggiornare l'AppKey di un qualsiasi ED, se specificato nel field-mask della richiesta.

L'algoritmo, come mostra lo pseudocodice 2.1 genera ripetutamente delle chiavi AES-128, con intervalli regolari scanditi da un timer. Queste chiavi, di volta in volta vengono inserite nel body delle richieste HTTP che vengono inviate a TTS. Di fatto la richiesta PUT utilizzata altro non fa che registrare

i campi inseriti nel field-mask nel Join Server. Quindi, inserendo nella sezione field-mask un nuovo valore di AppKey, questo viene registrato nel JS.

Poichè, come viene spiegato nella sottosezione 2.3.3, occorre effettuare il re-join del dispositivo, indipendentemente dal fatto che questo avvenga in maniera manuale o automatica, il programma di volta in volta deve aggiornare il valore della root key nel file di configurazione del dispositivo virtuale, così che questo poi corrisponda al nuovo valore registrato nel JS. La Fig. 2.3 mostra il seqeunce diagram dell'algoritmo descritto in questa sottosezione.

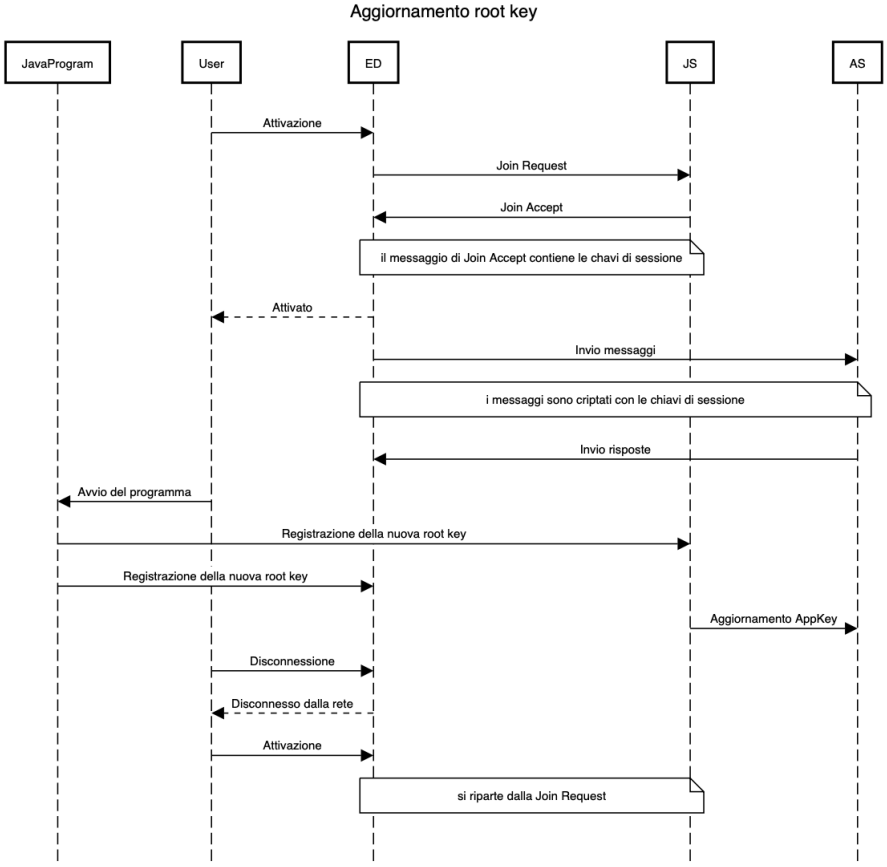

Figura 2.3: Sequence diagram dell'algoritmo di generazione e aggiornamento dell'AppKey

2.3.2 Richiesta PUT per l'aggiornamento

Come già accennato nella sottosezione 2.3.1, una richiesta PUT permette di aggiornare la root key AppKey.

Trattandosi di una richiesta HTTP, deve sicuramente avere un Header, nel quale viene specificato il tipo di autorizzazione, il cosiddetto Content-Type e lo User-Agent. Per quanto riguarda il tipo di autorizzazione, in questo caso si fa uso di una API key, e in particolare di una User API key. Il Content-Type specifica ovviamente il formato dei dati, che è di tipo JSON. Infine, lo User-Agent, che è buona pratica inserire, contiene il nome della propria integrazione e la sua versione. Il body della richiesta invece deve specificare tutte le informazioni necessarie per l'individuazione del dispositivo nella rete e il nuovo valore di AppKey nella sezione field-mask. La richiesta PUT ha verosimilmente ha un formato come quello mostrato nella Fig.2.4.

```
--header 'Authorization: Bearer NNSXS.XXXXXXXXX' --header 'Content-Type: application/json'
--header 'User-Agent: my-integration/my-integration-version' \
--request PUT \
--data-raw '{
  "end_device": {
    "ids": {
      "device_id": "newdev1",
      "dev_eui": "XXXXXXXXXXXXXXXX",
      "join_eui": "XXXXXXXXXXXXXXXX"
    },
    "network_server_address": "thethings.example.com",
    "application_server_address": "thethings.example.com",
    "root_keys": {
      "app_key": {
        "key": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
      }
    }
  },
  "field_mask": {
    "paths": [
      "network_server_address",
      "application_server_address",
      "ids.device_id",
      "ids.dev_eui",
      "ids.join_eui",
      "root_keys.app_key.key"
    ]
  }
}' \
```

Figura 2.4: Formato della richiesta PUT per registrare l'AppKey nel JS

2.3.3 Re-join

Affinchè vengano generate delle chiavi di sessione con la nuova root key ottenuta come spiegato nella sottosezione 2.3.1, occorre effettuare un nuovo Join, perché se così non fosse, la comunicazione con l'AS continuerebbe sfruttando le chiavi di sessione generate con il vecchio valore dell'AppKey.

