Politecnico di Torino

Politecnico di Torino

Collegio ETF

2022

Sessione di Laurea Dicembre

Online Key Management
OKM

Relatore: Claudio Passerone

Giacomo La Porta 280029

Abstract

Le compagnie ferroviarie spendono tempo e risorse per far in modo di rendere sicuri le comunicazioni tra i sistemi on-board e trackside, cosi da rendere le tratte il più sicuro possibile ed evitare incidenti. Oltre a rendere le linee più sicure le compagnie ferroviarie voglio incrementare la densità delle linee permettendo il passaggio di treni più frequentemente sulla stessa tratta. Tutto ciò è possibile se le comunicazioni avvengono in maniere sicura e univoca. L'obbiettivo del progetto è di creare un sistema di scambio chiavi di cifratura per creare una comunicazione sicura tra sistemi del treno e di terra. Il sistema deve essere fatto in moda da non permettere a dispositivi esterni di visualizzare , inviare o cambiare i dati che vengono scambiati dai sistemi. Per fare ciò, il sistema e formato da più sottosistemi che comunicano tra di loro e si scambiano chiavi per l'autenticazione.

Indice

CAPITOLO 1

Lista degli acronimi

- CA : Certificate Authority;
- CMP: Certificate Management Protocol;
- DB : DataBase;
- OCPS : On-line Certificate Status Protocol;
- PSK : Pre-Shared Key;
- RA : Registration Authority;
- TLS : Transport Layer Security;
- UTF-8: Unicode Transformation Format 8-bit;
- KM : Key Management;
- KMS : Key Management System;
- KMC : Key Management Center;
- KMAC :Key Management Autentication Code;
- ERTMS : European Reailway Traffic Management System;
- ETCS : European Traffic Control Signalling;
- EUROBALISE : ETCS/ERTMS transponder tra le rotaie;
- EURORADIO : ETCS/ERTMS apparecchiature di rete radio conformi con le funzioni richieste per una comunicazione sicura.

1.1 Termini e definizioni

- Certificate Authority (CA):L'entità responsabile dell'emissione di certificati digitali per associare le chiavi pubbliche alle identità degli utenti;
- Confidentiality: nell'ambito dei sistemi informatici la riservatezza, consente ai soli utenti autorizzati di accedere ai dati protetti utilizzando meccanismi specifici per garantire la riservatezza e salvaguardare i dati da intrusioni dannose;

- Cryptography:I principi, i mezzi e le modalità di trasformazione dei dati al fine di garantirne la riservatezza, l'autenticità, il non ripudio e l'integrità;
- ETCS entity:ETCS EVC, RBC or RIU
- ETCS ID : identificativo univoco dell'entità ETCS;
- Expanded ETCS ID: identificativo univoco dell'entità ETCS espanso con l'aggiunta di un codice che ne identifica il tipo;
- Key database: Contiene le voci chiave nelle entità KMS;
- Key entry : Una chiave di autenticazione (KMAC) con le seguenti informazioni correlate:
  - Identificatore del KMC che ha emesso la chiave;
  - Numero seriale della chiave;
  - Periodo di validità della chiave;
  - elenco di entità KMAC a cui è allocata questa chiave;
- Key serial number : Il numero che identifica in modo univoco una chiave all'interno del set di chiavi generato da un KMC;
- KMAC entity : entità on-board KMAC o entità trackside KMAC;
- KMAC on-board entity : Apparecchiature on-board ;
- KMAC trackside entity : RBC or RIU;
- KMS entity : entità KMAC o KMC;
- Pseudorandom number generator :Un generatore di numeri pseudocasuali è un algoritmo per generare una sequenza di numeri le cui proprietà si avvicinano alle proprietà di sequenze di numeri casuali;
- Registration Authority (RA) : L'entità responsabile in una PKI per l'accettazione delle richieste di certificati digitali e l'autenticazione dell'entità che effettua la richiesta.
- PEER : Per PEER si intente "tra pari", dato che è un sistema PEER to PEER. In inglese, peer to peer, spesso abbreviato come p2p, significa "tra pari" e descrive un tipo di rete di comunicazione in cui ciascun nodo comunica direttamente con gli altri, senza la mediazione di un server.

CAPITOLO 2

Introduzione

La necessità di avere tratte sicure e ben segnalate ha portato allo sviluppo di applicazione che consentissero queste comunicazioni in tempo reale. Per gestire il traffico ferroviario si utilizzano applicazioni che trasferiscono messaggi, tramite un sistema di trasmissione aperto, tra i componenti European Reailway Traffic Management System (ERTMS)/European Traffic Control Signalling (ETCS). L'ERTMS è un sistema unico europeo di segnalamento e controllo della velocità che garantisce l'interoperabilità dei sistemi ferroviari nazionali, mentre l'ETCS è il componente di segnalamento e controllo del sistema ERTMS.
Dato che i sistemi di trasmissione aperti sono intrinsecamente vulnerabili e di conseguenza è impossibile escludere accessi non autorizzati, si è cercato di sviluppare dei collegamenti per la trasmissione di dati sicuri. Al fine di garantire l'integrità e l'autenticazione dei messaggi inviati su un mezzo di trasmissione non attendibile, si è avuta la necessità di sviluppare tecniche crittografiche con chiavi segrete. Queste chiavi venivano trasmesse manualmente con degli addetti che salivano a bordo treno per installare la relativa chiave o il un loro sottoinsieme. Questo era molto scomodo e poco efficiente, questo portò al voler creare un interfaccia online in modo da poter comunicare più velocemente. L'armonizzazione delle interfacce avviene in modo policy-open, consentendo a ciascun operatore di attuare una politica di gestione delle chiavi adeguata alle proprie specifiche ed esigenze di sicurezza, per esempio utilizzando chiavi di autenticazione diverse per ciascuna coppia delle entità Key Management Autentication Code (KMAC) o utilizzando la stessa chiave di autenticazione per un gruppo di entità trackside KMAC.
Pertanto è necessario fornire un'interfaccia per l'installazione, l'aggiornamento e l'eliminazione di tali chiavi.

CAPITOLO 3

Ambiente del progetto

3.1 Revisione del sistema di segnalamento ferroviario

Per gestire il traffico ferroviario e mantenere la sicurezza del treno è stato introdotto il segnalamento ferroviario che è costituito da un insieme di sistemi, tutti questi sistemi lavorano insieme con un protocollo specifico. Poiché diversi treni condividono lo stesso binario, lo scopo principale è la gestione del traffico per prevenire possibili collisioni.
Questa esigenza è presente fin dalla nascita delle ferrovie. Tutto il sistema di sicurezza sviluppato nel tempo condivideva uno stesso concetto di base: i treni non possono scontrarsi se non possono occupare contemporaneamente lo stesso tratto di binario, un binario che costituisce un percorso è suddiviso in una serie di blocchi di lunghezza variabile da 1 a 10 km. Venivano usate diverse modalità per segnalare ai treni il permesso di ingresso nel blocco successivo e per controllare la presenza di un treno nello stesso.
All'inizio dell'era ferroviaria, 1800-1850, il personale era impiegato per sostare su ogni blocco e segnalare ai macchinisti, a mano o con la bandiera, la possibilità di entrare o meno in un blocco; questi uomini erano chiamati *sentinelle*. Le sentinelle non avevano prove per garantire che il treno abbia sgomberato la tratta, a causa dell'assenza di comunicazioni tra il guardiano del blocco(sentinella) e il treno. Infatti, se un treno precedente si è fermato per qualsiasi motivo, l'equipaggio del treno successivo non aveva idea delle presenza o meno di altri treni nel stesso blocco, a meno che non fosse chiaramente visibile; per questo motivo gli incidenti erano molto comuni. Per evitare questo problema sono stati sviluppati diversi sistemi per garantire la sicurezza.
I primi miglioramenti si sono avuti con l'invenzione della linea telegrafica, grazie alla quale le sentinelle potevano comunicare. Un'altra importante invenzione fu il semaforo nel 1830 che aiuta la segnalazione al personale del treno. Durante i cinquant'anni tra il 1850 e il 1900 furono sviluppati più sistemi per rilevare la presenza di un treno senza bisogno di un essere umano, come il circuito del treno e il contro asse, questi erano necessari per concentrare il controllo sulla segnalazione pseudo-automatica.
Lo sforzo dell'ente ferroviario dei principali paesi si è concentrato sulla creazione di un sistema di controllo automatico e centralizzato con l'affinamento dei sistemi precedenti e la creazione di protocolli di sicurezza sui binari e sui treni. Grazie alla crescita delle reti e all'aumento del traffico, le agenzie nazionali vollero massimizzare la capacità delle rotte aumentando il numero di entità ferroviarie in grado di operare sulla stessa rotta.
Questo aumento ha sollevato la necessità di conoscere continuamente molte informazioni sullo stato dei treni. I dati che erano raccolti dall'intera rete ferroviaria, erano gestiti nel centro di instradamento e ritrasmessi a ciascun treno. Ciò ha portato alla costruzione di reti complete di sensori e attuatori in pista e a bordo, collegati tramite diversi livelli fisici a centri di comando che hanno la capacità di

pianificare, dirigere, coordinare e controllare le entità sul campo. L'attrezzatura on-board è cambiata per poter comunicare le informazioni ricevute all'equipaggio permise di avere diverse segnalazioni in cabina per gestire direttamente la velocità di ogni treno su una parte nota del percorso.
Nel corso del tempo, per diversi motivi, ciascuna agenzia nazionale ha sviluppato un insieme di dispositivi, sistemi e protocolli per la gestione delle reti ferroviarie nazionali. Tale granularità nelle normative comporta difficoltà nell'interoperabilità dei diversi sistemi, in particolare durante il passaggio dei confini nazionali.

3.2 Standard ERTMS/ETCS

Nel dicembre 1989 il ministro dei Trasporti Europeo prende la decisione di avviare un progetto per analizzare le problematiche relative al segnalamento e al controllo dei treni. Alla fine del 1990, l'Istituto Europeo di Ricerca Ferroviaria (ERRI) inizia a pensare di sviluppare un comune Automatic train protection (ATP)/Automatic Train Control (ATC), un sistema che utilizza un'indicazione della velocità target e avvisi acustici per avvisare il conducente se il treno ha superato un segnale rosso o ha superato un limite di velocità azionando un freno, (se il freno è automatico è chiamato ATC), L'ATP o l'ATC potevano essere adottati in tutti i paesi europei. Questi tipi di protocolli e protezioni permettevano lo sviluppo di trasporti ad alta velocità (HS) e ad alta capacità (HC), i quali permettevano un tipo di trasporto significativamente più veloce del traffico ferroviario tradizionale o con molti più treni sullo stesso blocco.

- Una ferrovia ad alta velocità (HS) è una linea moderna con percorsi aventi un rettilineo pianeggiante, ove possibile, su cui scorrono appositamente dei treni progettati per raggiungere alte velocità massime.
- Il termini "Alta Capacità" (HC) indica un sistema ferroviario su cui si può trasportare sia il treno merci che i passeggeri su una linea HS, ma a volte può riferirsi a sistemi avanzati per il controllo del traffico ferroviario, cioè consentire il passaggio di un numero maggiore di treni perché più controllato e regolare.

3.2.1 Livelli del ETCS

L'ETCS è suddiviso in diversi livelli funzionali, i livelli dipendono da come è attrezzata la ferrovia e dal modo in cui le informazioni vengono trasmesse al treno.

3.2.2 Livello 0

Per questo livello gli standard ERTMS/ETCS prevedono l'assenza dei segnali laterali, l'attrezzatura a bordo treno controlla la velocità massima di quel tipo di treno e il macchinista deve osservare i segnali a terra.

3.2.3 Livello 1

È un sistema di segnalamento in cabina che può essere sovrapposto al sistema di segnalamento esistente, i radiofari (Eurobalise) raccolgono il segnale dai sensori a terra tramite adattatori di segnale e li trasmettono al veicolo come autorizzazione ad attraversare uno o più blocchi insieme ai dati di percorso. L'attrezzatura on-board monitora e calcola continuamente la velocità massima e la curva di frenata da questi dati.

