Formal Security Analysis of ZigBee Protocol

Sebastiano Morson

August 3, 2025

Abstract

Contents

1 Introduzione

L'ambito dell'Internet of Things (IoT) abbraccia milioni di oggetti dalle diverse caratteristiche e necessità. Per far questi oggetti comunicare tra loro sono stati progettati dei protocolli di comunicazione che tenessero conto dell'eterogeneità della rete sia in termini di richiesta di risorse da parte dei nodi sia rispetto alla loro ubicazione. Tra questi protocolli emerge Zigbee per il basso consumo di potenza e la capacità di fornire reti mesh sicure, affidabili ed estensibili. L'ampia diffusione di Zigbee ha tuttavia implicato la necessità di un'indagine sulle vulnerabilità e sulle proprietà di sicurezza che il protocollo deve assicurare [12]. La verifica formale è uno strumento fondamentale per garantire la sicurezza dei protocolli perché permette di trovare eventuali difetti prima che vengano sfruttati da potenziali attaccanti. In questo lavoro, analizzo la sicurezza del protocollo Zigbee con Tamarin-Prover, uno strumento per la verifica formale dei protocolli crittografici [5]. Tamarin-Prover consente di modellare il comportamento del protocollo e di verificare proprietà di sicurezza fondamentali quali segretezza e autenticazione. Attraverso questa analisi si vuole verificare se Zigbee è conforme agli standard di sicurezza. L'artigolo si sviluppa in più sezioni. In primis viene fatta una descrizione del protocollo Zigbee presentandone l'architettura a strati, i tipi di dispositivi supportatti, le topologie di rete di riferimento e la gestione delle chiavi di cifratura. A seguire, faccio una carrellata delle principali minacce alla sicurezza del protocollo e delle proprietà che il protocollo deve soddisfare. Proseguo poi con una panoramica su Tamarin-Prover spiegando come vengono formalizzati i protocolli all'interno dello strumento. A questo punto, espongo la modellazione del protocollo Zigbee in Tamarin. Procedo quindi a formalizzare le proprietà di sicurezza e descrivo le regole utilizzate per verificare le stesse. Infine discuto i risultati dell'analisi mettendo in evidenza le eventuali vulnerabilità scoperte.

2 Overview del protocollo ZigBee

Zigbee è un protocollo di comunicazione progettato per reti wireless a basso consumo energetico, ampiamente adottato nell'ecosistema dell'Internet of Things (IoT)[4][7][8]. La sua architettura garantisce flessibilità e scalabilità, consentendo la comunicazione tra dispositivi con risorse limitate. Questo capitolo fornisce una panoramica del protocollo, analizzandone la struttura dello stack, i tipi di dispositivi supportati, le possibili topologie di rete ed i meccanismi di gestione delle chiavi crittografiche.

2.1 Stack architecture

L'architettura dello stack Zigbee si basa su livelli distinti che gestiscono specifiche funzionalità [2]. In figura 1 è illustrato lo schema dei vari layer. Ci concentreremo sui livelli Network e Application, tralasciando i dettagli dei livelli fisici (IEEE 802.15.4).

Livello Network (NWK): Questo livello è responsabile della formazione, gestione e mantenimento della rete. Gestisce l'indirizzamento dei dispositivi, l'instradamento dei pacchetti e l'implementazione delle politiche di sicurezza a livello di rete. Il livello NWK consente la creazione di topologie di rete come stella, albero e mesh.

Livello Application (APL): Il livello Application comprende diversi componenti chiave:

Figure 1.1 Outline of the ZigBee Stack Architecture

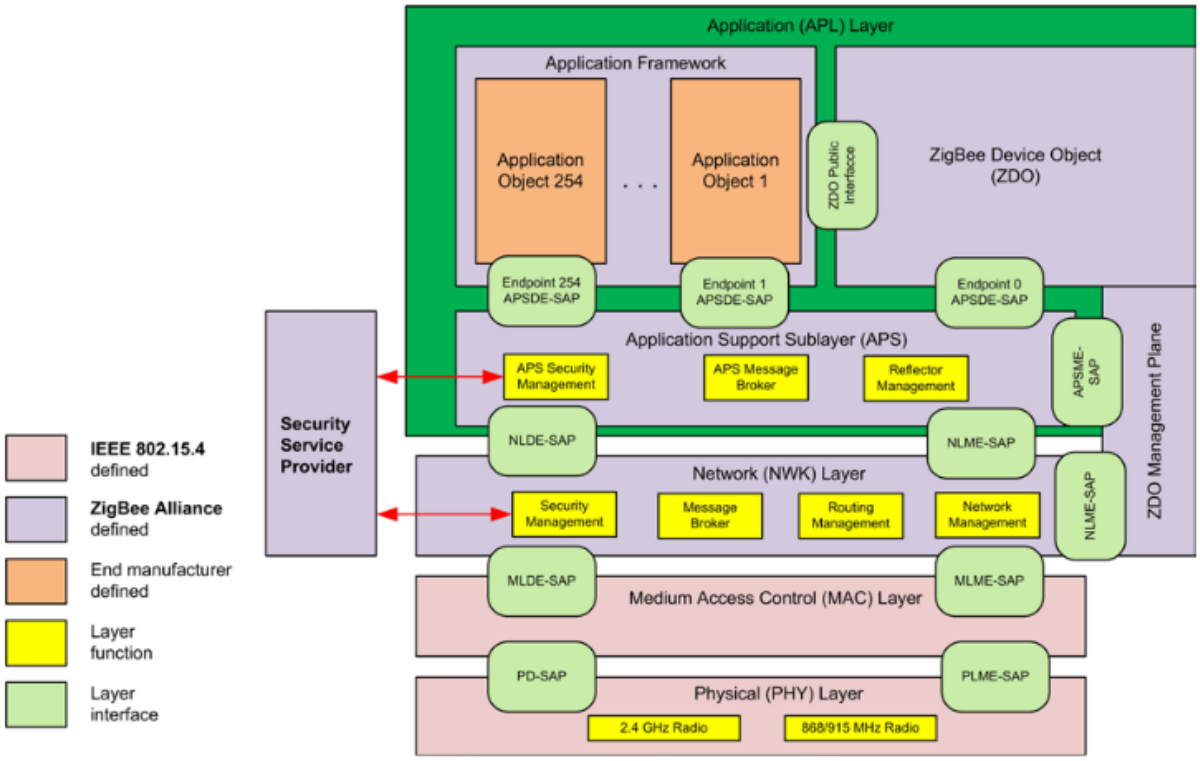

Figure 1: ZigBee stack architecture

1. **Application Support Sublayer (APS):** Fornisce servizi per la gestione delle comunicazioni tra le applicazioni, inclusa la gestione delle tabelle di binding che determinano quali dispositivi possono comunicare tra loro.

2. **Zigbee Device Object (ZDO):** Gestisce le funzionalità di dispositivo, come la scoperta e la gestione dei servizi, e coordina le operazioni di associazione e disassociazione dei dispositivi nella rete.

3. **Application Framework:** Consente agli sviluppatori di definire profili applicativi e cluster specifici per le esigenze dell'applicazione, facilitando l'interoperabilità tra dispositivi di diversi produttori.

2.2 Confronto con Bluetooth e Wifi

ZigBee, Bluetooth e Wi-Fi sono tre protocolli di comunicazione wireless progettati per scopi differenti. Rispetto alle altre due tecnologie, ZigBee si distingue soprattutto per il suo basso consumo energetico, che lo rende ideale per dispositivi che devono operare per lunghi periodi senza rimpiazzare la batteria, come monitoragigo ambientale, sistemi di domotica e dispositivi IoT in generale.

A differenza del Bluetooth, che consuma più energia (circa 60 mA in trasmissione), ZigBee consuma appena 25–35 mA in trasmissione e addirittura solo 3 μA in modalità standby. Questo è possibile in quanto Zigbee è pensato per dispositivi che genealmente operano inviando tanti pacchetti di piccola dimensione, perciò il protocollo permette una velocità di trasmissione più bassa (250 Kbps contro 1 Mbps del Bluetooth e 11 Mbps del Wi-Fi).

Wi-Fi, invece, è progettato per trasmissioni ad alta velocità e alta capacità (es. video streaming, navigazione web), ma ciò comporta un consumo energetico elevato (fino a 400 mA in trasmissione) e una maggiore complessità dello stack protocollare (fino a 1 MB contro 32–250 KB di ZigBee).

Dal punto di vista della scalabilità, ZigBee può gestire fino a 64.000 nodi per rete, grazie al suo supporto per topologie mesh. In confronto, Bluetooth classico è limitato a 7 dispositivi attivi per piconet, e Wi-Fi può gestire circa 32 dispositivi per access point.