Infatti la chiave di sessione AppSKey, che viene utilizzata per criptare e decriptare il payload dei messaggi che sono scambiati tra ED e AS, viene generata tramite l'AppKey ed è valida per un'intera sessione. Ciò vuol dire che a ogni nuova attivazione, cioè ogni volta che si effettua nuovamente il Join di un dispositivo, questa viene rigenerata.

Ovviamente anche se la root key non fosse cambiata, a ogni riconnessione del dispositivo alla rete verrebbe generata una nuova AppSKey, ma tutte queste chiavi di sessione sarebbero sempre derivate da un'unica AppKey, motivo per cui aggiornare periodicamente quest'ultima garantisce di avere un livello di sicurezza maggiore.

Appare scontato che questo meccanismo può essere utilizzato solo con dispositivi che sfruttano l'OTAA, in quanto ciò che è stato appena descritto non vale per l'APB. Infatti, come descritto nel sottosezione 1.1.3, se un ED viene configurato tramite ABP, mantiene le stesse chiavi di sessione per tutta la sua durata, quindi non avrebbe senso aggiornare le root key periodicamente.

Il re-join, per come il sistema realizzato è stato realizzato, deve avvenire in maniera manuale. Quindi sebbene il programma modifichi la root key sia a livello di server (quindi nel JS, grazie all'API di TTS), sia a livello locale (quindi nel file di configurazione del dispositivo virtuale). Poi spetta all'utente occuparsi di scollegare il dispositivo (quando e se lo crede opportuno) e ricollegarlo alla rete, cosè che questo possano essere generate le nuove chiavi di sessione sfruttando la nuova root key.

Come già detto nella sottosezione 2.3.1, le chiavi vengono aggiornate periodicamente. L'utente però non deve obbligatoriamente scollegare il dispositivo e ricollegarlo ogni volta che la root key viene cambiata. Infatti, anche se l'aggiornamento avviene mentre il dispositivo scambia messaggi con

l'AS non si crea alcun tipo di problema, in quanto la comunicazione avviene sfruttando le chiavi di sessione che erano state generate al precedente join, mentre la nuova AppKey non entra gioco finchè il join non avviene nuovamente.

2.3.4 Sicurezza

Tramite l'algoritmo proposto la sicurezza della rete LoRaWAN aumenta notevolmente.

Nella sottosezione 1.1.6 si era discusso proprio del fatto che le root key dovessero essere installate in maniera sicura in quanto uniche per tutto il ciclo del dispositivo. L'aggiornamento dinamico di queste aggiunge un livello di sicurezza in più, in quanto anche se un malintenzionato dovesse appropriarsi di una chiave, potendo così derivare le chiavi di sessione, i danni sarebbero limitati a quella specifica sessione poichè al successivo re-join del dispositivo, le chiavi in suo possesso diventerebbero del tutto inutilizzabili. Un possibile attacco di questo tipo viene mostrato nel sequence diagram rappresentato dalla Fig.2.5.

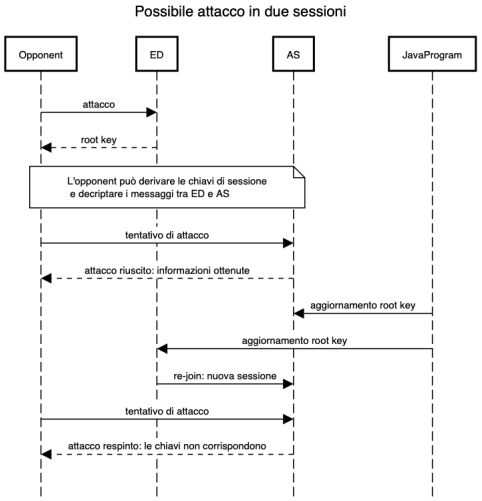

Figura 2.5: Possibile attacco

Capitolo 3

Implementazione

3.1 Registrazione dei Nodi nella Rete

Come già accennato nella sezione 2.1, sia gli ED che il gateway che sono stati utilizzati in questo progetto sono virtuali, quindi composti esclusivamente da una parte software, realizzata tramite Java.

Ovviamente questi devono essere registrati sulla piattaforma di backend, TTN, che offre un'interfaccia semplice e comprensibile, chiamata Console per realizzare questa operazione.

Una volta effettutato l'accesso su TTS con le proprie credenziali (impostate in fase di registrazione), occorre andare nella sezione Console e scegliere la regione, perchè come già spiegato nella sottosezione 1.1.1, le normative legate a LoRa variano in base alla regione. In questo caso si è lavorato su "Europe1".

Una volta aver avuto accesso alla Console, si vede come questa sia divisa in diverse sezioni. Quelle necessrie per questa operazione sono Gateway, che permette di gestire i propri gateway (e quindi anche di inserirne di nuovi), e Application, che consente di gestire e registrare le proprie applicazioni, all'intero delle quali vengono creati gli ED.