3.2.4 Livello 2

È un sistema di protezione digitale del treno basato sul segnale radio. Le autorizzazioni di movimento sono date al guidatore per consentire al treno di muoversi sul binario. E' possibile lavorare senza una segnalazione laterale a parte per alcuni pannelli indicatori. Tutti i treni segnalano la loro esatta posizione e direzione all'RBC (Radio Block Center) a intervalli regolari attraverso il GSM-R. L'autorizzazione di movimento viene trasmessa al treno continuamente con i dati di velocità e percorso.
A questo livello gli EUROBALISE vengono utilizzati come fari di posizionamento passivi. I fari di posizionamento vengono utilizzati in questo caso come punti di riferimento per la correzione degli errori di misurazione della distanza. A questo livello le infrastrutture posizionate a terra sono risultate molto limitate e la logica di sicurezza è prevalentemente interna al treno.

3.2.5 Livello 3

Questo livello è in fase di studio a causa delle implicazioni sulla sicurezza. L'idea è utilizzare solo la trasmissione continua di dati con EuroRadio senza la necessità di molte apparecchiature a terra. Ciò consente la possibilità di utilizzare blocchi logici a lunghezza variabile anziché blocchi fissi per un controllo basato sull'utilizzo attuale dell'infrastruttura.

3.3 Equipaggiamento track-side

La parte track-side è il sottosistema che comunica con gli elementi esterni ed è formata da 3 sistemi.

- **ETCS**, il sistema di controllo della sicurezza e della distanza del treno sulle rotte;
- **RBS** il quale gestisce i dati di scambio in modo sicuro;
- **KMS** utilizzato per le chiavi crittografiche per consentire comunicazioni di dati sicure su Euroradio.

L'ETCS e formato da i seguenti componenti componenti:

- **EUROCAB** è un'architettura aperta per le apparecchiature on-board che elaborano le informazioni ricevute da diverse sorgenti, dopo aver ricevuto queste informazioni l'**EUROCAB** elabora e aggiorna la curva di rottura che è la previsione di quanto deve rallentare il treno per evitare la collisione con un altro treno;
- **EUROBALISE** si tratta di un sistema discontinuo di trasmissione dati track-side composto da balise, un dispositivo elettronico o transponder installato tra i binari di una tratta ferroviaria in cui sussiste un dispositivo di controllo della sicurezza della circolazione dei treni, fissate tra i binari che inviano informazioni al treno in transito;
- **EURORADIO** è un sistema di trasmissione dati per consentire la comunicazione di sottosistemi ferroviari e trackside per comunicare utilizzando protocolli Internet.

La figura 3.1 mostra uno schema di come i vari sistemi comunicano.

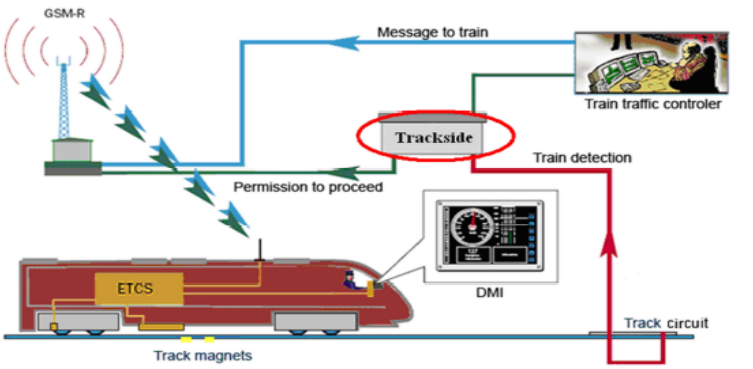

Figura 3.1: Track-Side equipment

3.4 Equipaggiamento on-bord

Una linea ferroviaria HS/HC può essere dotata di un complesso sistema di trasporto attrezzato in grado di controllare i movimenti dei treni al fine di garantire la sicurezza e la regolarità del traffico. L'architettura è composta da:

- **HSSS** Sistema di segnalazione ad alta velocità.
- **EVC** è il computer on-board, che elabora in sicurezza le funzioni on-board sulla base di:
  1. informazioni ricevute dalle apparecchiature on-board;
  2. dati introdotti dal conducente;
  3. dati provenienti dai sensori on-board;
- **DMI** è il principale mezzo di interazione tra il driver e il sistema. È usato per:
  1. visualizzare segnali e indicazioni tramite un monitor in ogni cabina di guida;
  2. acquisire i dati inseriti e abilitare funzioni specifiche tramite una serie di tasti e pulsanti;
  3. raggiungere l'interoperabilità tecnica sul lato conducente.
- **Odometria** è una tecnica per stimare la posizione e la velocità di un veicolo su ruote, basata sulle informazioni dei sensori per la misurazione della distanza percorsa. L'European Vital Computer (EVC) è in grado di calcolare ciclicamente ciascuno di questi valori, tra due punti definiti della pista P1, P2, identificati da due balise differenti. Quando il treno supera il punto P2, la sua posizione viene aggiornata e il processo di stima dell'odometria riprende.

La figura 3.2 mostra uno schema a blocchi dei due sistemi sia track-side che on-board.

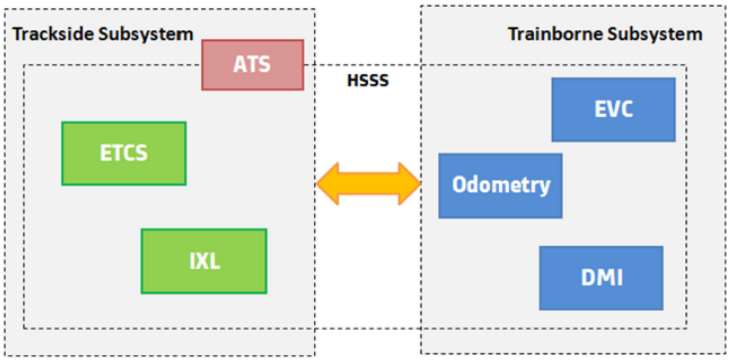

Figura 3.2: System Architecture

3.5 Scopo e finalità

Lo scopo del progetto è quello di creare un interfaccia online che permetta lo scambio di chiavi senza la necessità che una persona fisica salga sul treno per la gestione o installazioni di chiavi. Per fare ciò si è utilizzato il Key Management (KM) che è il componente che gestisce le chiavi e al quale si è integrata la parte di gestione online delle chiavi e i protocolli di verifica delle chiavi. La scheda usata è stata prodotta dal cliente con sistema linux ,minimizzato solo per permettere al programma di svolgere le sue funzioni. Il programma è stato sviluppato in linguaggio C usando la libreria openssl la quale è una libreria con le funzioni per la gestione delle connessioni Transport Layer Security (TLS).

CAPITOLO 4

Principi e concetti del KM

Al fine di garantire la comunicazione su un sistema di trasmissione aperto, le apparecchiature on-board e trackside nel sistema ERTMS/ETCS si scambieranno informazioni utilizzando il protocollo EuroRadio [Subset-037]. Quando un'apparecchiatura ETCS stabilisce una connessione con un'altra apparecchiatura ETCS, entrambi devono essere in grado di autenticare l'altra e verificare che si tratti di un entità autorizzato. In questo modo si ottiene anche l'autenticità e l'integrità delle informazioni scambiate tra di loro.
Il metodo per autenticare entrambe le entità comunicanti si basa su un dialogo di identificazione e autenticazione (IA); questo dialogo deve aver luogo ogni volta che due entità avviano una nuova connessione sicura. Dopo un dialogo , avvenuto con successo, i dati vengono protetti utilizzando un codice di autenticazione del messaggio (MAC).
Il calcolo di questo codice si basa sull'esistenza di una chiave di autenticazione segreta condivisa (KMAC) nota alle entità che comunicano tra loro. Il dialogo e le procedure di calcolo MAC sono completamente specificate nel modulo funzionale.
Tuttavia, le procedure non forniscono alcun mezzo per creare, distribuire o aggiornare queste chiavi. Inoltre, la loro efficacia dipende dal fatto che la chiave è segreta, la quale può restare tale solo utilizzando funzioni di gestione delle chiavi.

4.1 Architettura del KM

La figura 4.1 mostra l'architettura di riferimento del KM.

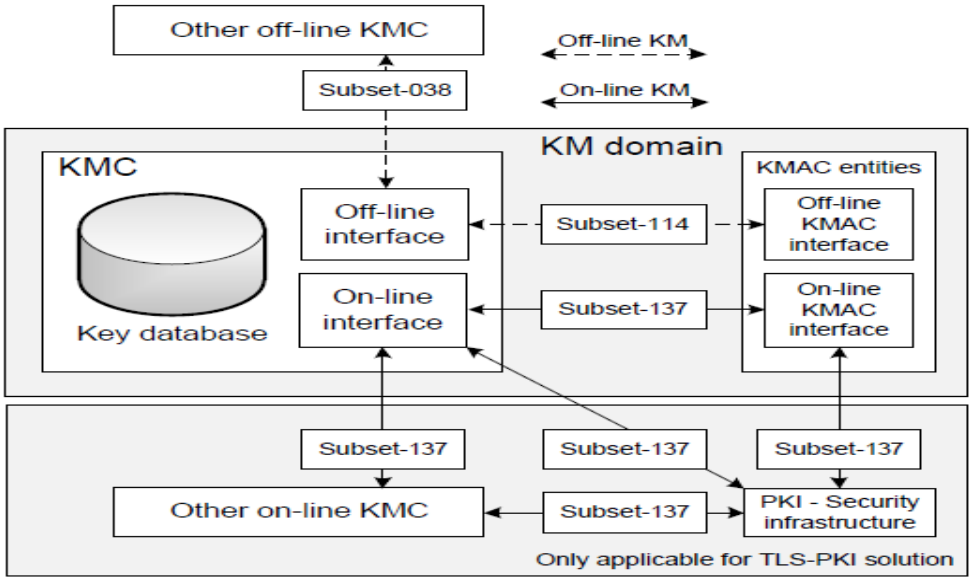

Figura 4.1: Architettura di riferimento del KM

Il KM è un insieme di entità con un unico KMC, tutti i KMAC (offline e online) fanno riferimento ad un solo KMC per la gestione delle sue chiavi. Un KMC può amministrare entità trackside e on-board; Tutti i KMAC appartenenti allo stesso dominio KM hanno un unico KMC di origine, inoltre il KMC d'origine gestisce le voci chiave per i KMAC specifici.
L'interfaccia online tra i Key Management System (KMS) consente a un KMC di gestire le chiavi di autenticazione con i KMAC nel suo dominio e con altri KMC. L'interfaccia consente a qualsiasi KMS di scambiare certificati digitali con l'infrastruttura di sicurezza. La comunicazione avviene per 2 motivi.

- per richiedere un nuovo certificato digitale o per rinnovarlo.
- per verificare se un determinato certificato emesso da quella PKI è ancora valido.

È importante notare che diversi tipi di rete possono imporre alcune limitazioni alle prestazioni.

4.1.1 KMAC

Il KMAC è un identificatore univoco che può identificare il numero di serie della chiave e l'ID ETCS del entità KMC che genera la chiave. Ogni chiave ha un periodo di validità che è definito con una data di inizio validità seguita dai dati di fine validità del entità KMAC, la data deve essere codificata nel formato HH/DD/MM/YY, per le ore viene utilizzato il formato 24 ore. Se si vuole indicare un periodo di validità infinito viene usato un formato speciale 0xFFFFFFFF.
Le unità KMAC devono fare riferimento a un solo KMC e devono utilizzare solo il proprio KMC di riferimento per la gestione delle chiavi; inoltre le unità KMAC garantiscono che le operazioni di gestione delle chiavi non influiscano i collegamenti già stabiliti per la supervisione dei treni, il KMAC

non può modificare o eliminare alcuna chiave installata dal KMC d'origine a meno che non venga ordinato dal KMC, se una chiave di autenticazione viene aggiornata non potrà essere applicata a una connessione attiva.
I KMAC on-board devono contattare regolarmente il proprio KMC di riferimento, il quale deve essere contattato se una delle seguenti condizioni è verificata:

- L'apparecchiatura on-board ERTMS/ETCS è accesa e le prove di avviamento, se previste, sono state completate con esito positivo;
- Il tempo trascorso dall'ultima sessione conclusa con successo con il KMC d'origine è superiore a un periodo di tempo predefinito. Il periodo di tempo è definito dal KMC d'origine e deve essere compreso tra 1 ora e 1000 ore, con il valore predefinito di 10 ore;
- Il personale addetto alla manutenzione on-board del KMAC richiede un aggiornamento chiave;
- L'entità on-board KMAC rileva una chiave non valida o danneggiata. Se l'apparecchiatura on-board non è in grado di completare con successo la connessione con il proprio KMC di origine il KMAC on-board dovrà riprovare a stabilire la sessione con il proprio KMC di origine ogni 10 minuti.