Infine, ZigBee è pensato per tempi di associazione molto rapidi (tipicamente 30 ms), ideale per dispositivi che si attivano solo in presenza di eventi. Bluetooth e Wi-Fi richiedono invece tempi di connessione più lunghi (Bluetooth $>$ 3 sec; Wi-Fi 1 sec). Tabella 2.2.

Wireless Parameter	**Bluetooth**	**Wi-Fi**	**ZigBee**
Frequency band	2.4 GHz	2.4 GHz	2.4 GHz
Physical/MAC layers	IEEE 802.15.1	IEEE 802.11b	IEEE 802.15.4
Range	9 m	75 to 90 m	Indoors: up to 30 m Outdoors (line of sight): up to 100 m
Current consumption	60 mA (Tx mode)	400 mA (Tx mode) 20 mA (Standby mode)	25–35 mA (Tx mode) 3 μA (Standby mode)
Raw data rate	1 Mbps	11 Mbps	250 Kbps
Protocol stack size	250 KB	1 MB	32 KB 4 KB (for limited function end devices)
Typical network join time	>3 sec	Variable, typically 1 sec	Typically 30 ms
Maximum number of nodes per network	7	32 per access point	64 K

Table 1: Comparison of Wireless Standards: Bluetooth, Wi-Fi, and ZigBee

2.3 Tipi di Dispositivi Zigbee:

Nelle reti Zigbee, esistono tre principali tipologie di dispositivi, ciascuna con ruoli e responsabilità specifiche:

1. **Zigbee Coordinator (ZC):** È il dispositivo principale della rete Zigbee, responsabile dell'avvio e della gestione della rete come l'assegnamento degli indirizzi ai dispositivi che si uniscono alla rete, il mantenimento delle informazioni sulla topologia e sullo stato dei dispositivi. In una rete Zigbee (non distribuita), può esistere un solo coordinatore.

2. **Zigbee Router (ZR):** Questi dispositivi estendono la copertura della rete fungendo da intermediari nell'instradamento dei pacchetti. I router possono comunicare con il coordinatore, con altri router e con dispositivi finali, facilitando la comunicazione in reti di grandi dimensioni o in ambienti conn ostacoli fisici.

3. **Zigbee End Device (ZED):** Sono dispositivi con funzionalità limitate che comunicano esclusivamente con un coordinatore o un router. Non partecipano all'instradamento dei pacchetti e sono spesso progettati per operare con un consumo energetico minimo

2.4 Topologie di Rete Supportate:

Zigbee supporta diverse topologie di rete, adattandosi a varie esigenze applicative:

- **Topologia a Stella:** Tutti i dispositivi finali comunicano direttamente con il coordinatore. Questa struttura è semplice ma limita l'estensione della rete.
- **Topologia ad Albero:** Il coordinatore funge da nodo radice, con router e dispositivi finali organizzati in una struttura gerarchica. Questa topologia facilita l'estensione della rete in modo organizzato.
- **Topologia Mesh:** Ogni dispositivo può comunicare con altri dispositivi nel raggio d'azione, permettendo instradamenti multipli e aumentando la resilienza della rete. La topologia mesh è particolarmente vantaggiosa in ambienti dove l'affidabilità della comunicazione è cruciale.

2.5 Esempio di Rete Zigbee Reale:

Se si considera un'abitazione domotizzata è facile che al suo interno sia presente una rete ZigBee organizzata in questo modo:

- Zigbee Coordinator (ZC): Un hub centrale, come Amazon Echo Plus o SmartThings Hub, che avvia e gestisce la rete Zigbee.
- Zigbee Router (ZR): Lampadine intelligenti distribuite in varie stanze che, oltre a fornire illuminazione, estendono la copertura della rete facilitando la comunicazione tra dispositivi distanti.
- Zigbee End Device (ZED): Sensori di temperatura e umidità alimentati a batteria che monitorano le condizioni ambientali e comunicano periodicamente i dati al coordinatore tramite i router.

In questa configurazione, l'hub centrale (ZC) gestisce la rete e coordina le comunicazioni. Le lampadine intelligenti (ZR) non solo ricevono comandi dall'hub ma fungono anche da intermediari per i sensori (ZED), garantendo che i dati dei sensori raggiungano l'hub anche se questi sono fuori dal raggio diretto del coordinatore. Questo esempio illustra come i diversi tipi di dispositivi collaborino per creare una rete Zigbee efficiente e affidabile.

2.6 Panoramica delle chiavi crittografiche e gestione delle sessioni

Il protocollo ZigBee adotta un modello di sicurezza basato su chiavi simmetriche a 128 bit, progettato per garantire confidenzialità, integrità e autenticazione dei dati. Questo modello, introdotto con ZigBee 1.0 e mantenuto in ZigBee 3.0, si compone di due livelli principali: sicurezza a livello di rete (NWK) e a livello applicativo (APS). Le chiavi utilizzate sono di tre tipi principali: chiave di rete, chiave di collegamento con il Trust Center e chiave di collegamento applicativa, ciascuna con funzioni e modalità di gestione specifiche [2][1][3].

2.6.1 Network Key

La chiave di rete è utilizzata per crittografare e autenticare i messaggi a livello di rete. Tutti i dispositivi autorizzati a far parte della rete condividono questa chiave, che viene aggiornata periodicamente dal Trust Center per mitigare i rischi di compromissione. Ogni aggiornamento è identificato da un numero di sequenza (da 0 a 255), che permette ai dispositivi di riconoscere la versione corretta della chiave da utilizzare.

2.6.2 Trust Center Link Key

Questa chiave è utilizzata per comunicazioni sicure end-to-end tra un dispositivo e il Trust Center. Le sue funzioni principali sono:

1. crittografare il trasferimento iniziale della chiave di rete a un dispositivo che si unisce alla rete
2. gestire aggiornamenti della chiave di rete per dispositivi che si riconnettono

3. proteggere messaggi di sicurezza APS tra router e Trust Center

Il Trust Center può gestire queste chiavi in modo centralizzato, utilizzando chiavi uniche per ogni dispositivo, chiavi derivate da dati condivisi (come l'indirizzo IEEE) o una chiave globale comune a tutti i dispositivi. A partire dalla versione R23, è stato introdotto il supporto per le Dynamic Link Keys (DLK), che utilizzano la negoziazione di chiavi basata su curve ellittiche per stabilire chiavi temporanee e autenticare i dispositivi prima del loro ingresso nella rete.

2.6.3 Application Link Keys

Oltre alla chiave di collegamento con il Trust Center, i dispositivi possono stabilire chiavi di collegamento applicative per comunicazioni sicure tra di loro, senza coinvolgere il Trust Center. Queste chiavi possono essere configurate manualmente dall'applicazione o generate dal Trust Center su richiesta. Nel secondo caso, il Trust Center genera una chiave casuale e la invia a entrambi i dispositivi coinvolti nella comunicazione. Questo meccanismo aggiunge un ulteriore livello di sicurezza per le comunicazioni dirette tra dispositivi.

3 Minacce e Proprietà di Sicurezza di Zigbee

In questa sezione analizziamo le principali minacce alla sicurezza delle comunicazioni Zigbee e le proprietà di sicurezza che devono essere verificate per garantire un funzionamento sicuro del protocollo.

3.1 Minacce Comuni nei Protocolli Zigbee

Le seguenti minacce rappresentano alcuni dei principali rischi per la sicurezza di Zigbee:

- **Eavesdropping**: Poichè Zigbee utilizza la comunicazione wireless, un attaccante potrebbe intercettare i messaggi trasmessi per ottenere informazioni sensibili, come chiavi di crittografia o dati di controllo.
- **Replay Attack**: Un attaccante potrebbe registrare un messaggio valido e ritrasmetterlo successivamente per ingannare i dispositivi nella rete, eseguendo operazioni non autorizzate.
- **Key Compromise**:Un attaccante potrebbe compromettere le chiavi crittografiche utilizzate nel protocollo, diventando in grado di decifrare le comunicazioni, impersonare dispositivi legittimi o modificare i messaggi trasmessi.

3.2 Proprietà di Sicurezza Attese

Per mitigare le minacce precedentemente descritte, un protocollo Zigbee sicuro deve garantire le seguenti proprietà di sicurezza fondamentali:

3.2.1 Autenticazione

L'autenticazione assicura che i dispositivi comunicanti siano effettivamente quelli dichiarati, impedendo attacchi di impersonificazione. Utilizziamo il modello di Lowe [10] per formalizzare le proprietà di autenticazione:

1. **Aliveness**: Un'entità deve essere sicura che l'altra parte ha partecipato alla sessione di comunicazione.
2. **Weak Agreement**: Le due entità coinvolte concordano sull'aver comunicato tra loro, ma non necessariamente sugli esatti dettagli della comunicazione.
3. **Non-Injective Agreement**: Le due entità concordano sulla comunicazione e sugli identificatori della sessione, sebbene possano verificarsi sessioni duplicate.
4. **Injective Agreement**: Simile alla non-injective agreement, ma con la garanzia che ogni sessione sia unica e non venga riutilizzata.