Per registrate il gateway devono essere inseriti necessariamente i seguenti campi.

- Owner: chi crea il gateway, quindi l'user
- Gateway ID: identificatore univoco per il gateway stesso
- Gateway EUI: identificatore univoco a 64-bit che permette agli ED di identificare il gateway

In seguito, dopo aver creato l'Application, inserendo un Owner e un ID, si può procedere con la creazione del dispositivo. Trattandosi di un dispositivo virtuale, e quindi non reale, questo non è presente nella repository di LoRaWAN, quindi va inserito manualmente, settando diversi campi.

- Versione di LoRaWAN
- Versione dei parametri regionali
- Frequency plan
- DevEUI
- AppEUI
- AppKey
- End Device ID

Per quanto riguarda la versione di LoRaWAN, in questo progetto è stata utilizzata la versione 1.0.2, piuttosto che l'ultima, ovvero la versione 1.1. I motivi di questa scelta sono principalmente due. Innanzitutto per una questione di semplicità, in quanto in tutte le versioni precedenti la 1.1 non vi sono due root key, ma solo una, ma ovviamente tutto ciò che è stato fatto può essere adattato anche al caso in cui vi ne siano due. Il secondo motivo riguarda l'implementazione dei dispositivi virtuali, su cui non ci si soffer-ma in questo elaborato. Occorre però far riferimento al fatto che il campo "DevNonce" (citato nella sottosezione 1.1.3) fino alla versione 1.0.2 doveva assumere un valore sempre diverso o casuale. Dalla versione successiva, invece, è diventato un contatore che viene incrementato di volta in volta e che

viene mantenuto persistente. Dato che l'implementazione degli ED virtuali non contiene alcun metodo di salvataggio di questo contatore, se si fosse fatto uso di una versione successiva alla versione 1.0.2, ogni qualvolta si fosse tentato di effettuare un re-join, si sarebbe verificato un errore relativo al DevNonce.

Per quanto riguarda invece l'attivazione dei dispositivi (sia gateway che ED), trattandosi di un semplice applicativo Java, basta avviare il programma. Ovviamente bisogna inserire gli stessi parametri dei dispositivi che sono stati impostati in fase di registrazione, più l'indirizzo dell'AS. Questi possono essere passati come paramteri o inseriti in un file di configurazione, che a sua volta deve essere inserito come parametro quando si avvia il programma. Una riga di comando che può avviare il programma di un ED, passando il file di configurazione è:

```
$ java -cp lorawan.jar test.LorawanDevice -f cfg\dev.cfg
```

Un file di configurazione altro non è che un file di testo che contiene identificatori e chiavi necessari per avviare i dispositivi. Ad esempio, se si volesse creare un file di configurazione per un CounterDevice, si avrebbe un testo del tipo:

- appEui= 0000000000000001
- appKey= C9AF8714B512698B2CEE7282CA2682B8
- devEui= feffff2002115001
- devType= CounterDevice
- devTime= 60
- fport= 1

Allo stesso modo viene lanciato il gateway.

La console di TTS, sia nella parte dedicata al Gateway che in quella dedicata agli ED, fornisce un sezione "Live Data", che permette di vedere in tempo reale i messaggi che giungono e che vengono inviati dai dispositivi.

3.2 Algoritmo per l'Aggiornamento dell'AppKey

Nella sottosezione 2.3.1 è stato descritto quali sono le operazioni principali dell'algoritmo. In questa sezione vengono analizzate più nello specifico le entità coinvolte e le loro funzioni.

3.2.1 Richieste HTTP

La classe che permette di interagire con TTS tramte l'API è TTnConnection.

In questa classe in realtà non vi è nulla di strettamente legato a LoRaWAN e TTS, ma all'interno di essa vengono dichiarati e implementati i metodi che permettono di fare richieste HTTP GET, POST e PUT.

Fa uso della libreria di Java HttpUrlConnection, che permette appunto di recuperare informazioni di qualsiasi url HTTP, come informazioni di intestazione, codice di stato, codice di risposta ecc. Il metodo *openConnection()* dell'url, che indica l'indirizzo cui si vuole fare richiesta, stabilisce la connessione.

Ognuno dei metodi che viene implementato in questa classe, per fare una richiesta HTTP, prevede la creazione di un oggetto HttpURLConnection, e l'utilizzo di due metodi ad esso legati:

- *setRequestMethod(String method)*: per specificare il tipo di richiesta HTTP si tratta
- *setRequestProperty(String key, String value)*: che permette di settare le proprietà della richiesta. Il parametro *value* di questo metodo altri non è che l'API Key descritta nella sezione 2.2. Inoltre questa funzione può essere chiamata più volte per impostare proprietà differenti, sempre analizzate nella sezione 2.2.

In seguito, grazie ad un oggetto di tipo DataOuputStream è possibile scrivere data types Java primitivi in modo portabile. Infatti proprio grazie ad esso si scrive il body della richiesta in formato JSON.

Allo stesso modo un oggetto di classe InputStream permette di leggere la risposta.