4.1.2 KMC

Il KMC è il responsabile della generazione delle chiavi di autenticazione necessarie per stabilire una connessione sicura tra le entità track-side KMAC appartenente al suo dominio e qualsiasi entità on-board KMAC operante nello stesso dominio. Il KMC che emette o aggiorna una chiave è responsabile inoltre di garantire che il periodo di validità di questa chiave non si sovrapponga a nessun altro periodo di validità di qualsiasi altra chiave applicabile a qualsiasi connessione a per cui è necessaria una chiave di immissione corrente.
Nel momento in cui viene richiesta una chiave di autenticazione per stabilire una connessione sicura tra RBC appartenenti a domini KM diversi, il KMC è responsabile della generazione della chiave che deve essere concordata tra gli operatori dei diversi domini, così che il KMC identificherà in modo univoco tutte le sue chiavi generate con un numero di serie, anche se è possibile assegnare lo stesso valore KMAC a connessioni relative a diverse apparecchiature on-board il numero identificativo della chiave è univoco. Il KMC inoltre è responsabile dell'installazione, dell'aggiornamento e dell'eliminazione delle chiavi in tutti i KMAC appartenenti al proprio dominio e di elaborare le richieste di generazione, installazione, aggiornamento ed eliminazione delle chiavi da un KMC. Deve inoltre segnalare l'aggiornamento dello stato della chiave a un KMC che ha richiesto la generazione, l'aggiornamento, l'installazione o l'eliminazione delle chiavi.
Quando viene richiesto da un altro KMC di installare, aggiornare o eliminare delle chiavi, il KMC verificherà che tali chiavi siano state emesse dal KMC richiedente.
Per una gestione più controllata delle chiavi e trasparenza tra i KMC si è fatto in modo che il KMC sia in grado di controllare il database delle chiavi nelle entità KMAC appartenenti al suo dominio KM e recuperare le chiavi da eventuali casi di chiavi danneggiate. Questo deve essere fatto secondo le regole del dominio KM.

CAPITOLO 5

Interfaccia Online

5.1 Protocollo dell'applicazione

Per gestire le operazioni sulle chiavi nell'applicazione si è sviluppato un protocollo che consente la distribuzione, l'aggiornamento e l'eliminazione delle chiavi tra due KMC o tra entità KMC e KMAC; da ora in poi si identificherà con il termine "entità" un singolo elemento appartenete all'insieme delle apparecchiature on-board e trackside. Il protocollo fornisce i mezzi per richiedere operazioni sulle chiavi, eseguire un controllo di coerenza del database e informare sullo stato di distribuzione delle chiavi.

5.2 Specifiche sull'interfaccia dell'applicazione

L'interfaccia dell'online KMS ha le seguenti funzioni.

- Aggiungere chiavi;
- Cancellare chiavi;
- Cancellare tutte le chiavi;
- Aggiornare la validazione della chiave;
- Aggiornare la chiave;
- Controllare il database delle chiavi;
- Controllare lo stato del aggiornamento delle chiavi;
- Richiedere le operazioni sulle chiavi.

5.3 Algoritmo Checksum

L'algoritmo di checksum è usato per verificare la coerenza delle chiavi tra i database del KMC principale e le entità KMAC. La figura seguente 5.1 riporta il funzionamento generale dell'algoritmo di checksum.

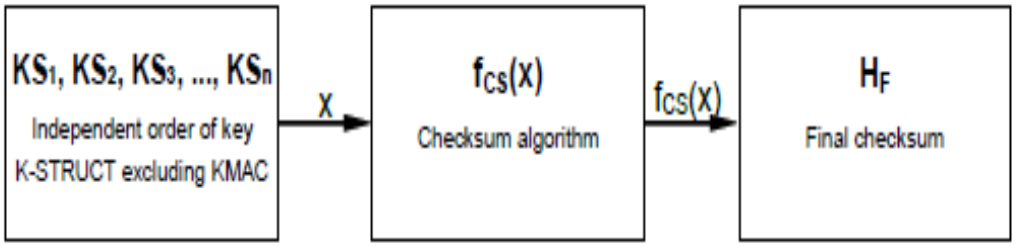

Figura 5.1: Meccanismo generale del checksum

nella figura 5.1 la x indica l'input dell'algoritmo. L'algoritmo, $f_{cs}(x)$, ha 2 funzioni principali:

1. Trovare le differenze tra le chiavi del KMC principale e un entità KMAC.
2. Produrre lo stesso risultato finale H_f indipendentemente dall'ordine della struttura delle chiavi KS. La formula che riassume il suo comportamento è : $f_{CS}(P_a(KS_1, KS_2, ...KS_n)) = f_{CS}(P_b(KS_1, KS_2, ...KS_n))$ dove P_a e P_b sono le differenti permutazioni delle chiavi.

L'algoritmo è illustrato e riportato nella figura seguente 5.2.

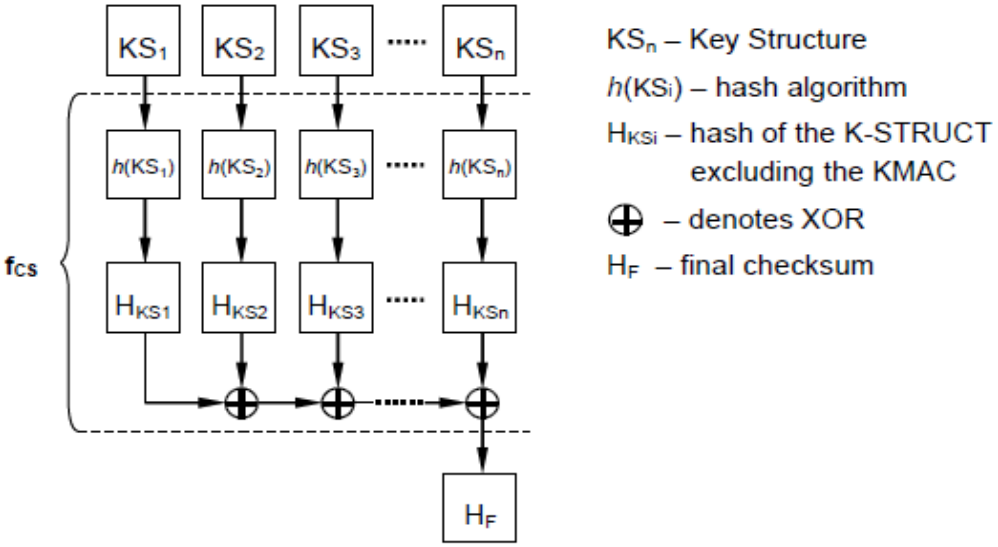

Figura 5.2: Definizione dell'algoritmo di checksum

Per l'algoritmo dell'hash è ustato usato l'MD4, per i dettagli del MD4 vedere [RFC-1320], l'input per l'algoritmo dell'hash ($h(KS_i)$) dipende dalla struttura delle chiavi KS_n. La struttura dei KS_n è riportata nella tabella 5.1:

Campo	Lunghezza	Descrizione
K-LENGTH	1	Lunghezza del KMAC
K-IDENTIFIER	8	Struttura che identifica in modo univoco il KMAC
PEER-NUM	2	Numero di entità KMAC al pari
ECTS-ID-ECP	4*PEER-NUM	Elenco delle entità KMAC collegate a questa chiave
VALID-PERIOD	8	inizio e fine validità per il KMAC

Tabella 5.1: Struttura dei KS_n

5.4 Definizione dei messaggi

Per eseguire le funzioni i messaggi che si scambiano i KMS devono essere strutturati in modo preciso affinché passino i controlli di struttura. I messaggi sono divisi in comandi, richiesta e notifiche:

- I messaggi per i comandi richiedono alcune modifiche al database delle chiavi del KMS ricevente;
- I messaggi per le richiesta richiedono solo una risposta del servizio di gestione delle chiavi del ricevente senza alcuna modifica del database delle chiavi;
- I messaggi di notifica sono usati per 4 scopi;
    1. Rispondere ai messaggi;
    2. Notifica di una connessione TLS stabilita;
    3. Notifica di uno stato di aggiornamento;
    4. Notifica fine aggiornamento.

Le seguenti figure riportano i tipi di messaggi e la loro direzione, entità di partenza e entità di arrivo, rispettivamente per richiedere comandi 5.3, per le richiesta 5.4 e notifiche5.5.

Message - Command	Message flow direction		
CMD_ADD_KEYS	KMC	→	KMS entity
CMD_DELETE_KEYS	KMC	→	KMS entity
CMD_DELETE_ALL_KEYS	KMC	→	KMAC entity
CMD_UPDATE_KEY_VALIDITIES	KMC	→	KMS entity
CMD_UPDATE_KEY_ENTITIES	KMC	→	KMS entity
CMD_REQUEST_KEY_OPERATION	KMC	→	KMC

Figura 5.3: Elenco dei comando

Message – Inquiry	Message flow direction		
INQ_REQUEST_KEY_DB_CHECKSUM	KMC	→	KMAC entity

Figura 5.4: Elenco delle richieste

Message – Notification	Message flow direction		
NOTIF_KEY_UPDATE_STATUS	KMC	→	KMC
NOTIF_ACK_KEY_UPDATE_STATUS	KMC	→	KMC
NOTIF_SESSION_INIT	KMC	→	KMS entity
	KMS entity	→	KMC
NOTIF_END_OF_UPDATE	KMC	→	KMS entity
NOTIF_RESPONSE	KMS entity	→	KMC
NOTIF_KEY_OPERATION_REQ_RCVD	KMC	→	KMC
NOTIF_KEY_DB_CHECKSUM	KMAC entity	→	KMC

Figura 5.5: Elenco dei messaggi di notifica

Formato e controllo dei messaggi

I messaggi sono espressi in formato binario e tutti i valori sono serializzati in ordine (Big Endian); per big endian si identifica la configurazione per cui il MSB ha l'indirizzo inferiore, ovvero il bit più significativo si trova in cima all'array. Questi messaggi consistono in un Message Header e un Message Body, vedi tabella 5.2 , il message header serve per specificare il tipo di messaggio nel body. La massima lunghezza del messaggio non può superare i 5000 byte.

Message Header	**Message Body**

Tabella 5.2: Struttura dei messaggi

Quando un KMS riceve un messaggio deve verificarne la struttura; se c'è qualche errore nell'header o nel messaggio deve scartare il messaggio e rispondere con una notifica riportando il tipo di errore. La verifica dell'header e della struttura del messaggio deve:

- Verificare che l'header del messaggio contenga l'identificativo univoco del ricevente;
- Verificare che l'header del messaggio contenga l'identificativo univoco dell'entità per la connessione corrente;
- Controllare che il valore di ogni campo sia nel range specificato;
- Verificare che il campo Lunghezza messaggio nell'intestazione corrisponda alla somma delle parti di cui è composto il messaggio, in modo tale che, durante l'analisi nessun dato venga letto all'esterno e nessun dato venga lasciato non analizzato;
- Controllare che la richiesta sia supportata;
- Controllare che l'header sia supportato.

Per ogni messaggio inviato online all'interfaccia del KMS deve esserci una chiave unica e univoca.

Message header

Tutti gli header dei messaggi devono avere la seguente struttura 5.3 ogni campo dell'header descrive le informazioni per connettersi e i comandi da mandare alle varie entità.