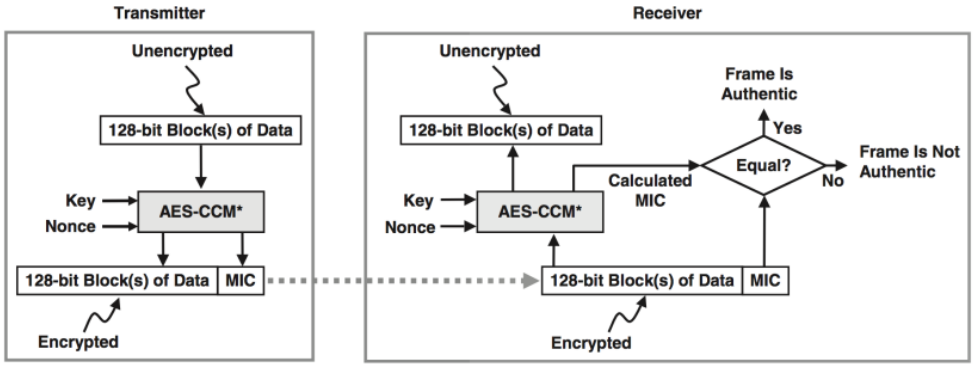

Figure 2: AES-CCM*

3.2.2 Confidenzialità

La confidenzialità garantisce che le informazioni trasmesse non siano accessibili a entità non autorizzate. Per realizzare questa proprietà, Zigbee adotta la suite crittografica AES-CCM*, una variante modificata della modalità CCM (Counter con CBC-MAC) applicata all'AES-128. La modalità CCM* integra in un'unica procedura crittografica la cifratura dei dati, l'autenticazione e la verifica di integrità, supportando anche operazioni in modalità "solo cifratura" o "solo autenticazione" (figura 2).

In fase di trasmissione, il messaggio in chiaro viene suddiviso in blocchi da 128 bit ed elaborato dall'AES-CCM*, che produce il testo cifrato accompagnato da un Message Integrity Code (MIC). Il destinatario, a sua volta, utilizza la stessa procedura per decifrare i dati e genera un MIC indipendente da confrontare con quello ricevuto. Se i due valori coincidono, il messaggio è considerato autentico e integro.

Il processo utilizza un **nonce** di 13 byte costruito a partire dal campo di controllo di sicurezza, il frame counter e l'indirizzo sorgente dell'intestazione ausiliaria. Il MIC può avere una lunghezza di 32, 64 o 128 bit, e fornisce una garanzia più forte di autenticità rispetto a meccanismi più semplici come il CRC.

Poiché AES-CCM* è formalmente riconosciuto come schema sicuro per la cifratura simmetrica autenticata, in questa trattazione si assume che la sua robustezza non rappresenti un punto critico. La nostra analisi si concentra pertanto sulla corretta gestione delle chiavi e sulla prevenzione di attacchi legati alla riutilizzazione del nonce.

3.2.3 Integrità

L'integrità garantisce che i messaggi ricevuti non siano stati alterati durante la trasmissione. Zigbee assicura questa proprietà attraverso due meccanismi complementari:

- **MIC (Message Integrity Code)**: Un codice crittografico calcolato dal mittente e verificato dal destinatario, progettato per rilevare modifiche intenzionali o accidentali ai messaggi. In Zigbee, il MIC è prodotto come parte integrante dell'operazione AES-CCM*, garantendo coerenza tra cifratura e verifica d'integrità.
- **Frame Counter**: Un contatore incrementale incluso nei messaggi cifrati, utilizzato per prevenire attacchi di replay e garantire l'ordine e l'unicità delle trasmissioni. Il frame counter contribuisce anche alla generazione del nonce, e il suo riutilizzo può compromettere la sicurezza del canale cifrato.

Il MIC generato tramite CCM* offre una protezione più avanzata rispetto ai checksum tradizionali, poiché è resistente sia a errori involontari che a manipolazioni malevole dei dati.

3.3 Specifiche da verificare

Tra i vari protocolli che compongono Zigbee, il paper di Li,Podder e Hoque [9] si concentra sulla verifica formale di quattro in particolare.

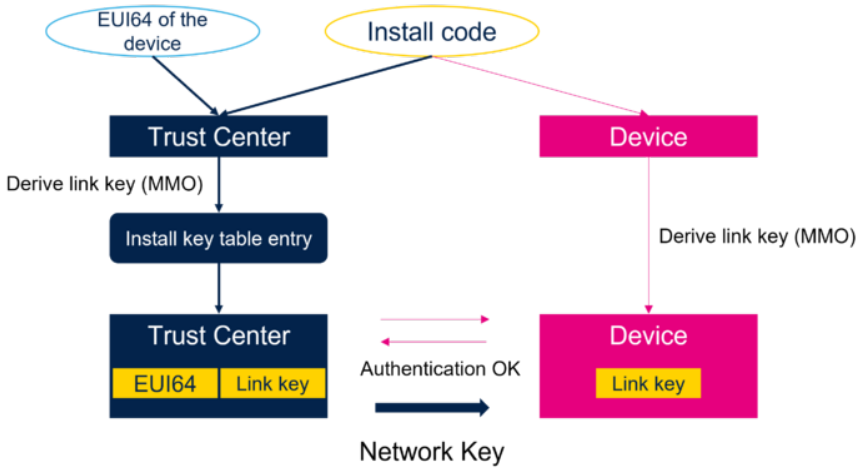

Figure 3: Installation code generation

3.3.1 Network Key Sharing

Nel momento in cui un dispositivo richiede l'accesso alla rete e viene autorizzato, riceve dal coordinator una network key. La trasmissione della network key da trust center a end-device avviene cifrando la network key con la trust center link key. Per generare questa chiave sono definiti due metodi:

- **pre-configured global key**: questa era la soluzione adottata da Zigbee 1.0. Si basa sull'utilizzo di una chiave globale condivisa tra device e coordinator (5A 69 67 42 65 65 41 6C 6C 69 61 6E 63 65 30 39). Ovviamente se l'attaccante viene a conoscenza della chiave globale è in grado di ottenere la network key.
- **install code**: in ZigBee 3.0, è stato introdotto l'uso di chiavi derivate da un codice di installazione. Il codice di installazione, composto da 6, 8, 12 o 16 byte più un CRC a 16 bit, è condiviso tra trust center e device viene elaborato da entrambe le parti tramite una funzione di hash AES-MMO per generare una chiave di collegamento preconfigurata. L'install code, insieme all'indirizzo EUI64 del dispositivo, deve essere comunicato al Trust Center fuori banda (ad esempio, tramite telefono o Internet) per consentire l'autenticazione sicura durante il joining nella rete.

  In figura 3 si può vedere uno schema di come viene derivata la pre-configured key attraverso l'install-code da parte di entrambe le parti coinvolte.

3.3.2 Joining a secure network

Per accedere a una rete un device deve ricevere il consenso del parent attraverso una beacon request. Se viene accettata, il device passa allo stato joined. Alla ricezione della MAC association request notifica al trust center la richiesta del device. A questo punto il trust center può richiedere nuove informazioni al device prima completare l'autenticazione. Se il trust center riconosce il dispositivo, autentica il device e invia la Network Key cifrata attraverso la Trust Center Link Key. In figura 4 si può vedere lo schema di joining.

Dopo essere stato autenticato, le specifiche di Zigbee 3.0 stabiliscono che è necessario che il device faccia richiesta di update della Trust Center Link Key. In figura 5 si può osservare il comportamento adottato dalle due parti durante questa fase.

3.3.3 Application Key Establishment

Affinchè due device istanzino una comunicazione end-to-end sicura, Zigbee prevede l'utilizzo di Application Key da usare per cifrare le comunicazioni. Le Application Key sono univoche per ciascuna coppia di devices. Il modello usato per stabilire l'Application Key è il seguente:

1. Un initiator invia una richiesta Request Key al coordinator con Key Type "Application Link Key"
2. Il coordinator risponde all'initiator con il comando Transport Key che contiene l'Application Key
3. Il coordinator condivide con il secondo device l'Application Key

3.3.4 Network Key Update

Al fine di evitare replay attack, ogni pacchetto del Network Layer presenta un Frame Counter all'interno dell'Auxiliary Header. Questo frame counter assume il valore massimo di 0xFFFFFFFF, superato il quale dev'essere azzerato prima di poter procedere all'invio di nuovi messaggi. Per poter essere azzerato è necessario che avvenga un Network Key update e il valore del NwK frame counter sia superiore a 0x80000000.