Nel listato 3.1 viene riportato il metodo utilizzato per l'invio di una richiesta PUT.

```
public String sendPUTRequest(URL url, String urlParameters,
    String key, String value) {
    HttpURLConnection connection = null;

    try {
    //Create connection
    connection = (HttpURLConnection) url.openConnection();
    connection.setRequestMethod("PUT");
    connection.setRequestProperty(key,
        value);

    connection.setRequestProperty("Content-Length",
    Integer.toString(urlParameters.getBytes().length));
    connection.setRequestProperty("Content-Type",
        "application/JSON");
    connection.setRequestProperty("Content-Language",
        "en-US");

    connection.setUseCaches(false);
    connection.setDoOutput(true);

    //Send request
    DataOutputStream wr = new DataOutputStream (
        connection.getOutputStream());
    wr.writeBytes(urlParameters);
    wr.close();

    //Get Response
    InputStream is = connection.getInputStream();
    BufferedReader rd = new BufferedReader(new
    InputStreamReader(is));
    StringBuilder response = new StringBuilder();
    String line;
```

```
34      while ((line = rd.readLine()) != null) {
35        response.append(line);
36        response.append('\r');
37      }
38      rd.close();
39      return response.toString();
40    } catch (Exception e) {
41      e.printStackTrace();
42      return null;
43    } finally {
44      if (connection != null) {
45        connection.disconnect();
46      }
47    }
48  }
```

Listing 3.1: Metodo per inviare una richiesta PUT

3.2.2 Generazione delle Chiavi

Per la generazione delle chiavi segrete AES-128 è stata creata la classe RootKeyGenerator. Anche questa ha solo l'utilità di fornire un metodo per la generazione delle chiavi crittografiche, che sono poi utilizzate come root key AppKey.

RootKeyGenerator fa uso della libreria di Java *KeyGenerator*, che fornisce appunto la funzionalità di un generatore di chiavi segrete (simmetriche).

I generatori di chiavi vengono costruiti utilizzando uno dei metodi *getInstance* di questa classe.

Gli oggetti KeyGenerator sono riutilizzabili, ovvero, dopo che una chiave è stata generata, lo stesso oggetto KeyGenerator può essere riutilizzato per generare ulteriori chiavi.

Quindi si specifica il tipo di chiave (AES) come parametro del metodo *getInstance()* e si inizializza il generatore con una determinata dimensione (128 in questo caso).

Poi istanziando un oggetto di tipo SecretKey e inizializzandolo chiamando il metodo *generateKey()* sul generatore si ottiene la chiave.

L'ultimo passaggio prevde la conversione di questa chiave in stringa esadeciamle, che è il formato richiesto perché possa essere inserita sulla piattaforma TTS.

In conclusione il funzionamento della classe è racchiuso nelle seguenti poche righe di codice:

```
KeyGenerator gen = KeyGenerator.getInstance("AES");
      gen.init(128); /* 128-bit AES */
      SecretKey secret = gen.generateKey();
      byte[] binary = secret.getEncoded();
      String text = String.format("%032X", new BigInteger(+1,
        binary));
```

Listing 3.2: Estratto del metodo per generare chiavi AES-128

3.2.3 Aggiornamento dellAppKey dell'ED

L'ultima classe che compone l'applicativo è chiamata EDHandler, e fornisce appunto le funzioni che permettono di aggiornare l'AppKey del dispositivo.

Occorre quindi un metodo che permetta di inviare la richiesta, sfruttando ovviamente le funzioni esposte dalla classe TTnConnection citata nella sottosezione 3.2.1, e che quindi fa uso dell'API di TTS.

Una volta istanziato un oggetto di tipo TTnConnection si può chiamare il metodo per l'invio di una richiesta PUT. Come già anticipato, le richieste HTTP PUT permettono di aggiornare delle risorse, e in questo caso si vuole registrare sul JS un nuovo valore di AppKey per un dispositivo già esistente.

Qui si può aprire una piccola parentesi sull'estendibilità del programma perchè ciò non toglie che se si fosse voluto registrare un nuovo ED su TTS, sfruttando solo le poche classi citate, sarebbe stato possibile. Ovviamente si tratta di un'operazione multi-step, perchè non basta una sola richiesta PUT. Occorre innanzitutto un richiesta POST che consenta di inserire un nuovo dispositivo in una determinata Application, registrando nell'Identity Server

JoinEUI, DevEUI e cluster address (questi citati sono campi obbligatori, ma se ne possono inserire anche altri nel field-mask, quali nome, descrizione ecc). Poi una richiesta POST per registrare delle impostazioni LoRaWAN nel NS, quali frequency plane e versione di LoRaWAN. Un'altra richiesta PUT per registrare DevEUI e JoinEUI nell'AS, e infine la richiesta PUT che viene utilizzata per l'aggiornamento della chiave.

Allo stesso modo, tramite una richiesta PUT, detta *SreamEventsRequest*, viene data la possibiltà di catturare live il flusso di eventi che avvengono sulla piattaforma. Si può speficiare anche un determinato dispositivo in una specifica Application all'interno del body del messaggio per il quale si vuole ottenere lo stream di eventi.

Quindi, chiusa questa parentesi, ci si può focalizzare sulla richiesta realmente utilizzata. Essa, come già anticipato nella sottosezione 2.3.1, consente di registrare i valori inseriti nel field-mask nel JS, e più precisamente nel JSEndDeviceRegistry, all'interno del quale sono salvati tutti i dispositivi OTAA.