Campo	Dimensione	Valore	Descrizione del campo
Lunghezza del messaggio	4	[20,.., 500]	Descrive la lunghezza totale del messaggio compreso body ed header in byte
Versione dell'interfaccia	1	2	Versione dell'interfaccia
ID del ricevitore	4	ECTS_ID_EXP	Identifica il ricevitore in maniera univoca
ID del trasmettitore	4	ETCS_ID_EXP	Identificatore unico del trasmettitore
Numero di transazione	4	$[0..2^{32}-1]$	Identifica una specifica transazione con un particolare set di operazioni. Se si usa 0 indica che sono transazioni che non necessitano risposta come NOTIF_SESSION_INIT
Numero della sequenza	2	[0..65535]	Permette di controllare i messaggi e segnalare una eventuale sequenza errata.
Tipo di messaggio	1	0	CMD_ADD_KEYS
		1	CMD_DELETE_KEYS
		2	CMD_DELETE_ALL_KEYS
		3	CMD_UPDATE_KEYS_VALIDITIES
		4	CMD_UPDATE_KEYS_ENTITY
		5	CMD_REQUEST_KEY_OPERATION
		6	INQ_REQUEST_KEY_FB_CHECKSUM
		7	NOTIF$_{K}EY_UPDATE_STATUS$
		8	NOTIF_ACK_KEYS_UPDATE_STATUS
		9	NOTIF_SESSION_INIT
		10	NOTIF_END_OF_UPDATE
		11	NOTIF_RESPONSE
		12	NOTIF_KEY_OPERATION_REQ_RCVD
		13	NOTIF_KEY_DB
		[14..200]	RESERVED
		[201..255]	UNDEFINED

Tabella 5.3: Descrizione dei campi dell'header dei messaggi

Di seguito vengono riportate le descrizioni di ogni campo dell'header message.

La tabella 5.4 riporta la struttura del campo ETCS-ID-EXP dell'header.

Campo	Dimensione	Valore	Descrizione del campo
ETCS-ID typer	1		Tipo dell'ID-ETCS
ETCS-ID	3		ID dell' ETCS

Tabella 5.4: Descrizione dei campi dell'ETCS-ID-EP

La tabella 5.5 riporta la struttura del campo CMD_ADD_KEYS dell'header.

Campo	Dimensione	Valore	Descrizione del campo
REQ-NUM	2	[1...100]	Numero della K-STRUCT
K-STRUCT	*		La dimensione dipende dal numero di chiavi da aggiungere e dal numero di entità KMAC per chiave

Tabella 5.5: Descrizione dei campi del CMD_ADD_KEYS del header

La tabella 5.6 riporta i campi del K-STRUCT.

Campo	Dimensione	Valore	Descrizione del campo
K-LENGTH	1	24	Lunghezza della chiave in byte
K-IDENTIFIER	8		Struttura che identifica in modo univoco una chiave
ETCS-ID-EP	4		L'ID ETCS espanso dell'entità KMAC di destinazione
KMAC	K-LENGTH		Chiave di autenticazione
PERR-NUM	2	[1..1000]	Il numero di entità al pari che seguono questo campo. Almeno un'entità paritaria deve essere specificata in K-STRUCT
ETCS-ID-EXP[PEER-NUM]	4*PERR-NUM		Lista dei KMAC al pari collegati a quella chiave
VALID-PERIOD	8		Periodo di validità della chiave

Tabella 5.6: Descrizione dei campi del K-STRUCT

La tabella 5.7 riporta la struttura del K-IDENTIFIER.

Campo	Dimensione	Valore	Descrizione del campo
ETCS-ID-EXP	4		L'identità del KMC che ha emesso la chiave.
SNUM	4		Numero seriale della chiave

Tabella 5.7: Descrizione dei campi del K-IDENTIFIER

La tabella 5.8 riporta la struttura del campo CMD_DELETE_KEYS del header.

Campo	Dimensione	Valore	Descrizione del campo
REQ-NUM	2	[1...500]	Il numero di strutture K-IDENTIFIER successive.
K-IDENTIFIER [REQ-NUM]	8*REQ-NUM		Elenco di K-IDENTIFIER

Tabella 5.8: Descrizione dei campi del campo CMD_DELETE_KEYS del header

Il campo CMD_DELETE_ALL_KEYS è costituito solo dall'intestazione del messaggio.

La tabella 5.9 riporta la struttura del campo CMD_UPDATE_KEYS_VALIDITIES del header.

Campo	Dimensione	Valore	Descrizione del campo
REQ-NUM	2	[1...500]	Il numero di strutture K-VALIDITY successive.
K-VALIDITY [REQ-NUM]	16*REQ-NUM		Elenco della struttura K-VALIDITY

Tabella 5.9: Descrizione dei campi del CMD_UPDATE_KEYS_VALIDITIES del header

La tabella 5.10 riporta la struttura del campo K-VALIDITY.

Campo	Dimensione	Valore	Descrizione del campo
K-IDENTIFIER	8		Struttura che identifica in maniera univoca la chiave.
VALID-PERIOD	8		Periodo di validità

Tabella 5.10: Descrizione dei campi del K-VALIDITY

La tabella 5.11 riporta la struttura del campo CMD_UPDATE_KEY_ENTITIES del header.

Campo	Dimensione	Valore	Descrizione del campo
REQ-NUM	2	[1..250]	Numero di K-ENTITIES.
K-ENTITIES [PERR-NUM]	4*PEEr-NUM		Lista di entità KMAC collegate a questa chiave

Tabella 5.11: Descrizione dei campo del CMD_UPDATE_KEY_ENTITIES del header

La tabella 5.13 riporta la struttura del campo K-ENTITIES.

Campo	Dimensione	Valore	Descrizione del campo
REQ-NUM	2	[1..250]	Numero di K-ENTITIES.
PEER-NUM	2		Numero di entità KMAC al pari
ETCS-ID-EXP [PEER-NUM]	4*PEER-NUM		Lista delle entità KMAC collegate a questa chiave.

Tabella 5.12: Descrizione dei campo del CMD_UPDATE_KEY_ENTITIES del header

La tabella 5.13 riporta la struttura del campo CMD_REQUEST_KEY_OPERATION.

Campo	Dimensione	Valore	Descrizione del campo
ETCS-ID-EXP	4		Entità KMAC per la quale è richiesta un'operazione sulla chiave.
REASON	1	0	Nuovo treno operante nel dominio KM emittente.
		1	Modifica dell'area di intervento nel dominio KM emittente.
		2	Riduzione dei permessi pianificati nel dominio KM emittente.
		3	Si avvicina la fine del periodo di validità di alcune delle chiavi rilasciate
		[4..200]	Riservato
		[201..255]	Non definito
VALID-PERIDO	8		Campo da inserire solo se REASON = 2. La decorrenza del periodo di validità è uguale all'inizio del periodo di validità della chiave per la quale è stata emessa richiesta di riduzione del permesso programmato. La data di fine del periodo di validità deve essere fissata alla data richiesta per la riduzione del permesso programmato.
TECT-LENGTH	2	[0..1000]	Lunghezza testo opzionale.
TEXT	TEXT-LENGTH		Testo facoltativo per fornire alcune informazioni aggiuntive per una richiesta di operazione con la chiave (se TEXT_LENGTH > 0). Il testo è codificato utilizzando UTF-8.

Tabella 5.13: Descrizione dei campo del CMD_REQUEST_KEY_OPERATION.

Il messaggio di richiesta **INQ_REQUEST_KEY_DB_CHECKSUM** è usato per richiedere a un'entità KMAC di calcolare il checksum sul suo database di chiavi e riportare il risultato al KMC. Questo messaggio è costituito solo dall'intestazione del messaggio.

Struttura dei messaggi di notifica (NOTIF)

I messaggi di riposta riportano informazioni sull'esecuzione del comando. In base al tipo di messaggio, al campo e al suo valore.
Il messaggio **NOTIF_KEY_UPDATE_STATUS** riporta informazioni sullo stato della chiave in un KMC, come è possibile notare nella tabella 5.14 ogni valore fa riferimento a uno stato della chiave.

Campo	Grandezza	Valore	Descrizione
K-IDENTIFIER	8		Identificatore della chiave di cui si richiede lo stato
K-STATUS	1	1	Chiave installata
K-STATUS	1	2	Chiave aggiornata
K-STATUS	1	2	Chiave eliminata

Tabella 5.14: Descrizione dei campi del messaggio **NOTIF_KEY_UPDATE_STATUS**

Il messaggio **NOTIF_ACK_KEY_UPDATE_STATUS** è un messaggio utilizzato per indicare che il destinatario ha ricevuto il messaggio di notifica sullo stato delle chiavi.

NOTIF_SESSION_INIT viene usato per indicare l'inizio di una nuova sessione. Questo messaggio informa l'entità che lo riceve del numero di sequenza iniziale, della lista delle versioni dell'interfaccia e il tempo di time-out dell'applicazione ed è formato dai seguenti campi indicati in tabella 5.15.

Campo	Grandezza	Valore	Descrizione
N-VERSION	1	1	Versione dell'interfaccia supportata.
INTERFACE-VERSION	N-VERSION	2	Lista delle versioni supportate
APP-TIME-OUT	1	[5. . 255]	Time-out dell'applicazione

Tabella 5.15: Descrizione dei campi del messaggio **NOTIF_SESSION_INIT**

NOTIF_END_OF_UPDATE questo messaggio indica che tutte le richieste di aggiornamento sono state trasferite. Questo messaggio è inviato dopo che tutti i messaggi di aggiornamenti hanno avuto risposta.

Il messaggio **NOTIF_RESPONSE** riporta tutti i risultati di un messaggio di notifica o di un messaggio di comando. Ogni campo identifica il risultato o un errore per ogni richiesta. La tabella 5.16 riporta i vari casi possibili e i rispettivi valori.

Campo	Grandezza	Valore	Descrizione
RESPONSE	1	0	Se il comando mandato contiene una richiesta multipla, per esempio se voglio aggiungere più chiavi contemporaneamente (se il campo REQ_NUM è diverso da 1) "0" significa che la verifica del messaggio è andta a buon fine.
		1	Richiesta non supportata in riferimento alla specifica subset 5.3.2.7
		2	Errore nella lunghezza del messaggio
		3	L'ID di chi trasmette il messaggio non corrisponde all'ETCS-ID-EXP del entità KMS
		4	L'ID di chi riceve non conrrisponde con l'ETCS-ID-EXP delle entità KMS
		5	Versione interfaccia non supportata
		6	DB non recuperabile
		7	Processamento della richiesta fallito
		8	Il Checksun non corrisponde
		9	Il sequence number non corrisponde
		10	Il transaction number non corrisponde
		11	Errore nella formattazione
		[12...254]	Rivervato
		255	Altri errori
REQ-NUM	2	[0..500]	Il numero dipende dal valore di **NOTIFICATION_STRUCT**. Questo campo sarà "0" se se il campo **RESPONSE** e diverso da "0".
NOTIFICATION_STRUCT [REQ_NUM]	REQ_NUM		Lista delle strutture NOTIFICATION_STRUCT

Tabella 5.16: Descrizione campi del messaggio **NOTIF_RESPONSE**

La struttura e la descrizione dei campi della **NOTIFICATION_STRUCT**, uno dei campi del **NOTIF_RESPONSE** che riporta i risultati dei singole operazioni sulle chiavi, è riportata nella tabelle 5.17

Campo	Grandezza	Valore	Descrizione
RESULT	1	0	Richiesta processata con successo.
		1	Chiave sconosciuta : chiave non trovata nel DB dell'entità KMS.
		2	Superato il numero massimo di chiavi nel DB dell'entità KMS
		3	Richiesta di installare una chiave già presente nel DB. L'installazione non verrà eseguita.
		4	Chiave corrotta
		5	Expanded ETCS-ID non trovato
		[6..254]	Riservato
		255	Altri errori

Tabella 5.17: Descrizione campi **NOTIFICATION_STRUCT**

NOTIF_KEY_OPERATION_REQ_RCVD questo messaggio ha duplice funzione. Identifica sia quando il messaggio di comando CMD_REQUEST_KEY_OPERATION è stato ricevuto, sia il tempo massimo per rispondere al comando che richiede l'operazione da svolgere sulla chiave. Ha un solo campo MAXTIME da 2byte e indica il tempo massimo per rispondere al messaggio di comando (in ore).

L'ultimo messaggio di risposta è il **NOTIF_KEY_DB_CHECKSUM** che riporta il valore del Checksum delle chiavi nel DB di un KMAC, nel caso di database vuoto il checksum sarà uguale a 0.

Specifiche delle funzioni

Le sezioni seguenti specificano le funzioni necessarie per la gestione delle chiavi online tra due KMC o tra KMC e KMAC. Sarebbe possibile aggiungere funzioni ma solo localmente e se non interferiscono con questo sottoinsieme.