Il Trust Center può distribuire la nuova chiave di rete in due modalità:

- Broadcast: la nuova chiave viene inviata a tutti i dispositivi, crittografata con la chiave di rete precedente.
- Unicast: la nuova chiave viene inviata singolarmente a ciascun dispositivo, crittografata con la rispettiva chiave di collegamento con il Trust Center.

Dopo la distribuzione, il Trust Center comunica attraverso il comando Key Switch ai dispositivi di passare alla nuova chiave, incrementando il numero di sequenza. Questo meccanismo garantisce che la rete rimanga sicura anche in caso di aggiornamenti periodici della chiave.

In caso un dispositivo perda un aggiornamento della network key il device viene essere buttato fuori dalla rete e ripeterà un trust center rejoin.

4 Introduzione a Tamarin Prover

Per verificare la sicurezza delle specifiche introdotte nel paragrafo 3.3, il lavoro di Li et al. utilizza lo strumento Tamarin Prover [9][5]. Da questo punto in avanti, farò riferimento a Tamarin Prover con la forma abbreviata *Tamarin*.

Tamarin è uno strumento per la verifica formale basata su modellazione simbolica, progettato per analizzare la sicurezza dei protocolli crittografici. Dato in input un modello del protocollo e un insieme di proprietà di sicurezza da verificare, Tamarin è in grado di determinare se esistono esecuzioni che violano tali proprietà, fornendo eventualmente un controesempio.

Nei paragrafi seguenti, illustrerò il processo di definizione del protocollo e delle proprietà di sicurezza nel contesto di Tamarin.

4.1 Specifica di un protocollo in Tamarin

In Tamarin, la modellazione di un protocollo e delle capacità di un avversario avviene tramite la definizione di rewriting rules (regole di riscrittura). Queste regole descrivono come il sistema evolve nel tempo, modificando lo stato rappresentato da un multi-set di fatti. Gli ingredienti di base delle regole di riscrittura sono:

- Termini: Rappresentano messaggi, bitstring o variabili utilizzati nel protocollo.
- Fatti: Utilizzati per definire informazioni sullo stato attuale del sistema, eseguire controlli su tali informazioni e registrare azioni eseguite.
- Stati del sistema: Sono rappresentati da multi-set di fatti, che descrivono la configurazione del sistema in un dato momento.

L'insieme delle regole di riscrittura definisce un labeled transition system su cui Tamarin esegue la verifica delle proprietà specificate dall'utente, come segretezza, autenticazione o integrità.

4.1.1 Terms

I termini in Tamarin rappresentano i dati manipolati dal sistema, come messaggi, chiavi, nonce o variabili. Possono essere:

- Costanti: Valori fissi, come alice, bob, k1.
- Variabili: Simboli che possono assumere diversi valori, come x, y, k.
- Funzioni: Applicazioni di funzioni a termini, come enc(m, k) (cifratura del messaggio m con la chiave k) o hash(m) (hashing del messaggio m).

La costruzione dei termini è regolata dalla *term algebra*. I termini servono per costruire fatti e regole. Nel primo caso possiamo utilizzare solo ground terms, nel secondo invece possono esserci sia ground terms che termini con variabili.

I termini possono riferirsi a secrets(preceduti dal simbolo $\sim$ o public(preceduti dal simbolo $)

4.1.2 Facts

I fatti sono i mattoni con cui è costruito lo stato del sistema. Essi rappresentano informazioni quali:

- stato: per esempio, Key(alice, k1) indica che Alice possiede la chiave k1.
- eventi: per esempio, ReceivedMessage(m1) registra che il messaggio m1 è stato ricevuto.
- controlli: per esempio, In(m1) indica che il messaggio m1 è disponibile come input per l'avversario.

I fatti possono essere:

- temporanei: consumati quando una regola viene applicata.
- persistenti: non vengono consumati e rimangono nel multi-set anche dopo che una regola viene applicata. Sono preceduti da (!)

I principali built-in facts sono: In(x), Out(x), K(x), Fr(x). I significati di questi built-int facts sono rispettivamente la presenza sul canale di un termine x, l'immissione sul canale di un termine x, la conoscenza da parte dell'attaccante di un termine x, la generazione fresh di un nuovo termine x.

4.1.3 Rewriting rules

Le rewriting rules si scrivono in questa forma

$$L - [A] -> R$$

in cui:

- L è un multi-set di fatti che definisce le precondizioni
- A è un multi-set di fatti che definisce le azioni (etichette per la regola, servono per tracciare le transizioni)
- R è un multi-set di fatti che definisce le conclusioni

L'insieme delle multi-set rewriting rules e degli stati ($\Gamma_t = \{F_0, F_1, ..., F_n\}$ con F_i un multi-set of facts), definisce un labeled transition system.

Data una multi-set rewriting rule R, un multi-set di fatti $S_t = \{F_0, \ldots, F_n\}$ e una sequenza di multi-set trace $T_t =< a_0, \ldots, a_{t-1} >$ al tempo t:

1. possiamo applicare R a S_t se e solo se esiste una ground instance $l - [a_t] \to r$ tale che $l \subseteq^{\#} S_t$

2. dopo l'applicazione della regola R abbiamo che

$$S_{t+1} = S_t \backslash^{\#} l \ \cup^{\#} r \tag{1}$$
$$T_{t+1} =< a_0, ..., a_{t-1}, a_t > \tag{2}$$

L'applicazione delle regole agli stati del sistema a volte può richiedere operazioni di sostituzione e unificazione [11].

Tamarin prevede alcune regole built-in:

- Out(x) -[]→ K(x) (l'avversario ricava un'informazione)
- K(x) -[K(x)]→ In(x) (l'avversario inietta un messaggio)
- ∅-[]→ Fr(x) (generate fresh value)

4.1.4 Teorie Equazionali

Il meccanismo di ricerca delle dimostrazioni simboliche di Tamarin supporta una classe particolare di teorie equazionali, chiamate convergent equational theories with the finite variant property [6]. Queste sono le teorie necessarie per modellare proprietà crittografiche e algebraiche dei protocolli. Nonostante Tamarin non verifichi che le equazioni fornite siano effettivamente in questa classe, definire equazioni che non appartengano a questa famiglia può portare lo strumento alla non terminazione o alla restituzione di risultati non corretti senza avvertimenti. Le proprietà principali delle teorie equazionali convergenti sono:

Confluenza: Una teoria equazionale convergente è *confluente*, ovvero se applichiamo le equazioni in qualsiasi ordine otterremo sempre la stessa forma normale. Questo garantisce che le rappresentazioni simboliche dei termini siano coerenti nella ricerca delle dimostrazioni. All'interno di una teoria equazionale confluente, se da un termine t posso ridurmi a due termini t_1 e t_2, e da t_1 posso ridurmi a t_3, allora anche da t_2 posso ridurmi a t_3.

Terminazione: La teoria deve essere *terminante*, garantendo che il processo di riscrittura dei termini mediante le equazioni raggiunga sempre una forma normale. Questo assicura che il processo di ricerca delle dimostrazioni non entri in loop infiniti. Ad esempio, la regola $x * y = y * x$ non garantisce la terminazione.

Proprietà della Variante Finita: La teoria deve possedere la *proprietà della variante finita*, che assicura che qualsiasi termine possa essere riscritto in un insieme finito di varianti possibili. Questa proprietà è cruciale per l'efficienza del ragionamento simbolico di Tamarin, poiché consente allo strumento di gestire le teorie equazionali in modo computazionalmente trattabile.

4.1.5 Trace

Una traccia rappresenta una sequenza di eventi (applicazioni di regole) che descrivono l'esecuzione del sistema. Ogni traccia è una possibile esecuzione del protocollo, e Tamarin verifica le proprietà di sicurezza su tutte le tracce possibili. Le tracce includono:

- Eventi: Azioni eseguite, come l'invio o la ricezione di un messaggio.
- Etichette: Informazioni aggiuntive associate agli eventi, come Send(m1) o Receive(m1).

Le proprietà di sicurezza sono espresse come formule logiche sulle tracce, ad esempio: "Non esiste una traccia in cui un messaggio segreto viene rivelato all'avversario."

Dato un transition system P, l'insieme delle possibili esecuzioni su P è definita come:

$$trace(P) = \{< a_1, \ldots, a_n \mid \exists S_1 \ldots \exists S_n (\emptyset \rightarrow_{a_0} S_1 \rightarrow_{a_1} S_2 \rightarrow ... \rightarrow_{a_{n-1}} S_n\}$$

4.1.6 Lemma

Come anticipato in precedenza, in Tamarin, le proprietà di sicurezza sono espresse come frammenti guardati della logica del primo ordine. Il costrutto sintattico adottato da Tamarin per indicare una proprietà da dimostrare è il **lemma**. I lemmi possono essere di due tipi:

- **Exists-trace**: Questi lemmi vengono dimostrati trovando una singola traccia valida del protocollo che soddisfi la formula.
- **All-traces**: Questi lemmi vengono dimostrati negando la proprietà e cercando di trovare un controesempio. Se non viene trovato alcun controesempio, la proprietà è valida per tutte le tracce.