La richiesta, come descritto nella sottosezione 3.2.1 necessita di un endpoint a cui deve giungere, quindi di un url, che in questo caso ha un formato del seguente tipo:

```
https://eu1.cloud.thethings.network/api/v3/js/applications/application-id/devices/device-id"
```

L'altro elemento necessario, oltre ovviamente all'API Key, è la stringa contenente i parametri, che costituiscono il body del messaggio, che come già anticipato devono essere in formato JSON. Un modello di body è stato già mostrato nella sottosezione 2.3.2.

I metodi successivamente implementati sono quelli necessari alla creazione e scrittura del file di configurazione.

createFile() consente di creare un file, se questo non esiste già, specificandone il percorso.

writeFile() scrive la stringa, che viene passata come argomento del metodo *write()* chiamato su un oggetto di classe FileWriter (libreria di Java),

sul file il cui percorso è specificato come attributo di questo stesso oggetto. La stringa, che quindi viene scritta sul file di configurazione, deve contenere identificatori, indirizzi e chiavi necessari per attivare l'ED, che sono gli stessi specificati nella sezione 3.1.

Infine. l'ultimo metodo di cui questa classe necessita, è quello che permette di richiamare le funzioni sopra citate ripetutamente nel tempo, grazie ad un timer. Nel programma utilizzato questo metodo è stato chiamato *updateKey(String key, String value, RootKeyGenerator newKey).*

Viene fatto uso della libreria Timer di Java. Essa permette di eseguire attività in background con un determinato ritardo o ripetutamente. A ciascun oggetto Timer corrisponde un singolo thread in background utilizzato per eseguire tutte le attività del timer, in sequenza. Le attività del timer dovrebbero essere completate rapidamente, perchè altrimenti se una prolungasse troppo la sua esecuzione porterebbe le altre a dover essere eseguite successivamente molto velocemente per essere completate entro i tempi richiesti. Inoltre questa classe è thread-safe: più thread possono condividere un singolo oggetto Timer senza la necessità di sincronizzazione esterna.

Quindi si inizializza un TimerTask che è una sorta di "ricetta" del timer, cioè contiene tutti gli step che devono essere messi in esecuzione in maniera ripetuta.

Ovviamente i metodi che devono essere chiamati più volte sono:

- metodo per la generazione della chiave AES-128
- metodo per inviare la richiesta PUT con il valore della chiava appena creata
- metodo per creare un file di configurazione, che se esiste già non viene creato
- metodo per scrivere gli elementi necessari per l'attivazione del dispositivo sul file. Se sul file ci sono già alcuni valori, questi vengono sovrascritti.

Viene riportato immediatamente sotto il codice del metodo *updateKey(String key, String value, RootKeyGenerator newKey)*.

```
1  public void updateKey(String key, String value,
       RootKeyGenerator
2  newKey) {
3      final Timer timer = new Timer();
4      // Initialize the TimerTask, this is like the recipe for
       the Timer
5      final TimerTask task = new TimerTask() {
6              // Function that will be ran in the TaskTimer
7              public void run() {
8                  //Code
9                String newKeyString = newKey.keyGenerator();
10               updateKeyPutRequest(newKeyString, key, value);
11               createFile();
12               writeFile(newKeyString);
13             }
14         };
15
16         long delay = 2000;
17
18           // Set the period (adjust this to your own needs)
19           long period = 50000;
20
21           // Schedule the TimerTask
22           timer.schedule(task, delay, period);
23   }
```

Listing 3.3: Metodo per l'aggiornamento periodico dell'AppKey

Appare scontato che la variabile *delay* indica il ritardo iniziale con cui vengono eseguiti i vari metodi, mentre *period* il tempo che intercorre tra una chiamata e l'altra.

3.3 Re-Join

Anche questo aspetto, riguardante il re-join è stato già trattato (si veda la sezione 2.3.3), e ne è stata quindi già spiegata l'utilità. Tuttavia in questa sezione ne vengono analizzati gli aspetti implementativi.

Per come il programma è stato strutturato, e per come sono stati implementati i dispositivi virtuali, il re-join deve essere fatto in maniera manuale. Quindi occorre spegnere il dispositivo e ricollegarlo alla rete manualmente.

Sebbene si tratti di ED virtuali utilizzati per testare il sistema, viene logico pensare che anche con dispositivi fisici, un utente decida dopo un certo periodo di tempo di spegnere l'ED e poi riaccenderlo, per cui il fatto che, affinchè l'aggiornamento delle chiavi sia sensato (quindi avvenga anche l'aggionramento delle chiavi di sessione), preveda un'operazione manuale, appare come uno scenario realistico. Comunque è sicuramente buona pratica riavviare un dispositivo elettronico periodicamente.

Nella sezione 1.1.3 è stato spiegato come attivare i dispositivi virtuali. Quindi, lasciando connesso il gateway, basta bloccare l'esecuzione del programma dell'ED (anche semplicemente con la combinazione *Control + C* se si fa uso del terminale) per poi farlo ripartire allo stesso modo.

Ciò non toglie che questa operazione possa essere automatizzata. Per far ciò però occorre cambiare il codice sorgente dei dispositivi virtuali, in maniera tale che possa interagire con il programma per l'aggiornamento chiavi. Così quest'ultimo, con poche righe di codice, può bloccare l'esecuzione del metodo *main()* dei dispositivi ogni volta che viene generata una nuova chiave, per poi riavviarlo caricando il file di configurazione aggiornato con il nuovo valore dell'AppKey.