Add Key

Questa funzione è usata dal KMC per installare una o più chiavi di autenticazione nei KMAC o per scambiare chiavi con un KMC non appartenente al proprio dominio. La funzione deve definire:

- La chiave di autenticazione da installare;
- Il KMAC ricevente;
- La lista dei KMAC associate con la chiave che sta generando;
- Il periodo di validità associata alla chiave.

Per installare una o più chiavi in un KMS, il KMC deve inviare un comando con la dicitura "Aggiungi chiave", quando un KMS riceve il messaggio "Aggiungi chiave", che supera la verifica, risponde con un messaggio di notifica per ciascuna chiave indicandone il risultato.

Delete Keys

Questa funzione serve al KMC per:

- Cancellare una o più chiavi in un KMAC;
- Cancellare una o più chiavi di altro KMAC.

Per eliminare una o più voci chiave in un KMS, il KMC deve inviare un comando "Elimina chiavi" includendo la chiave da eliminare. Quando un KMS riceve il messaggio che supera la verifica della struttura del messaggio, deve rispondere con un messaggio di notifica con una risposta per ciascuna chiave indicandone il risultato dell'operazione. La cancellazione deve essere eseguita in modo tale che le chiavi cancellate non possano essere recuperate.

Delete All Keys

Questa funzione è usata dal KMC per cancellare tutte le chiavi conservate in un KMAC. Per cancellare le chiavi il KMC deve mandare un messaggio "Delete all Keys". Come per la funzione Delete keys ogni messaggio ricevuto dal KMAC deve rispondere con un notifica indicando il risultato dell'operazione.

Update Key Validity Periods

Questa funzione è usata dal KMC per:

- Aggiornare il periodo di validità di una chiave già esistente nel KMAC.
- Aggiornare il periodo di validità di una chiave già esistente nel KMC.

Per aggiornare il periodo di validità di una o più chiavi nel KMS, il KMC deve inviare un comando "Aggiorna periodi di validità delle chiavi" includendo una richiesta per chiave che deve essere aggiornata. Quando un KMS riceve un comando "Aggiorna periodi di validità della chiave" che supera la verifica dell'intestazione e della struttura del messaggio, deve rispondere con una notifica e una risposta per ogni aggiornamento del periodo di validità della chiave. Ciascuna risposta deve indicare il risultato dell'aggiornamento del periodo di validità e della corrispondente immissione della chiave. Il periodo di validità aggiornato dal comando sostituisce il precedente periodo di validità associato alla chiave corrispondente.

Update Key Entities

L'utilizzo di questa funzione permette:

- Aggiornare le liste dei KMAC collegati alle chiavi ancora valide in un KMAC;
- Aggiornare l'elenco dei KMAC collegate alle chiavi di un altro KMC.

Per aggiornare l'elenco dei KMAC per una o più chiavi in KMS, il KMC deve inviare un comando "Aggiorna chiave" e una richiesta per ogmi chiave che deve essere aggiornata. Quando un KMS riceve un comando "Aggiorna chiave" che supera la verifica dell'intestazione e della struttura del messaggio, deve rispondere con un messaggio di notifica con una risposta per ogni aggiornamento delle chiavi richieste dal comando. Ciascuna risposta indica il risultato dell'aggiornamento della chiave corrispondente. L'elenco dei KMAC aggiornato dal comando "Aggiorna chiave" sostituisce qualsiasi elenco precedentemente.

Check Key Database

Questa funzione viene utilizzata dal KMC per richiedere il checksum calcolato sul database delle chiavi di KMAC. Il checksum restituito viene utilizzato dal KMC per controllare lo stato del database delle chiavi di un KMAC. Per avviare un controllo dello stato del database delle chiavi in un KMAC, il KMC deve inviare un messaggio di richiesta "Richiedi checksum del database delle chiavi", quando riceve un messaggio "Richiesta di checksum del database di chiavi" dal proprio KMC di appartenenza, deve calcolare un checksum sul proprio database di chiavi e rispondere con un messaggio di notifica che riporta il checksum calcolato.

Report Key Update Status

Questa funzione viene utilizzata dal KMC per segnalare una modifica dello stato di una chiave in KMAC nel suo dominio KM al KMC che ha emesso la chiave. Lo stato della chiave potrebbe essere cambiato a causa di una richiesta del KMC che ha emesso la chiave o a causa di eventi nel dominio KM a causa del KMAC. Quando un KMC ha installato con successo una chiave emessa da un altro KMC, deve segnalarlo al KMC emittente. Quando una KMC ha aggiornato con successo il periodo di validità o l'elenco del KMAC per una chiave emessa da un altro KMC, deve segnalarlo alla KMC emittente, a meno che non vi sia un aggiornamento in sospeso per questa chiave. Quando un KMC ha eliminato con successo una chiave emessa da un altro KMC, nell KMAC pertinente e nel proprio database di chiavi, deve segnalarlo al KMC emittente. Se una chiave è stata eliminata senza essere mai

stata installata in un KMAC, il KMC risponderà dopo aver eliminato la voce della chiave dal proprio database di chiavi. Quando un KMC riceve un messaggio di notifica "Segnala stato aggiornamento chiave" da un altro KMC, aggiorna lo stato della chiave nel suo database e risponde che lo stato segnalato della chiave è stato preso in considerazione. La gestione dei casi degradati di aggiornamento chiave rientra nell'ambito del KMC d'origine dal KMAC e la mancata installazione, eliminazione o aggiornamento di una chiave in un KMAC non viene segnalata alla KMC emittente.

Request Key Operation

Questa funzione viene utilizzata dal KMC per richiedere a un KMC emittente di generare, aggiornare o eliminare chiavi per un KMAC appartenente al dominio KM richiedente. La richiesta deve specificare uno dei seguenti motivi per l'operazione chiave:

- Nuovo treno operante nel dominio KM emittente;
- Modifica dell'area di intervento nel dominio KM emittente;
- Riduzione dei permessi programmati nel dominio KM emittente (data di fine operatività del KMAC nel dominio KM);
- In prossimità della fine del periodo di validità di alcune chiavi emesse.

Per richiedere a un altro KMC di eseguire un'operazione con la chiave, il KMC deve inviare un messaggio "Richiesta operazione con la chiave" che include l'identità del KMAC per la quale è richiesta l'operazione con la chiave. Quando un KMC emittente riceve un comando "Richiesta operazione chiave" che supera la verifica dell'intestazione e della struttura del messaggio, deve rispondere con un messaggio di notifica indicante che la richiesta dell'operazione chiave è stata ricevuta e includendo il tempo massimo richiesto per rispondere a la richiesta. Il KMC emittente può rispondere a una richiesta di operazione con la chiave aggiungendo, aggiornando o eliminando una chiave. Nel caso in cui il KMC non sia in grado o non sia autorizzato a eseguire l'operazione richiesta entro il tempo indicato, questo non viene segnalato al KMC richiedente. Trascorso questo tempo, la situazione deve essere gestita da qualche procedura operativa.

5.5 Gestione del flusso di dati

Protocollo della connessione tra le entita'

Affinché le entità possano comunicare tra loro, si sono suddivisi i compiti: Il KMC che stabilisce la connessione tra i KMAC trackside, mentre il KMAC è il responsabile per la connessione on-board tra i KMC. La connessione tra le entità del KMS deve essere stabilita solo per mandare le richiesta, i comandi e gli aggiornamenti delle chiavi tramite messaggio di notifica. Per stabilire una connessione TLS tra 2 KMS tutte e due le entità devono inviare un messaggio NOTIF_SESSION_INIT: Quando si riceve questo messaggio la connessione è considerata stabilita a livello applicazione. Il messaggio NOTIF_SESSION_INIT deve includere la sequenza iniziale usata per la gestione, la lista delle versioni dell'interfaccia e il time-out. Esso viene definito prima e viene impostato ai KMC tramite il KMC che stabilisce la connessione. Il KMS non manda tutti gli altri messaggi finché non riceve il NOTIF_SESSIONS_INIT dalle altre entità. Dopo aver ricevuto e mandato il messaggio di notifica, si stabilisce la connessione TLS nel caso in cui abbiano la stessa versione dell'applicazione. Una volta stabilita la connessione, ogni entità controlla il time-out dell'applicazione e resetta il timer ad ogni ricezione.

Trasmissione dati

Una volta stabilita la connessione e aver inviato un messaggio per il quale è prevista una risposta, il KMC non invierà nessun altro messaggio fino a quando non avrà ricevuto una risposta con lo stesso numero di transazione del messaggio inviato. Il KMS che risponde a un messaggio ricevuto, identificato da un numero di transazione, deve utilizzare lo stesso numero di transazione del messaggio a cui risponde. Ogni transazione ha un numero diverso.

Rilasso della connessione

Completate tutte le transazioni il KMC invierà un messaggio di NOTIF_END_OF_UP_DATE e terminerà la connessione. Se la connessione tra un KMAC e un KMC viene terminata prima che il KMC abbia emesso il messaggio di fine connessione, la transazione non sarà eseguita. Quando una connessione viene ristabilita con un KMAC con il quale vi è stato una connessione terminata, il KMC può verificarne lo stato del Data Base (DB) inviando un messaggio INQ_CHECK_KEY_DB e utilizzando il checksum restituito per verificare se una transazione non riconosciuta sia stata elaborata o meno.

Gestione degli errori

Se un'entità che ha mandato il messaggio per la connessione (NOTIF_SESSION_INIT) non riceve dall'entità con la quale si vuole connettere entro 15 secondi, la connessione TLS deve essere terminata dal KMS che rivela il timeout. Alla ricezione del messaggio, il KMS controlla il numero sequenziale prima del numero di transazione: se il numero della sequenza di un messaggio ricevuto non è consecutivo al precedente, il KMS che rileva questo problema deve inviare il messaggio NOTIF_RESPONSE segnalando la mancata corrispondenza del numero sequenziale, quindi terminare la connessione. Un altro errore possibile avviene quando il KMS controlla il numero di transazione nei messaggi ricevuti in risposta a un messaggio inviato: se questo numero di transazione non corrisponde al numero nel messaggio inviato, il KMS invierà un messaggio NOTIF_RESPONSE segnalando la mancata corrispondenza del numero di transazione e quindi rilascerà la connessione.
Le seguenti immagini mostrano il comportamento per i relativi messaggi. La configurazione iniziale di ogni comando è la stessa. Appena viene stabilita la connessione entrambe le entità che comunicano inviano una NOTIF_SESSION_INIT con il numero sequenziale iniziale. Questo fa sì che il KMC mandi il comando che vuole eseguire e non invii altri messaggi fintantoché non riceva il NOTIF_RESPONSE

corrispondente al comando inviato precedentemente. Il KMAC elabora i comandi e risponde con un NOTIF_RESPONSE utilizzando lo stesso numero di transazione. La figura 5.6 mostra come aggiungere, cancellare o aggiornare le chiavi.
Terminati i comandi, il KMC risponde con un messaggio di fine connessione NOTIF_END.

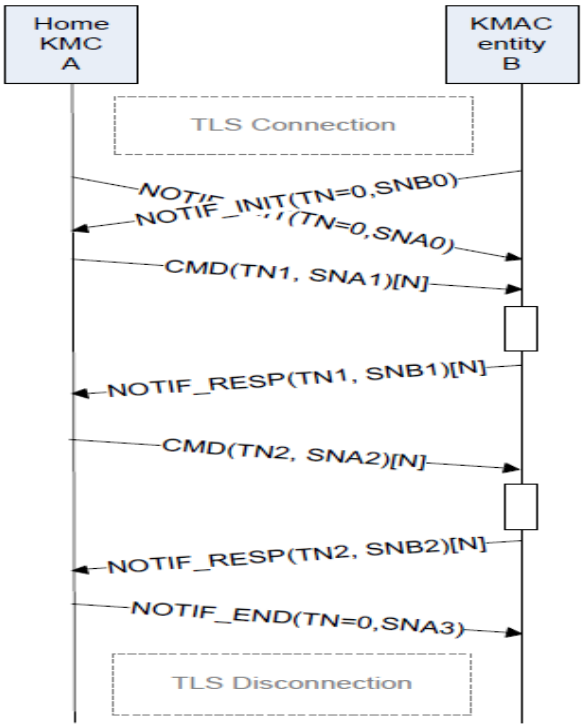

Figura 5.6: KMC-KMAC gestione delle chiavi

La figura 5.7 mostra come il KMAC termina una sessione anomala.
Se il KMAC on-board ha bisogno di terminare la connessione manda un messaggio di NOTIF_SESSION_ABORT e termina la connessione. Il KMC può determinare in base ai messaggi dell'entità di bordo del KMAC che il primo comando è stato eseguito, ma il secondo no. Quando una sessione viene ristabilita con l'entità KMAC, il KMC può riprendere l'aggiornamento.