Per specificare un lemma, possiamo combinare i seguenti atomi:

- La costante **false** ($\bot$);
- Operatori logici: $\neg$ (negazione), $\wedge$ (congiunzione), $\vee$ (disgiunzione), $\implies$ (implicazione);
- Quantificatori e variabili: $\forall$ (quantificatore universale), $\exists$ (quantificatore esistenziale) e variabili come a, b, c;
- Uguaglianza tra termini: $t_1 \approx t_2$
- Ordinamento e uguaglianza dei punti temporali: $i \leq j$ e $i = j$;
- Fatti d'azione in punti temporali: F@i (per un fatto d'azione F al punto temporale i).

La semantica delle trace formulas, su una traccia T e una valutazione Θ è:

$$
\begin{aligned}
T, \Theta \vDash F@i &\iff 1 \leq \Theta(i) \leq n \wedge \Theta(F) \in T[\Theta(i)] && (3)\\
T, \Theta \vDash i \dot{<} j &\iff \Theta(i) < \Theta(j) && (4)\\
T, \Theta \vDash i \doteq j &\iff \Theta(i) = \Theta(j) && (5)\\
T, \Theta \vDash t_1 \approx t_2 &\iff \Theta(i) \approx \Theta(j) && (6)\\
T, \Theta \vDash \neg\varphi &\iff T, \Theta \nvDash \varphi && (7)\\
T, \Theta \vDash \varphi \wedge \psi &\iff T, \Theta \vDash \varphi \wedge T, \Theta \vDash \psi && (8)\\
T, \Theta \vDash \exists x : s.\varphi &\iff \exists v \in D_s \mid T, \Theta[x \to v] \vDash \varphi && (9)\\
& && (10)
\end{aligned}
$$

Un protocollo P soddisfa un lemma ϕ se le tracce generate a partire ϕ sono contenute nell'insieme delle tracce generate a partire P

$$P \vDash \phi \iff trace(\phi) \subseteq trace(P)$$

5 Modellazione del protocollo Zigbee in Tamarin

In questo paragrafo presento la modellazione in Tamarin del protocollo Zigbee da parte di Li (et al.)[9]. Per essere consistente con la nomenclatura delle regole del codice Tamarin, utilizzerò i diagrammi di stato presenti nel paper originale.

5.1 Generazione delle chiavi iniziali

La modellazione prevede innanzitutto la specifica delle regole per la generazione delle chiavi fondamentali, ossia la *network key* da parte del coordinator, la *pre-configured key* condivisa tra coordinator e device, ed infine la link key condivisa tra due device

Essendo le regole ripetitive nella struttura, procederò a descrivere solo la regola per la generazione della *pre-configured key*, mentre per le successive regole mi limiterò ad evidenziare solamente i passaggi più interessanti.

La regola per la generazione della *pre-configured key* è la seguente:

```
rule D_pck_generation:
  [ Fr(~pck) ]
  --[ SecretPCK(~pck) ]->
  [ !PCK($D,$C,~pck) ]
```

La precondizione `[ Fr(~pck) ]` indica che `~pck` è un valore fresco (univoco e generato al momento), dove Fr è un fatto speciale per la generazione di termini freschi e $\sim$ denota un termine segreto. L'etichetta `--[ SecretPCK(~pck) ]-->` marca `~pck` come segreta, utile per verificare proprietà di sicurezza. La post-condizione `[ !PCK($D, $C, ~pck) ]` memorizza la chiave come fatto persistente (`!`), associandola ai dispositivi `$D` e `$C`, dove `$` indica variabili di messaggio. In sintesi, la regola genera una chiave PCK segreta, la memorizza in modo persistente e la associa a due dispositivi specifici, garantendo la sicurezza delle loro comunicazioni.

Il modello proposto nel paper originale definisce anche le regole `Reveal_nk`, `Reveal_pck`,`Reveal_ntlk`, necessarie per la definizione dei lemmas in cui le chiavi segrete diventano pubbliche. Nel caso della chiave di rete la regola è la seguente:

```
rule Reveal_nk:
  [ !NwkKey(C,~nk) ] --[ RevNK(C)]-> [ Out(~nk) ]
```

L'applicazione della regola fa sì che se C è in possesso di una network key sicura (`[!NwkKey(C,~nk)]`), questa viene trasmessa sul canale (`[ Out(~nk) ]`).

5.2 Join di un nuovo nodo alla rete

Per modellare la comunicazione per il join di un nuovo nodo alla rete gli autori hanno seguito lo schema in figura 6 e descritto nel paragrafo 3.3.2.

Per simulare la trasmissione out-of-band dei messaggi (in particolare della pre-configured link key), gli autori hanno definito le due regole `ChanOut_S` e `ChanOut_S`.

```
rule ChanOut_S:
        [ Out_S($A,$B,x) ]
      --[ ChanOut_S($A,$B,x) ]->
        [ !Sec($A,$B,x) ]

rule ChanIn_S:
        [ !Sec($A,$B,x) ]
      --[ ChanIn_S($A,$B,x) ]->
        [ In_S($A,$B,x) ]
```

Le regole `ChanOut_S` e `ChanIn_S` definiscono rispettivamente

- la ricezione di un nuovo messaggio `x` dalla rete. Se in uno stato si riceve un messaggio `x` scambiato da due entità `A` e `B` (`[ Out_S($A,$B,x) ]`) allora `x` è sicuramente stato scambiato da A e B (`[ !Sec($A,$B,x) ]`)
- l'invio di un messaggio `x` nella rete sicura. Se è sempre vero che il messaggio `x` è sicuro (`[ !Sec($A,$B,x) ]`) allora è possibile inoltrare il messaggio sul canale sicuro (`[ In_S($A,$B,x) ]`)

Dal momento che il protocollo viene modellato sia per ZigBee version 1.0 che per la versione 3.0, gli autori hanno provveduto alla definizione dei due fatti `!ZigBeeV3()` e `!ZigBeeV1()`[9].

Di seguito la regola che modella la prima Beacon request prevista nel joining alla rete da parte di un device.

```
rule D1_Beacon_req:
  [ !PCK($D,$C,~pck) ]
  --[ Beacon_req() ]->
  [ Out(<$D,'0x07'>), BeaconReq($D), !ZigbeeV3(), !ZigbeeV1() ]
```

Come si può vedere nei fatti conclusivi sono presenti entrambi i fatti `!ZigbeeV3(), !ZigbeeV1()` perciò in seguito alla beacon request è possibile procedere con ulteriori regole appartenenti a entrambe le versioni.

Se una certa regola è applicabile solo in una delle due versioni, allora nel multi-set fact delle premesse sarà presente solo uno dei due fatti, come ad esempio nel caso

```
rule D2_3_association_req:
  [ In(<C,D,'panID'>),  BeaconReq(D), !PCK(D,C,~pck), !ZigbeeV3() ]
  --[ Send_Association_Req(D,C) ]->
  [  Association_req(D,C), Out_S(D,C,~pck) ]

rule D2_1_association_req:
  [ In(<C,D,'panID'>),  BeaconReq(D), !PCK(D,C,~pck), !ZigbeeV1() ]
  --[ Send_Association_Req(D,C) ]->
  [  Association_req(D,C), Out(<D,'pck'>) ]
```

Per quanto simili, in `D2_3_association_req` le conclusioni prevedono l'inoltro sul canale sicuro della pre-configured key `Out_S(D,C,~pck)` , mentre la versione 1.0 prevede nelle conclusioni l'inoltro sul canale della coppia `<D,'pck'>` che quindi è trasmesso in modo insicuro.

5.3 Aggiornamento della Trust Center Link Key

Zigbee 3.0 ha introdotto la necessità di aggiornare la Trust Center Link Key in seguito al joining nella rete. Questo passaggio è modellato come in figura 7.