Tuttavia cambiare il codice sorgente dei dispositivi virtuali non è prerogativa di questo progetto, motivo per cui l'operazione può essere effettuata solo manualmente.

Una volta che re-join è stato effettuato, si possono constatare i risultati ottenuti.

Capitolo 4

Valutazione Sperimentale

4.1 Test Effettuati

Per controllare che il sistema funzionasse correttamente e che quindi tutte le parti comunicassero tra loro così come dovrebbero, sono stati effettuati numerosi test.

Affinchè un test possa essere effettuato, la prima cosa da fare è connettere il gateway, così come spiegato nella sezione 3.1. Una volta aver constatato, tramite la Console di TTS, che il gateway è connesso, quindi lo stato diventa "Connected", occorre fare lo stesso per gli ED.

Una volta che un ED è connesso, sempre attraverso la Console, seguendo il percorso $Application \rightarrow myApplication - ID \rightarrow EndDevices \rightarrow MyEndDevice - ID$ è possibile visualizzare alcune informazioni relative al dispositivo finale, tra cui l'AppKey e le chiavi di sessione.

Constato il valore delle varie chiavi, si vede come, nella sezione *Live Data*, sia possibile leggere il flusso di messaggi che vengono scambiati tra il dispositivo e il gateway. *Live Data*, infatti, è presente sia per l'ED che per il gateway.

Tra le varie informazioni che si possono leggere sui messaggi, c'è anche lo stesso payload, che appare però criptato. Ovviamente il plaintext può essere estrapolato facendo uso proprio della chiave di sessione appSKey.

La Fig. 4.1 mostra uno screenshot della sezione *Live Data* di un ED, la Fig. 4.2 di un gateway, mentre la comunicazione è aperta, e quindi vi è uno scambio di messaggi.

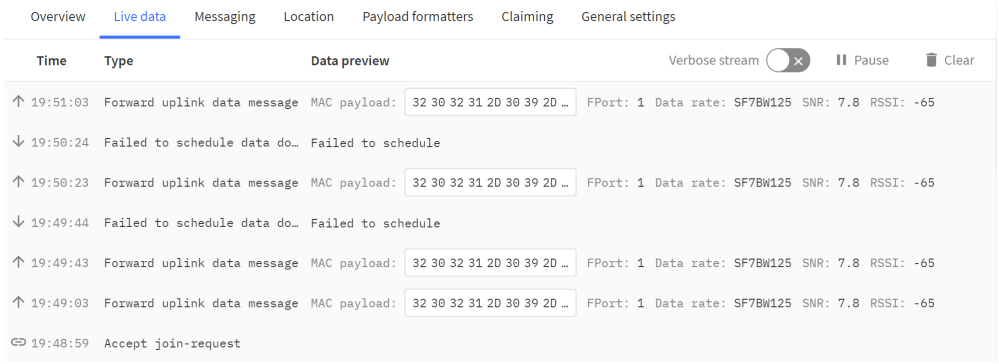

Figura 4.1: Sezione Live Data di un ED nella Console di TTS

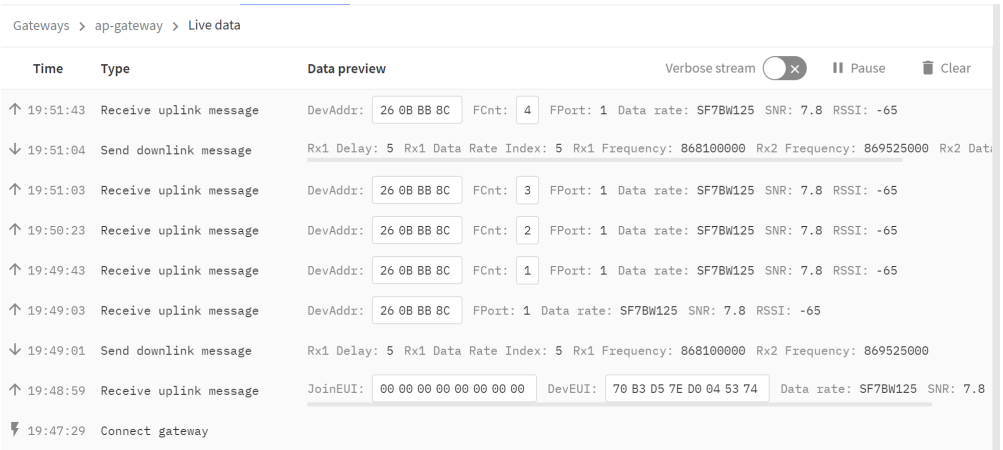

Figura 4.2: Sezione Live Data di un gateway nella Console di TTS

Una volta che viene avviato il programma per l'aggiornamento delle chiavi, sulla schermata mostrata in Fig. 4.1 viene notificato l'aggiornamento del dispositivo. Ciò può essere effettuato sia mentre la comunicazione è aperta, quindi mentre il dispositivo è accesso, sia mentre è spento.

Quindi controllando il valore dell'AppKey nella pagina dedicata all'End Device, si nota che questo è cambiato. Ovviamente, per i motivi già amplia-

mente analizzati, i valori delle chiavi di sessione rimangono gli stessi fino a che non viene effettuato nuovamente il join.

Dopo aver riavviato il dispositivo, se il join ha successo (ciò non sempre avviene, ma questo aspetto è descritto nella sezione 4.2), si può constatare che il valore delle chiavi di sessione è stato aggiornato, così come ci si aspettava.