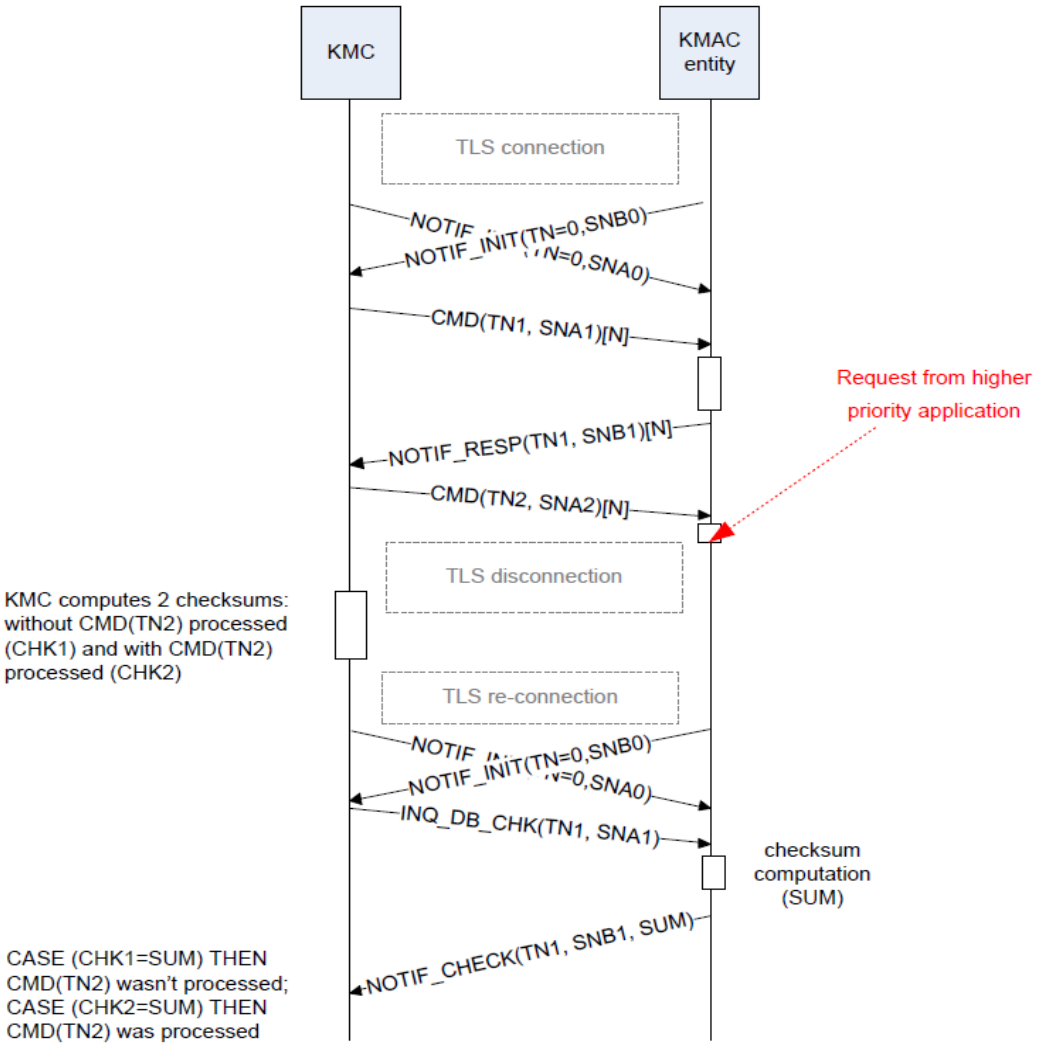

Figura 5.7: KMC-KMAC: terminazione sessione anormale

La figura 5.8 mostra come aggiungere, cancellare o aggiornare le chiavi di un KMAC proveniente da un altro dominio.
Il metodo di connessione iniziale resta lo stesso, solamente che in questo caso vengono coinvolte due KMC: una principale (Home KMC) e un'altra KMC (Issuring KMC A). Le due KMC stabiliscono la connessione e si scambiano i comandi con i relativi messaggi di notifica.In seguito, il KMC principale stabilisce la connessione con l'entità KMAC, processa il comando dato dal KMAC esterno rispondendo con un NOTIF_RESPONSE e usando lo stesso numero di transazione. Dopo che il KMC principale riceve il NOTIF_RESPONSE per i comandi da eseguire, termina la connessione con il KMAC e stabilisce una nuova TLS con il KMC controllando lo stato delle dei comandi da lui inviati. Finite tutte la transazioni, il KMC che ha emesso i comandi manda un messaggio di NOTIF_END_OF_UPDATE e termina la connessione.

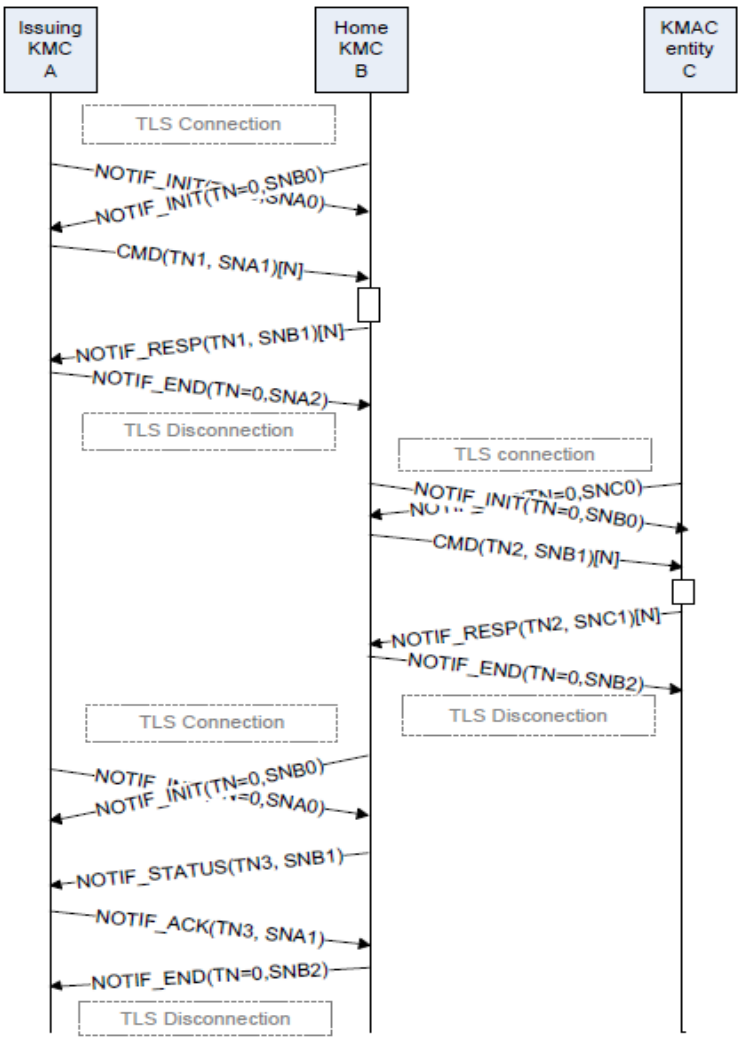

Figura 5.8: KMC-KMC gestione chiavi

La figura 5.9 mostra come vengono gestiti i time-out durante le connessioni.
Come nei casi precedenti, viene stabilita la connessione e mandato il messaggio NOTIF_SESSION_INIT. Messe in comunicazione, tutte e due le entità controllano: il tempo tra l'invio e la ricezione del comando; la sequenza e il numero di transazioni. Se il messaggio di inizio connessione di una delle due entità non viene ricevuto, l'entità che non ha ricevuto il messaggio aspetta per un periodo di tempo dettato dal time-out: se non riceve niente in quel lasso di tempo, allora termina la connessione. Nel frattempo, l'entità che ha perso il messaggio rilascerà la connessione per il tempo dettato dal time-out dell'applicazione.

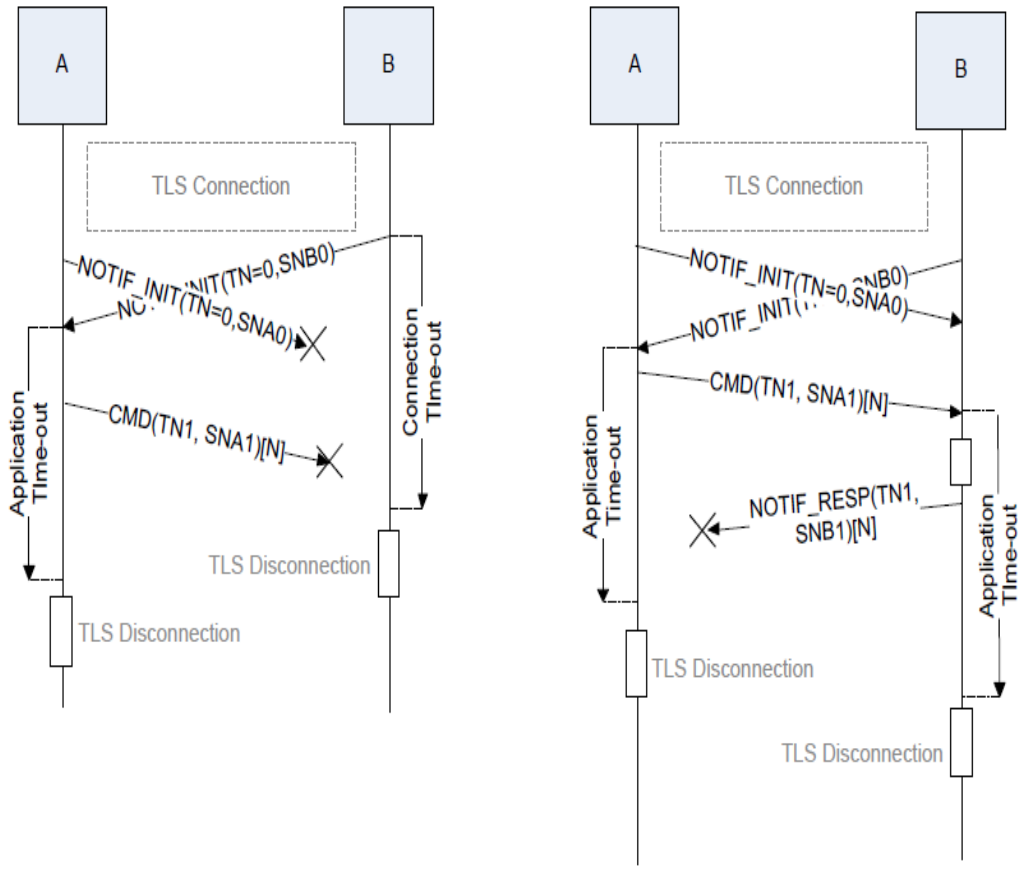

Figura 5.9: Gestione del supervisore del timeout

Le figure 5.10 5.11 5.12 mostrano cosa succede quando vi è un errore nel numero di sequenza e nei numeri di transazione.
Stabilita la connessione, entrambe le entità controllano il loro numero sequenziale e il numero di transazione. Come nella figura precedente 5.9, se uno dei due messaggi non viene ricevuto le entità aspettano un lasso di tempo pari al time-out e poi terminano la connessione. Se invece, come in 5.11, una delle due entità trasmette un numero di sequenza sbagliato, chi riceve il numero errato manda un messaggio di NOTFI_RESPONSE, dove specifica che il numero è errato e termina la connessione. Lo stesso succede per il numero di transazione: cambia solo l'header del messaggio di notifica.