Bisogna notare che all'interno della regola `D4_NTLK_verify`

```
rule D4_NTLK_verify:
  [ In(senc(ntlk, ~pck)), NTLK_req(D,C), !PCK(D,C,~pck) ]
  --[NTLK_Verifiy_Req(D,C,ntlk)]->
  [ Out(senc(sdec(ntlk,~pck),(sdec(ntlk,~pck)))) ]

rule C4_NTLK_verified:
  [ In(senc(sdec(ntlk,pck),(sdec(ntlk,pck)))), NTLKSent(C,D,ntlk) ]
  --[NTLK_verified(ntlk)]->
  [ Counter(D,'0'), !SendMsg(D,C) ]
```

la notazione `senc(ntlk, ~pck)` indica che il valore `ntlk` è già stato cifrato con la chiave `~pck`. Non è un'operazione che avviene nel momento in cui viene scritta, ma rappresenta un dato che è stato cifrato in precedenza. Quindi, possiamo interpretarlo come "ntlk è una chiave cifrata con pck." Questa interpretazione è fondamentale perché spiega perché, quando il dispositivo riceve `senc(ntlk, ~pck)`, per ottenere ntlk deve applicare `sdec(senc(ntlk, pck), pck)`, ovvero la decifratura con la chiave `pck`.

5.4 Aggiornamento della chiave di rete

L'aggiornamento della chiave di rete è un processo automatico che il protocollo esegue quando il frame counter associato alla network key raggiunge il suo valore massimo. Questo meccanismo è modellato dalla regola:

```
rule Frame_counter_increase:
[ !SendMsg(D,C), Counter(D, val) ]
--[Msg_Send()]->
[ Counter(D, inc(val)), Out(inc(val)) ]
```

Questa regola rappresenta il fatto che per ogni messaggio scambiato tra il dispositivo D e il coordinator C, il contatore viene incrementato (`inc(val)`). Ma la regola prevede che il nuovo valore del contatore venga rivelato all'esterno tramite `Out(inc(val))`, cioè reso potenzialmente accessibile all'attaccante. Questo è vero perchè nel dettaglio del protocollo il contatore non è protetto e quindi un attaccante può usarlo per ottenere informazioni utili ad un attacco e Tamarin può trovare casi in cui questa informazione porta ad una violazione.

All'interno delle altre regole non viene specificato il limite massimo per il counter, questo perchè non è un'informazione rilevante. Difatti il protocollo non subirebbe variazioni.

6 Formalizzazione delle proprietà di sicurezza in Tamarin

Nel modello di sicurezza formale sviluppato con Tamarin, le proprietà di sicurezza del protocollo vengono espresse sotto forma di *lemma*, che corrispondono a enunciati logici che devono essere verificati all'interno del sistema. Ogni lemma utilizza una combinazione di quantificatori esistenziali e universali, eventi e predicati per formalizzare aspetti cruciali della sicurezza del protocollo.

Un primo gruppo di lemma è legato alla correttezza dell'esecuzione del protocollo. Ad esempio, il lemma `execute` esprime che se un evento `Beacon_req()` si verifica in un certo istante, allora esisterà un momento successivo in cui un altro evento `NTLK_verified(y)` avrà luogo. Formalmente, questo si scrive come:

$$\exists\text{-trace} \quad (\exists \#i.\text{Beacon\_req}()@\#i) \Rightarrow (\exists y \#j.\text{NTLK\_verified}(\text{y})@\#j)$$

Questo significa che, in almeno una possibile traccia di esecuzione del protocollo, la richiesta di connessione porta necessariamente a una verifica della chiave di sessione `NTLK`. Un'idea simile si applica al lemma `execute_network_update`, che assicura che una richiesta di connessione implicherà un aggiornamento della rete (`Update_network()`).

6.1 Formalizzazione delle proprietà di segretezza

Oltre ai lemmi di esecuzione, il modello include proprietà di **segretezza**, che assicurano che determinate chiavi non diventino pubbliche a meno che non si verifichino determinate condizioni. Ad esempio, il lemma `secrecy_NK` ha la forma:

$$\forall x \#i. \quad \text{SecretNK}(x)@\#i \Rightarrow \neg(\exists \#j.\text{K}(x)@\#j) \quad | \quad \ldots$$

Questa formula stabilisce che, se un valore `x` è dichiarato segreto (`SecretNK(x)`) in un certo istante, allora non esisterà alcun momento `#j` in cui la chiave `x` sarà pubblicamente conosciuta (`K(x) @ #j`), a meno che non si verifichi una delle condizioni indicate a destra della formula. Le condizioni di eccezione sono espressioni di tipo `Ex C #r. RevNK(C) @ r`, che indicano eventi in cui la segretezza della chiave potrebbe essere revocata (`RevNK`).

Questo schema si ripete per altre chiavi critiche del protocollo, come `NTLK`, `PCK` e `LK`, con lemmi come `secrecy_NTLK`, `secrecy_PCK` e `secrecy_LK`. Tutti seguono una struttura simile: la segretezza è preservata a meno che non intervengano specifici eventi di revoca (`RevNK`, `RevPCK`, `RevNTLK`).

Questa formalizzazione consente di dimostrare che le proprietà di sicurezza del protocollo sono rispettate o, in caso contrario, di identificare scenari in cui potrebbero essere violate.

6.2 Formalizzazione della proprietà di autenticazione

Nell'analisi della sicurezza dei protocolli crittografici, l'*autenticazione* è un aspetto fondamentale che assicura che le entità coinvolte in una comunicazione abbiano effettivamente preso parte allo scambio dei messaggi in modo corretto. Il concetto di *autenticazione* è stato formalizzato in diversi modi, e il modello di Lowe fornisce una classificazione strutturata delle garanzie offerte da un protocollo.

6.2.1 Accordo sulla chiave

Un protocollo sicuro deve garantire che quando due entità stabiliscono una chiave di sessione, entrambe abbiano la certezza di usare la stessa chiave. Questo principio è espresso dal seguente lemma:

```
lemma key_agreement:
 "All nk1 nk2 coordinator device #i #j.
 Send_Network_Key(nk1,coordinator,device) @ i
 & Send_Network_Key(nk2,coordinator,device) @ j
   ==>
    nk1 = nk2"
```

Formalmente, questo significa che se un *coordinatore* invia due chiavi `nk1` e `nk2` a uno stesso dispositivo `device` in due momenti diversi `#i` e `#j`, allora le chiavi devono essere uguali. Questo vincolo impedisce a un attaccante di indurre il dispositivo a credere di aver negoziato una chiave diversa da quella effettivamente stabilita con il coordinatore.

6.2.2 Unicità della chiave

Oltre a garantire che non ci siano discrepanze nella chiave scambiata tra due entità, è anche necessario assicurarsi che una stessa chiave non venga inviata due volte in istanti temporali distinti. Questo concetto è espresso dal lemma:

```
lemma uniqueness_of_key:
 "All key coordinator device #i #j.
         Send_Network_Key(key,coordinator,device) @ i
         & Send_Network_Key(key,coordinator,device) @ j
           ==>
            #i = #j"
```

Questa proprietà garantisce che una chiave venga inviata una sola volta per ogni sessione. Se il coordinatore invia una chiave di rete `key` al dispositivo in due istanti `#i` e `#j`, allora questi due istanti devono coincidere. In altre parole, una stessa chiave non può essere trasmessa più volte in momenti diversi, riducendo così il rischio di attacchi di replay.

6.2.3 Weak Agreement

Il weak agreement rappresenta il livello più debole di autenticazione. Esso garantisce che, se un dispositivo `d` ha ricevuto una chiave `x1` dal coordinatore `c`, allora almeno una delle seguenti condizioni deve essere vera:

1. Il coordinatore ha effettivamente inviato una chiave di rete (`Send_Network_Key(c,d,x2)`)
2. La chiave è stata compromessa e resa pubblica (`RevNK(c)`)
3. La chiave preconfigurata `PCK` tra il coordinatore e il dispositivo è stata rivelata (`RevPCK(d,c)`)

```
lemma weak_agreement:
  " All c d x1 #i.
 NwkKeyRecv(d,c,x1) @ #i
 ==>
  ((Ex x2 #j. Send_Network_Key(c,d,x2) @ #j)
 |(Ex #r. RevNK(c) @ r)
        | (Ex #r. RevPCK(d,c)  @ r))"
```

Questa proprietà assicura che un dispositivo non possa dichiarare di aver ricevuto una chiave senza che ci sia una giustificazione per tale affermazione, ovvero senza un'effettiva trasmissione o una compromissione della chiave.

6.2.4 Non-Injective Agreement

Un livello più forte di autenticazione è il *non-injective agreement*. A differenza del *weak agreement*, questo lemma impone che la chiave ricevuta `x` debba coincidere esattamente con quella inviata dal coordinatore `c`. Tuttavia, non vi è alcun vincolo sulla possibilità che la stessa chiave venga usata in più sessioni.

```
lemma non_injective_agreement:
  " All c d x #i.
 NwkKeyRecv(d,c,x) @ #i
 ==>
  ((Ex #j. Send_Network_Key(c,d,x) @ #j)
 |(Ex #r. RevNK(c) @ r)
        | (Ex #r. RevPCK(d,c)  @ r))"
```

Qui la differenza rispetto a `weak_agreement` è che non vengono considerate chiavi `x2` diverse: la chiave `x` ricevuta deve essere esattamente quella inviata. Tuttavia, il lemma non impone che ogni esecuzione sia distinta, quindi lo stesso `x` potrebbe essere riutilizzato in più sessioni senza invalidare la proprietà.