Ovviamente, anche controllando il file di configurazione del dispositivo, sia prima che dopo l'avvio del programma, si nota che l'AppKey ha valori differenti.

Appare scontato che, se non viene fermata l'esecuzione del programma, il valore dell'AppKey continua a cambiare, sia nel file di configurazione, sia su TTS, ma finchè non si disconnette e si riconnette il dispositivo, questi cambiamenti non hanno alcun effetto.

4.2 Problematiche di Re-Join

Durante i test, sono state riscontrate alcune problematiche legate al re-join dei dispositivi.

Innanzitutto, come già spiegato nella sezione 3.1, facendo uso di dispositivi la cui versione di LoRaWAN è superiore alla 1.0.2, non è stato possibile effettuare il re-join, per via del campo DevNonce. Gli errori che vengono visualizzati sulla Console tentando di riconnettere dispositivi con versione di LoRaWAN superire alla 1.0.2 sono i seguenti.

- *DevNonce is too small*: indica che il valore del campo DevNonce è troppo piccolo, in quanto, non salvandolo al termine della precedente sessione, questo viene ricalcolato nuovamente, motivo per cui vi è un'incongruenza. Bisogna ricordare infatti che, come descritto nella sottosezione 1.1.3, la DevNonce fa parte del messaggio di Join-Request, quindi qualsiasi disuguaglianza in questo campo porterebbe a un messaggio di Join-Request non riconosciuto dal JS.

- *MIC mismatch*: il MIC (Message Integrity Code) è una sequenza di 4 bytes, che viene calcolata utilizzando, tra le altre cose, anche la De-

vNonce, motivo per cui anche questo errore è dovuto al mancato salvataggio di essa. Anche il MIC fa parte della Join-Request, quindi è fondamentale che sia corretto perchè il dispositivo possa riconnettersi alla rete.

Un'altra problematica riscontrata durante i test riguardante il re-join prescinde dalla versione di LoRaWAN cui fa capo il dispositivo. Sono diversi i tentativi di re-join per cui la Console ha notificato due tipi di errori:

- *Accept join-request*: a primo impatto non sembra essere un errore, sembra che la connessione alla rete sia riuscita, ma in realtà non inizia uno scambio di messaggi, e periodicamente appaiano altre notifiche di "Accept join-request" identiche.
- *Failed-to-schedule join-accept for trasmission on Gateway Server*: l'ED non riesce a schedulare il messaggio di Join Accept.

La Fig. 4.3 riporta, per comodità, il sequence diagarm dell'attivazione (OTAA) di un ED, che ovviamente vale anche in caso di re-join.

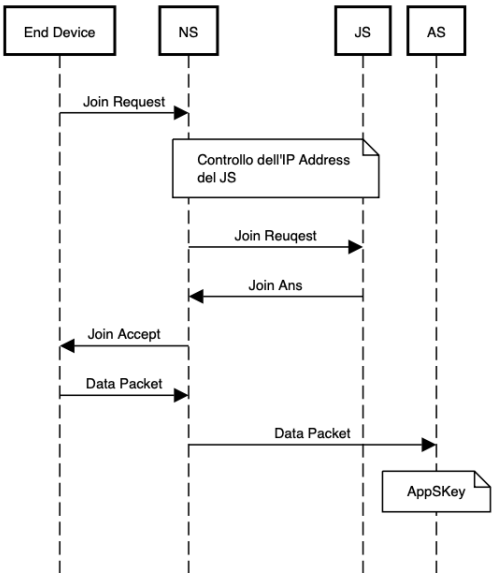

Figura 4.3: Sequence diagram del join di un ED

I numerosi test effettuati hanno portato a fare una considerazione sul motivo di questo errore. Innanzitutto, affinchè non vi siano problemi di re-join, bisogna attendere pochi secondi prima di riattivare il dispositivo, e non farlo in maniera immediata, così che tutte le procedure di disconnessione dalla rete siano ultimate.

In secondo luogo si è notato che gli errori sopra citati si verificano quando, nel momento in cui si disconnette l'ED, il gateway ha inviato un messaggio. In altre parole, nella sua sezione Live Data, come mostrato nel Fig. 4.2 riportata sopra (4.1), la prima riga mostra "Send downlink message".

Per cui, nel momento in cui si disconnette il dispositivo, se il gateway ha ricevuto un messaggio (nella sezione Live Data la prima riga riporta "Receive uplink message"), e si attendono alcuni secondi, non si riscontrano problemi in caso di riattivazione dell'ED. Quindi il re-join va a buon fine, e ED e gateway scambiano messaggi correttamente.

I problemi inerenti al re-join sono stati riscontrati anche in assenza del meccanismo di aggiornamento delle chiavi. I test per il funzionamento di questa procedura sono stati effettuati sia senza che con l'interazione con il programma che permette di cambiare l'AppKey.

4.3 Incremento del Livello di Sicurezza

Alla luce di ciò che è stato scritto nella sezione 4.2, nel momento in cui vengono rispettate tali accortezze per effettuare il re-join, il meccanismo di aggiornamento delle chiavi funziona correttamente e aumenta la sicurezza della rete.

Nella sezione 1.1.6 si parla, tra le altre cose, della necessità di utilizzare dei sistemi di archiviazione sicuri, e di non salvare le chiavi in semplici file, per evitare attacchi come il Plaintext Key Capture.