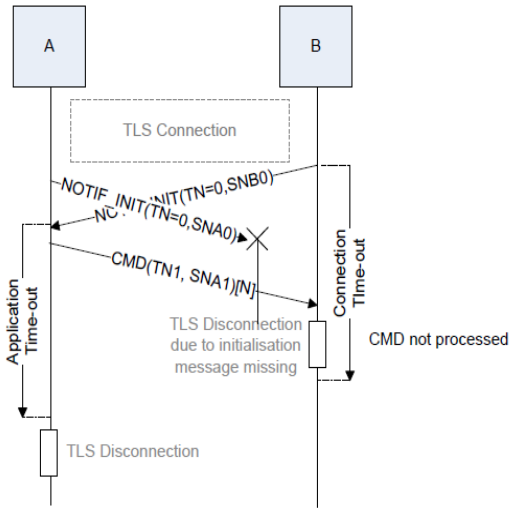

Figura 5.10: Errore sequenza durante una connessione

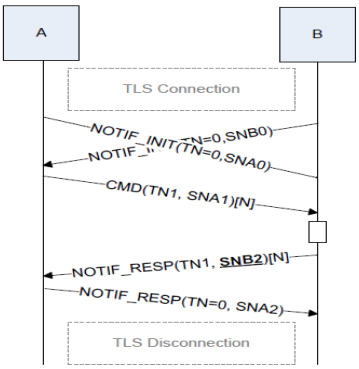

Figura 5.11: Errore nel numero della sequenza durante una connessione

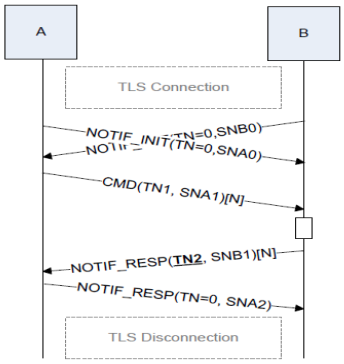

Figura 5.12: Errore nel numero di transazioni

Generatore di numeri casuali

Per rendere sicuri e crittografati gli scambi, si è deciso si usare un generatore di numeri casuali o pseudo-casuali: esso permette di generare chiavi e protocolli di comunicazione crittografati sicuri.

Tale generatore deve essere utilizzato per:

- Generare una coppia di chiavi pubblica/privata;
- Generare una chiave precondivisa utilizzando la soluzione TLS-PSK;
- Generare il seriale del KMAC;
- Eseguire la procedura di handshake TLS.

Nell'utilizzo di un generatore di numeri pseudo-casuali, è necessario prestare particolare attenzione al processo di inizializzazione e alla segretezza del seme del generatore di numeri pseudo-casuali. Il generatore di numeri casuali deve soddisfare i requisiti del European Union Agency for Cybersecurity (ENISA).

5.6 Interfaccia della sicurezza

Osservazione sull'interfaccia della sicurezza

Per ottenere riservatezza, autenticità e integrità del materiale crittografico è stato scelto il protocollo Transport Layer Security (TLS). L'autenticazione deve essere garantita o utilizzando certificati di una Public Key Infrastructure (PKI) o utilizzando chiavi segrete precondivise Pre Shared Key (PSK), il PSK può essere più conveniente dal punto di vista della gestione delle chiavi in piccoli domini KM. In tali domini potrebbe essere più semplice utilizzare chiavi precondivise piuttosto che impostare a infrastruttura a chiave pubblica.
Il KMS può supportare TLS-PSK o come alternativa il TLS-PKI, ma solo per l'uso all'interno di un dominio KM e non tra due KMC. La scelta di utilizzare un protocollo TLS-PSK o TLS-PKI dipende dal task che viene chiamato durante l'esecuzione.

TLS

Il TLS è un protocollo crittografico usato nel campo delle telecomunicazioni e dell'informatica che permette una comunicazione sicura dalla sorgente al destinatario. Il protocollo TLS lavora in 2 fasi. Nella prima fase vi è l'handshake per l'autenticazione del client e del server e scambiarsi le informazioni per la criptografia. Nella seconda fase vi è lo scambio di dati usando le chiavi e gli algoritmi definiti nella fase 1. L'handshake gestisce anche l'autenticazione, che di solito consiste nel fatto che il server dimostri la propria identità al client e viceversa. Ciò è possibile tramite le chiavi pubbliche. Le chiavi pubbliche sono chiavi di crittografia che utilizzano la crittografia unidirezionale, il che significa che chiunque disponga della chiave pubblica può decodificare i dati crittografati con la chiave privata del server per garantirne l'autenticità, ma solo il mittente originale può crittografare i dati con la chiave privata. La chiave pubblica del server fa parte del suo certificato TLS. Una volta che i dati sono crittografati e autenticati, vengono quindi firmati con un codice di autenticazione del messaggio (MAC). Il destinatario può quindi verificare il MAC per garantire l'integrità dei dati.

TLS-PSK

TLS-PSK si basa sullo scambio segreto di una chiave precondivisa dal KMC per ogni KMC e ogni KMAC per autenticare l'entità. La distribuzione e l'installazione di chiavi segrete precondivise deve essere supportata da procedure atte a garantirne la segretezza e l'autenticità. Poiché sia i client TLS che i server TLS possono avere chiavi precondivise con diverse entità, è necessario sapere quale chiave utilizzare. Pertanto, il client TLS indica quale chiave utilizzare includendo una "identità PSK" nel messaggio ClientKeyExchange.

TLS-PKI

TLS-PKI si basa sullo scambio di certificati digitali gestiti e distribuiti da un autorità di certificazione esterna (CA) per autenticare l'entità Il certificato della Certificate Authority (CA) di origine deve essere installato in tutte le entità utilizzando procedure operative a garanzia della sua autenticità. Le CA possono essere organizzate in una struttura ad albero gerarchica con certificati CA emessi da una CA di livello superiore. Questa struttura ad albero ha una singola radice, chiamata "CA radice" e tutti i client devono conoscere il certificato della CA radice. La figura 5.13 sottostante riporta la gerarchia dei certificati PKI.

Figura 5.13: Definizione del algoritmo di checksum

Funzioni di consegna dei certificati del client

Per la generazione di certificati esistono 3 possibilità:

1. Primo certificato : Generare un primo certificato con una nuova chiave pubblica;
2. Rinnovo certificato : Generare un nuovo certificato con una chiave preesistente;
3. Ridigitare certificato: Generare un nuovo certificato con una nuova chiave da un certificato pre-validato.

Per il KM si è deciso di utilizzare solamente o generare un primo certificato con una nuova chiave pubblica o ridigitare un certificato con una nuova chiave da un certificato pre-validato.
In caso di prima richiesta di certificato, il client PKI si autentica utilizzando le informazioni segrete condivise per creare il campo "protezione" contenuto nel messaggio "Richiesta di certificato".
In caso di richiesta di ridigitare un certificato, il client PKI dispone di un certificato valido e deve utilizzare tale certificato valido per autenticarsi quando richiede un nuovo certificato. Il client PKI dovrà creare il campo "protezione" contenuto nel messaggio "Richiesta certificato" utilizzando la chiave privata associata al proprio certificato valido.
Se il server PKI considera valida una richiesta di certificato da un client PKI, il server PKI firmerà e consegnerà un nuovo certificato al client PKI se questa operazione va a buon fine il client PKI deve mandare una conferma di ricezione certificato al server PKI. Se diversamente la richiesta del certificato non è valida il server PKI deve mandare una risposta negativa.

Interfaccia di trasporto

Panoramica del protocollo di trasporto

Per il trasporto dei certificati si e scelto di utilizzare il protocollo Transmission Control Protocol (TCP), dato che il protocollo TLS è un livello sopra lo stack del protocollo TCP/IP, le entità KMS devono essere in grado di stabilire o accettare connessioni TCP da entità accoppiate al fine di implementare le interfacce. I KMS devono anche essere in grado di stabilire connessioni TCP con la PKI poiché la distribuzione e la convalida dei certificati digitali si basano su TCP/IP. L'interfaccia di rete principale deve essere in grado di connettersi e scambiare dati utilizzando il Protocollo TCP su protocollo IP. Dobbiamo usare interfacce preesistenti perché le risorse di rete sono gestite da un componente esterno con l'accesso concesso tramite Application Programming Interface (API) fornita dal cliente. In questo modulo intendiamo gestire le connessioni e le interazioni che implicano l'operazione di lettura e scrittura sulla rete tramite un canale non protetto. Stabilire connessioni end-to-end a livello di trasporto dall'entità KMAC on-board comporta uno specifica sul indirizzo una definizione dei parametri TCP.
Per l'indirizzo si è usata una Domain Name System (DNS) per risolvere l'indirizzo IP del KMC, Certificate Authority (CA) e Registration Authority (RA). Il DNS traduce i nomi di dominio in indirizzi IP, in modo che i browser possano caricare le risorse Internet.Ogni entità collegato a Internet è dotato di un indirizzo IP univoco, che altre macchine usano per trovarlo. I server DNS eliminano la necessità di memorizzare gli indirizzi IP. Per RA e CA, il Fully Qualified Domain Name (FQDN) da utilizzare deve essere configurato in ciascuna entità KMS. Un FQDN è un nome non ambiguo che specifica la posizione assoluta di un nodo all'interno della gerarchia dell'albero DNS.
Per la definizione dei paramatri del TCP deve essere seguita una specifica data nel [Subset-037].

CAPITOLO 6

Simulazioni dell'applicazione

6.1 Simulazione della connessione TLS_PSK

Per sviluppare il programma di test si è scritto un programma in C++ con una classe per ogni macro funzione viste nei capitoli precedenti. Ogni macro classe richiama una funzione e ne riporta i relativi messaggi e le relative risposte del'OKM. Uno dei primi test fatto dopo lo sviluppo del codice è il test di connessione usando il protocollo TLS-PSK, di seguito viene riportato il log file dove vengono riportati i vari passaggi di verifica e in particolare quello relativo alla connessione e allora scambio dei certificati.

```
KM : KM_Init() -- Starting State Machine Initialization/Reset
KM : KM_Init() -- State Machine Initialization/Reset OK --
KEY : KEYM_init() -- EXECUTING KEY MANAGER INIT
KM : KM_Init() -- Starting Network Manager Initialization/Reset
NM : NM_disconnect() -- Clear text connection, no need to free SSL CTX
NM : NM_disconnect() -- Clear text connection, no need to free SSL session
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: CLOSED
TCP : TCPCLIENT_genericRead() -- TCP ports already closed. Nothing to do.
KM : KM_Init() -- Network Manager Initialization/Reset OK --
KM : KM_Init() -- Starting Security Manager Initialization/Reset
KM : KM_Init() -- Security Manager Initialization/Reset OK --
TEST : Start initKM()
KM : SUCCESS loading ETCSID in KM_setETCSID
TEST : Start KM_setInfoHomeKMC ()
KM : KM_setNetworkContext() -- OK --
TEST : Start KM_setSecurityContextDN ()
KM : KM_setSecurityContextDN() -- DNs load OK
TEST : Start KM_setSecurityContextPSK ()
SEC : SEC_setPskCredential() -- TLS method init OK --
SEC : SEC_setPskCredential() -- SSL context struct init OK --
SEC : SEC_setContextPSK() -- Set PSK credentials OK
KM : KM_setSecurityContextPSK() -- Set PSK context OK --
TEST : Start KM_setSecurityContextPKI ()
SEC : SEC_setPkiCredential() -- TLS method init OK --
SEC : SEC_setPkiCredential() -- SSL context struct init OK --
```