6.2.5 Injective Agreement

L'injective agreement è il livello più forte di autenticazione e impone non solo che la chiave ricevuta sia esattamente quella inviata, ma anche che ci sia un ordine temporale ben definito.

```
lemma injective_agreement:
  " All c d x #i.
 NwkKeyRecv(d,c,x) @ #i
 ==>
  (Ex #j. Send_Network_Key(c,d,x) @ #j
 &  j < i)
 |(Ex #r. RevNK(c) @ r)
        | (Ex #r. RevPCK(d,c)  @ r)"
```

Questo lemma stabilisce che se `d` riceve la chiave `x` dal coordinatore `c` in un istante `#i`, allora deve esserci stato un momento precedente `#j < #i` in cui `c` ha inviato quella chiave.

Diversamente dal non-injective agreement, qui si richiede che ogni run sia differente e linearmente ordinata, in modo che il protocollo non ammetta attacchi di replay o di riutilizzo della chiave in altre run.

7 Analisi dei risultati: vulnerabilità trovate

La verifica effettuata con lo strumento Tamarin dimostra che Zigbee 3.0 soddisfa tutte le proprietà di sicurezza desiderate. Diversamente, la versione di Zigbee 1.0 presenta vulnerabilità a livello della segretezza delle chiavi. Andando sulla web-interface di Tamarin e caricando il modello che formalizza il protocollo e le proprietà di sicurezza espresse nei paragrafi 5 6 [13], sul lato sinistro vediamo evidenziato in rosso il lemma `secrecy_NK` (figura 9). Cliccando su `SOLVED` possiamo visualizzare i counterexamples del lemma. In figura 10 è visibile il constraint system che giustifica la violazione della segretezza della chiave. L'uso di una chiave precondivisa la cui segretezza non è richiesta dalla specifica del protocollo, permette di decifrare l'ultimo messaggio scambiato dalla regola `C2_1_Send_Nwk_Key` e quindi di ottenere la chiave di rete.

8 Conclusioni

L'analisi sul protocollo Zigbee ha permesso di ottenere due interessanti risultati. Da un lato ha permesso di individuare una importante vulnerabilità di sicurezza nella versione 1.0 e dimostrare che la versione 3.0 risolve tale problema, assicurando la segretezza e l'autenticità dei messaggi che vengono scambiati. Dall'altro lato, rappresenta un'ulteriore conferma dell'efficacia di Tamarin-prover come strumento di verifica formale per i protocolli crittografici, in quanto è in grado sia di scoprire falle di sicurezza sia di dimostrare che le correzioni apportate sono corrette.

References

[1] ZigBee Specifications 2012. URL: http://www.zigbee.org/wp-content/uploads/2014/%2011/docs-05-3474-20-0csg-zigbee-specification.pdf.

[2] Zigbee Base Device Behavior Specification Version 1.0 2016. URL: http:%20//www.zigbee.org/wp-content/uploads/2014/10/docs-13-0402-13-00zi-Base-%20Device-Behavior-Specification-2.pdf.

[3] Zigbee Security Basics 2017. URL: https://research.kudelskisecurity.com/2017/11/08/%20zigbee-security-basics-part-2/.

[4] Zigbee Alliance. URL: https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf.

[5] David Basin et al. "Symbolically analyzing security protocols using tamarin". In: *ACM SIGLOG News* 4.4 (Nov. 2017), pp. 19–30. DOI: 10.1145/3157831.3157835. URL: https://doi.org/10.1145/3157831.3157835.

[6] Hubert Comon-Lundh and Stéphanie Delaune. "The finite variant property: how to get rid of some algebraic properties". In: *Proceedings of the 16th International Conference on Term Rewriting and Applications*. RTA'05. Nara, Japan: Springer-Verlag, 2005, pp. 294–307. ISBN: 3540255966. DOI: 10.1007/978-3-540-32033-3_22. URL: https://doi.org/10.1007/978-3-540-32033-3_22.

[7] csa-iot.org. URL: https://csa-iot.org/all-solutions/zigbee/why-zigbee/.

[8] Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen. "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi". In: *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*. 2007, pp. 46–51. DOI: 10.1109/IECON.2007.4460126.

[9] Li Li, Proyash Podder, and Endadul Hoque. "A formal security analysis of ZigBee (1.0 and 3.0)". In: *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*. HotSoS '20. Lawrence, Kansas: Association for Computing Machinery, 2020. ISBN: 9781450375610. DOI: 10.1145/3384217.3385617. URL: https://doi.org/10.1145/3384217.3385617.

[10] G. Lowe. "A hierarchy of authentication specifications". In: *Proceedings 10th Computer Security Foundations Workshop*. 1997, pp. 31–43. DOI: 10.1109/CSFW.1997.596782.

[11] Alberto Martelli and Ugo Montanari. "An Efficient Unification Algorithm". In: *ACM Trans. Program. Lang. Syst.* 4.2 (Apr. 1982), pp. 258–282. ISSN: 0164-0925. DOI: 10.1145/357162.357169. URL: https://doi.org/10.1145/357162.357169.

[12] Yahoo Finance - Zigbee market spread. URL: https://finance.yahoo.com/news/zigbee-market-projected-hit-usd-145700998.html?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAACGyZdU9lFYpmkk8SURvRoPDE60vY3Oul-Rx2p_m2UXrsFgoIZxwH1DTM6NY_AS-dsaq0H4CwfMXLxz0vyZALrCS-Tba9EOeXNKgUkQBMIR-3fxysfVn19IEFzNDMZbMn2J_Hg0PFa91alS4SEJTA-nKzfT939oMdZthVX_ihQ_-.

[13] Li-Syr. *Li-Syr/Zigbee-Tamarin*. URL: https://github.com/Li-Syr/zigbee-tamarin.