Nelle specifiche, viene menzionato che, oltre alle root-key, tutte le chiavi di sessione devono essere archiviate in modo sicuro, così da impedire l'estrazione e il riutilizzo di esse da parte di avversari. È possibile proteggere la memoria

non volatile dei dispositivi finali con tools hardware quali SE o HSM. Grazie al meccanismo di aggiornamento delle chiavi, che le rende dinamiche, si può sicuramente dare meno importanza a questi aspetti.

Anche in questo progetto, le chiavi dei dispositivi sono salvate in file di configurazione, che altro non sono che semplici file di testo. Il fatto di non doversi preoccupare di utilizzare sistemi di archiviazione a prova di attacco comporta una diminuzione sia della complessità, a livello implementativo, sia dei costi dell'hardware.

Ovviamente, rinunciare a uno di questi meccanismi affidandosi al sistema di aggiornamento delle root key equivale a scendere a compromessi. L'utilizzo in contemporanea di tools hardware, sistemi di archiviazione sicura e re-keying dinamico sicuramente porta ad un aumento notevole della sicurezza. Infatti le chiavi non solo sarebbero archiviate in maniera sicura, ma anche se qualcuno dovesse impossessarsene, il loro utilizzo sarebbe limitato solo a quella sessione, perchè al join successivo le chiavi di sessione sarebbero derivate da root key nuove, e così via per tutto il ciclo di vita del dispositivo. Trattandosi poi di chiavi generate da un algoritmo in Java, con metodi che fanno uso di diversi meccanismi di casualità, è impossibile prevedere qual è la prossima chiave che verrà generata.

Conclusioni

Alla luce dei test e delle osservazioni effettuate, si può concludere che il sistema realizzato fornisce da un lato un'alternativa ai meccanismi di sicurezza già presenti, dall'altro un incremento della sicurezza generale della rete.

Ricapitolando, quello che succede è che la root key AppKey viene sì stabilita in fase di fabbricazione (per così dire visto che si tratta di dispositivi virtuali), ma poi viene aggiornata dinamicamente. Tale cambiamento non ha alcun effetto sulla rete fino a quando non si effettua il re-join del dispositivo, così che possano essere rigenerate le chiavi di sessione, le quali vengono derivate non più sempre dalla stessa root key (che nella classica versione di LoRaWAN rimane invariata per tutto il ciclo di vita del dispositivo), ma da chiavi sempre diverse.

I prossimi step, per un futuro sviluppo del progetto, possono focalizzarsi sull'utilizzo del meccanismo di aggiornamento delle chiavi tramite l'API di TTS applicato su dispositivi fisici, non virtuali.

Un'altra possibile evoluzione può comprendere l'automatizzazione del processo di re-join del dispositivo, in maniera tale che l'utente non debba preoccuparsi di effettuarlo periodicamente in maniera manuale. Nel caso dei dispositivi virtuali questo cambiamento è facilmente applicabile, avendo a disposizione il codice sorgente di essi, mentre diventa appena più complesso con dispositivi reali. Bisogna anche tener conto dell'errore che si verifica in fase di re-join di cui si è discusso nella sezione 4.2, avendo cura di inserire anche controlli sui messaggi ricevuti e inviati dal gateway per stabilire qual è il momento corretto per effettuare la disconnessione e la riconnessione del

dispositivo.

Infine, quello che sarebbe lo sviluppo sicuramente più interessante, prevede la generazione di chiavi di sessione ogni qualvolta viene generato un nuovo valore di root key, indipendentemente dal re-join, tenendo quindi aperta la comunicazione con il server. Ovviamente questo meccanismo sarebbe ben più complesso di quello attuale, e richiederebbe un cambiamento alquanto significativo nelle norme di comunicazione di LoRaWAN.

Si può quindi concludere dicendo che il sistema realizzato, sebbene ancora in fase embrionale, si prefigura come un buon punto di partenza per diversi sviluppi futuri volti a migliorare la sicurezza di dispositivi IoT di tipo LPWAN che sono sempre più presenti nella vita di tutti i giorni e la cui diffusione non accenna a diminuire.

Bibliografia

[1] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. A comparative study of lpwan technologies for large-scale iot deployment. *ICT Express*, 5(1):1–7, 2019.

[2] Ismail Butun, Nuno Pereira, and Mikael Gidlund. Security risk analysis of lorawan and future directions. *Future Internet*, 11(1), 2019.

[3] LoRa Alliance Technical Marketing Workgroup. *LoRaWAN What is it?* LoRa Alliance, November 2015.

[4] Semtech Corporation. *LoRa and LoRaWAN: A Technical Overview*, December 2019.

[5] The Things Industries. Everything you need to know about lorawan in 60 minutes - johan stokking (the things industries). `https://www.youtube.com/watch?v=ZsVhYiX4_6o&t=1s`, 2021.

[6] The Thing Industries. End device activation, 2021.

[7] The Thing Industries. What is the things stack? `https://www.thethingsindustries.com/docs/getting-started/what-is-tts/`, 2021.

[8] The Thing Industries. Using the api. `https://www.thethingsindustries.com/docs/getting-started/api/`, 2021.

[9] The Thing Industries. Authentication. https://www.thethingsindustries.com/docs/reference/api/authentication/, 2021.