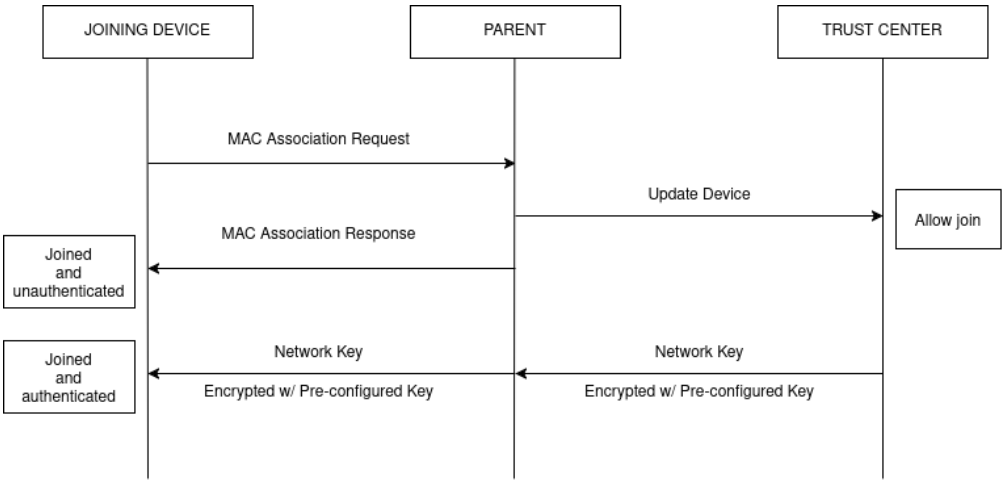

Figure 4: Joining a secure network

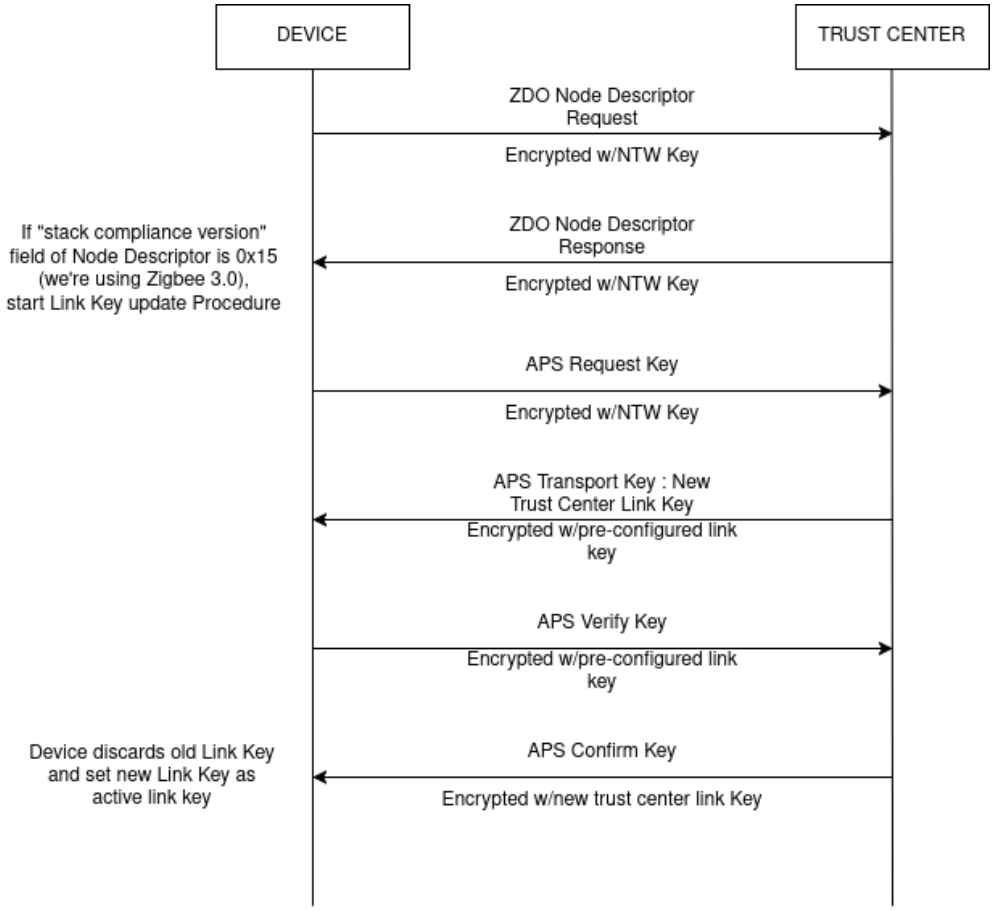

Figure 5: Updating Trust Center Link Key

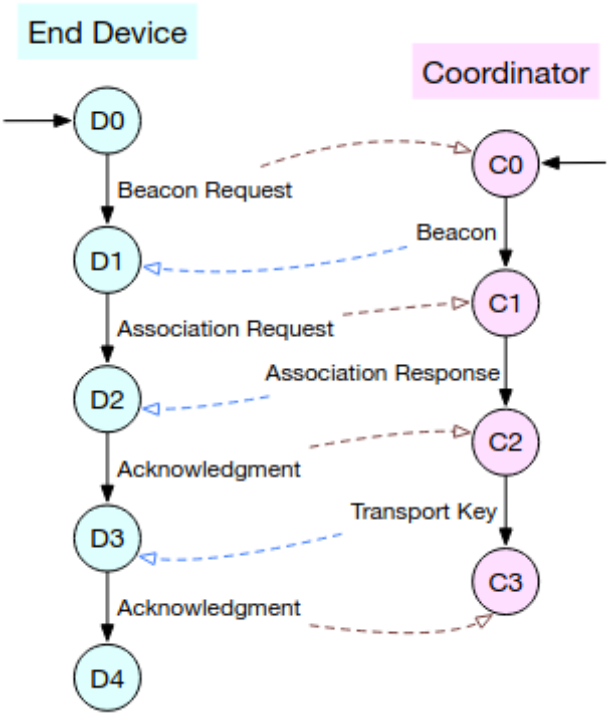

Figure 6: State diagram of network joining

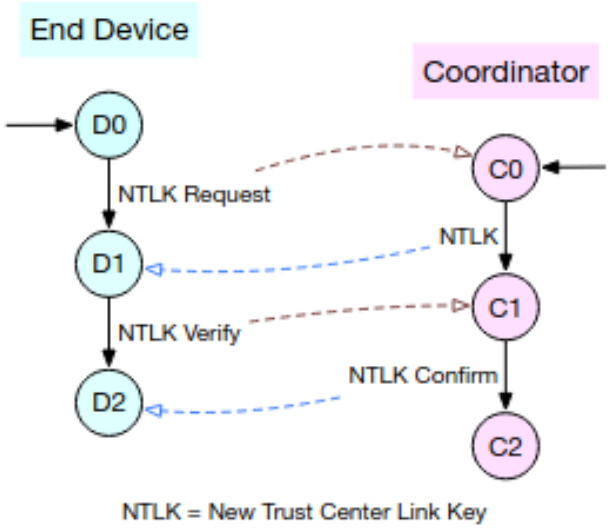

Figure 7: State diagram of trust center link key update

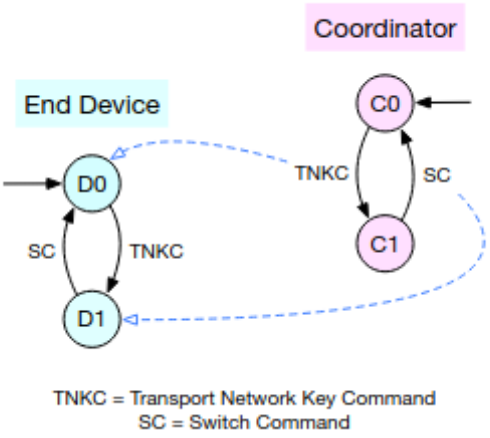

Figure 8: State diagram of network key update

```
lemma secrecy_NK:
  all-traces
  "∀ x #i.
         (SecretNK( x ) @ #i) ⇒
         (((((¬(∃ #j. K( x ) @ #j)) ∨ (∃ C #r. RevNK( C ) @ #r)) ∨
           (∃ C D #r. RevPCK( D, C ) @ #r)) ∨
          (∃ C D #r. RevNTLK( C, D ) @ #r))"
simplify
solve( !KU( ~nk ) @ #vk )
  case C2_1_Send_Nwk_key
  SOLVED // trace found
next
  case C2_3_Send_Nwk_key
  by sorry
next
  case Reveal_nk
  by contradiction /* from formulas */
qed
```

Figure 9: Tamarin found a trace that violates the lemma Secrecy_NK

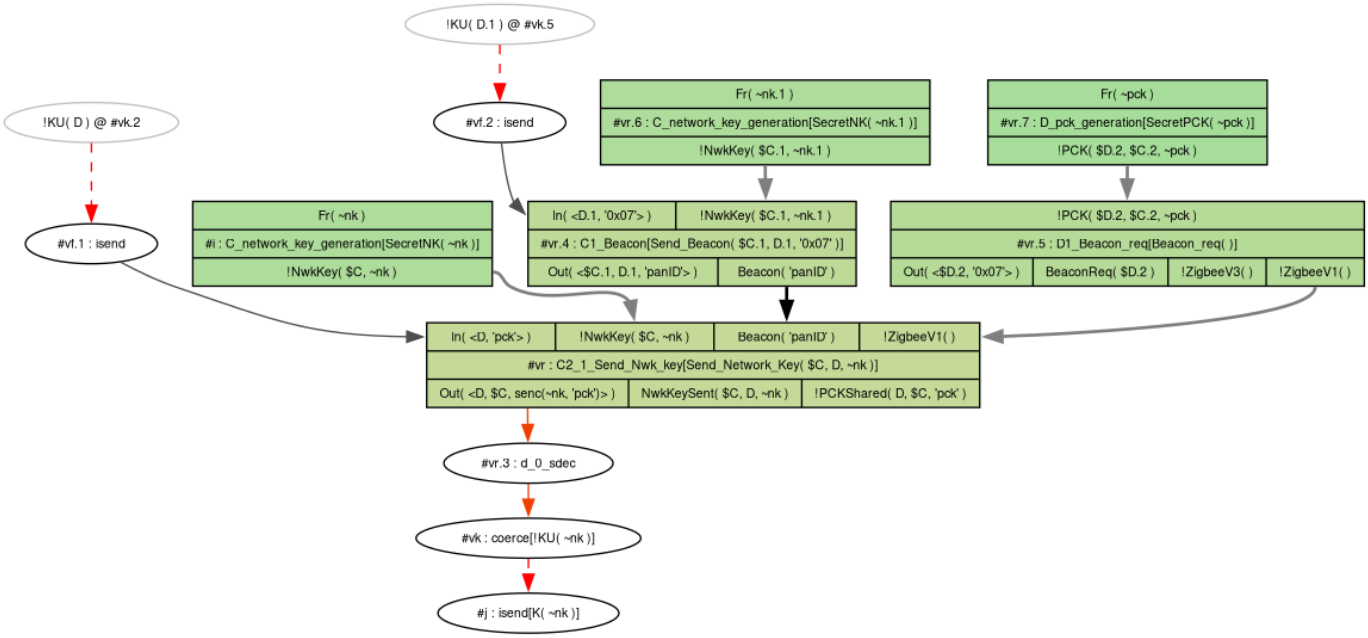

Figure 10: Network Key Disclosure