Questa prima parte del log file è la parte di inizializzazione dell'applicazione dove viene anche settato il tipo di connessione in questo caso TLS-PSK.

```
1  SEC : SEC_setPkiCredential() -- load RSA CA certificate PATH: ./support_TLS/x509/ca-
       chain.pem --
2  SEC : SEC_setPkiCredential() -- load CA certificate OK --
3  SEC : SEC_loadX509CertFromLocalPath() -- BIO new OK --
4  SEC : SEC_loadX509CertFromLocalPath() -- BIO read Cert file OK --
5  SEC : SEC_loadX509CertFromLocalPath() -- Create X509 Cert structure OK --
6  SEC : SEC_loadCaLocalCert() -- Load the local CA certificate from path OK
7  SEC : SEC_validateLocalCaCertificate() -- Load CA certificate OK
8  SEC : SEC_validateCaLocalCertificate() -- Check CA certificate expiration status
9  SEC : SEC_checkCertRevokeTime() -- Starting local Cert revoke time
10 SEC : SEC_checkCertRevokeTime() -- Get current time OK -- Time_now : 1665586340
       seconds
11 SEC : SEC_checkCertRevokeTime() -- Certificate revoke time : 1691562432 seconds
12 SEC : SEC_checkCertRevokeTime() -- Certificate renew time : 1691526432 seconds
13 SEC : SEC_checkCertRevokeTime() -- Cert expiration time check: OK
14 SEC : SEC_validateLocalCaCertificate() -- Check CA certificate expiration status OK
15 SEC : SEC_verifyCertificateChainAgainstCA() -- CA certificate valid --
16 SEC : SEC_loadClientKeyAndCert() -- load client certificate PATH: ./support_TLS/x509/
       client1.pem --
17 SEC : SEC_loadClientKeyAndCert() -- load client certificate OK --
18 SEC : SEC_loadClientKeyAndCert() -- load local private-key PATH: ./support_TLS/x509/
       client1-key.pem --
19 SEC : SEC_loadClientKeyAndCert() -- load local private-key OK --
20 SEC : SEC_loadClientKeyAndCert() -- local certificate and private-key matches --
21 SEC : SEC_setPkiCredential() -- Load local keys and certificate OK --
22 SEC : SEC_loadX509CertFromLocalPath() -- BIO new OK --
23 SEC : SEC_loadX509CertFromLocalPath() -- BIO read Cert file OK --
24 SEC : SEC_loadX509CertFromLocalPath() -- Create X509 Cert structure OK --
25 SEC : SEC_verifyLocalCertificateChain() -- Create X509 store object OK --
26 SEC : SEC_verifyLocalCertificateChain() -- Create X509 store context OK --
27 SEC : SEC_verifyLocalCertificateChain() -- Load CA certificate into X509 struct OK --
28 SEC : SEC_verifyLocalCertificateChain() -- X509 store context init OK --
29 SEC : SEC_verifyLocalCertificateChain() -- Certificate is valid
30 SEC : SEC_setPkiCredential() -- Verify local certificate against CA OK --
31 SEC : SEC_setContextPKI() -- OK
32 KM : KM_setSecurityContextPKI -- Set PKI context OK --
33 STORAGE : STORAGE_openFile() -- File open OK: ./keys-storage.bin
34 STORAGE : STORAGE_readData() -- Reading file: ./keys-storage.bin
35 STORAGE : STORAGE_readData() -- Read file OK: ./keys-storage.bin
36 TEST : testFillKeyStorageFromFile() -- OK -- Key storage load from file OK
37 KEY : Load page : 0
38 KEY : Key loaded : 3
39 TEST : debugInit() -- key storage page 0 settingKeyRelation OK
40 TEST : debugInit() -- Key storage checksum:
41  00  00  00  00  2A  8F  A6  B3  36  62  6A  35  4F  69  2F  ED  DC  54  5D  4B
42 SM : Received -- KM_ACTION_SET_CONNECTION_REQUEST: 2 -- EXTERNAL action
43 SM : KM_ACTION_SET_CONNECTION_REQUEST propagated
44 TMAIN_TEST : Executing run task...
```

In questa parte vengono caricati e convalidati i vari certificati per far si che la connessione avvenga solamente tra entità che condividono lo set di certificati o un loro sottoinsieme. I certificati vengono caricati seguendo un percorso predefinito in memoria dove vi sono salvati tutti i certificati validi ai fini della connessione, in questo caso **./support_TLS/x509/ca-chain.pem**. In seguito vengono validati i certificati caricati precedentemente per verificare che non ci siano certificati scaduti o non più validi. In finite viene richiamata la funzione checksum che riporta lo stato del DB dell'entità KMAC e lo confronta con gli altri DB per verificare che tutti siano aggiornati allo stesso stato, visibile in riga 41.

```
SM : Executing -- ST_MACHINE_STATE_INIT -- state
SM : SM_init_State() done.
SM : -- ST_MACHINE_STATE_INIT -- state running exit status: 0
SM : Received -- ST_MACHINE_E_INIT_DONE: 2 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE


TMAIN_TEST : Executing run task...


SM : Executing -- ST_MACHINE_STATE_IDLE -- state
SM : -- ST_MACHINE_STATE_IDLE -- state running exit status: 0
SM : Received -- ST_MACHINE_E_CONNECT_REQ: 5 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE


TMAIN_TEST : Executing run task...


SM : Executing -- ST_MACHINE_STATE_CONNECTING_TO_HKMC -- state
NM : NM_openConnection() -- Timer enabled
NM : NM_openConnection() -- OK -- Try to open TCP port
TCP : TCPCLIENT_openTcpPort() -- Socket creation OK --
TCP : TCPCLIENT_openTcpPort() -- set Server IP: 127.0.0.1 OK --
TCP : TCPCLIENT_openTcpPort() -- establish TCP/IP connection to the SSL Server OK --
NM : NM_openConnection() -- TCP port open OK
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_TLS_HANDSHAKE --
SM : -- ST_MACHINE_STATE_CONNECTING_TO_HKMC -- state running exit status: 0
SM : Received -- ST_MACHINE_E_TLS: 15 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE


TMAIN_TEST : Executing run task...


SM : Executing -- ST_MACHINE_STATE_TLS_HANDSHAKE -- state
NM : NM_startSecureHandshake() -- Starting TLS handshake --
NM : NM_tryHandshake() -- Starting TLS handshake
NM : NM_tryHandshake() -- PKI handshake selected
SEC : SEC_loadX509CertFromLocalPath() -- BIO new OK --
SEC : SEC_loadX509CertFromLocalPath() -- BIO read Cert file OK --
SEC : SEC_loadX509CertFromLocalPath() -- Create X509 Cert structure OK --
SEC : SEC_validateLocalCertificate() -- Check certificate's validation time
SEC : SEC_checkCertRevokeTime() -- Starting local Cert revoke time
SEC : SEC_checkCertRevokeTime() -- Get current time OK -- Time_now : 1665586342
    seconds
SEC : SEC_checkCertRevokeTime() -- Certificate revoke time : 1691604552 seconds
SEC : SEC_checkCertRevokeTime() -- Certificate renew time : 1691568552 seconds
SEC : SEC_checkCertRevokeTime() -- Cert expiration time check: OK
NM : NM_connectPKI() -- Exec connect PKI
TLS : TLS_init() -- OpenSSL lib init OK
SEC : SEC_getContextPKI() -- Get SSL PKI context OK --
TLS : TLS_setPKICipherSuites() -- Set cipher suites OK --
TLS : TLS_setEllipticCurves() -- Set elliptic curves OK --
TLS : TLS_setOcspStatusRequestExtension() -- Set OCSP status request extension OK
TLS : TLS_loadCredPKI() -- SSL create session OK --
NM : NM_connectPKI() -- Set socket file descriptor OK --
TLS : TLS_execHandshake() -- Starting TLS Handshake
```

```
62 TLS : TLS_execHandshake() -- TLS Handshake OK
63 SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_RESPONDING_TO_HKMC --
64 SM : -- ST_MACHINE_STATE_TLS_HANDSHAKE -- state running exit status: 0
65 SM : Received -- ST_MACHINE_E_RESPOND_REQ: 10 -- INTERNAL event
66 TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
67 TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
68 TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
69 TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
70
71
72 TMAIN_TEST : Executing run task...
73
74
75 SM : Executing -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state
76 KEY : Generate NOTIF_SESSION_INIT message
77 PARSER : Parsing NOTIF_SESSION_INIT
78 SM : Sending Response of -- 23 bytes --
79  00  00  00  17  02  02  44  55  66  01  11  22  33  00  00  00  00  00  00  09  01
    02  FF
80 NM : NM_writeData() -- Sending data to secure write function
81 SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG --
82 SM : -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state running exit status: 0
83 SM : Received -- ST_MACHINE_E_WAIT_MSG: 14 -- INTERNAL event
84 TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
85 TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
86 TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
87 TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
88 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
89
90
91 TMAIN_TEST : Executing run task...
92
93
94 SM : Executing -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG -- state
95 NM : Start Application Timer: 15 sec
96 NM : Running Application Timer
97 TCP : TCPCLIENT_readMessage() -- Start reading message
98 TCP : TCPCLIENT_readMessage() -- Reading data from secure read function
99 TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
100 TLS : TLS_secureRead() -- Max TCP blocking time reached
101 TCP : TCPCLIENT_readMessage() -- Read message duration: 201 milliseconds
102 SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG --
103 SM : -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG -- state running exit status: 0
104 SM : Received -- ST_MACHINE_E_NOTHING: 0 -- INTERNAL event
105 TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
106 TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
107 TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
108 TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
109 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
110 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
```

Questa ultima sezione del log file invece riporta le varie fasi della macchina a stati del KM. Le fasi della macchina a stati sono:

- INIT da riga 1 a riga 5;
- IDLE da riga 1 a riga 15;
- CONNECTING TO HKMC fasi di connessione con l'home KMC da riga 21 a riga 34;
- TLS HANDSHAKE procedura effettuata nella connessioni TLS da riga 40 a 69;

- RESPONDING TO HKMC nella quale viene mandato il messaggio di NOTIF_SESSION_INIT per far connettere le entità. da riga 75 a 88. In riga 79 viene riportato l'header message (20 byte) e il body (3 byte) con i rispettivi campi in esadecimale, facendo riferimento alla tabella 5.3 è possibile notare che il 20-esimo byte è proprio 09 che corrisponde alla richiesta di inizio connessione mentre il 21, 22 e 23-esimo fanno riferimento al body cioè al messaggio NOTIF_SESSION_INIT 5.15, il byte 01 indica la versione dell'interfaccia, il byte 02 alla lista delle versioni supportate e il byte FF corrisponde al time-out dell'applicazione.
- WAITING FOR HKMC MSG in questo stato l'entità aspetta la risposta al messaggio di notifica da parte dell'entità HKMC di inizio sessione, appena la risposta è stata ricevuta la connessione ha inizio.

6.2 Simulazione del comando aggiungi chiave

La simulazione dei comandi è stata eseguita in chiaro, senza la parte di connessione perché quest'ultima verrà gestita dal cliente con il proprio server con i propri certificati.
Per questa parte del test si è voluto simulare un comando multiplo "add 3 keys" ovvero aggiungere contemporaneamente 3 chiavi. La prima parte del log file non è riportata perché riporta le stesse diciture della sezione precedente senza la presenza del handshake.

```
1  NOTIF SESS INIT --> ADD 3 KEYS --> NOTIF SESS END
2
3
4  SM : Executing -- ST_MACHINE_STATE_PARSING_HKMC_MSG -- state
5  KEY : KEYM_getLocalEtcsIdExp() -- OK -- Returning local ETCS_ID: 1
6  PARSER : Found valid Header
7  KEY : KEYM_setServerTrNumber() -- OK -- Server Transaction Number: 0
8  PARSER : PARSE_FromBufferToMessageBody() -- OK -- Complete and valid message (
       notification or command) received
9  SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND --
10 SM : -- ST_MACHINE_STATE_PARSING_HKMC_MSG -- state running exit status: 0
11 SM : Received -- ST_MACHINE_E_EXEC_REQ: 9 -- INTERNAL event
12 TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
13 TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
14 TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
15 TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
16 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
17 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
18 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
19
20
21 TMAIN_TEST : Executing run task...
22
23
24 SM : Executing -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND -- state
25 KEY : Executing NOTIFY_SESSION_INIT command
26 NM : Application timeout 50 sec has been requested by HKMC.
27 SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_RESPONDING_TO_HKMC --
28 SM : -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND -- state running exit status: 0
29 SM : Received -- ST_MACHINE_E_RESPOND_REQ: 10 -- INTERNAL event
30 TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
31 TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
32 TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
33 TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
34 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
35 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
36 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
```

```
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND


TMAIN_TEST : Executing run task...


SM : Executing -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state
SM : Nothing to write
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG --
SM : -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state running exit status: 0
SM : Received -- ST_MACHINE_E_WAIT_MSG: 14 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE


TMAIN_TEST : Executing run task...


SM : Executing -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG -- state
NM : Start Application Timer: 50 sec
NM : Running Application Timer
TCP : TCPCLIENT_readMessage() -- Start reading message
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
TCP : TCPCLIENT_genericRead() -- Read success. Read 187 bytes.
TCP : TCPCLIENT_readMessage() -- Read message duration: 0 milliseconds
KEY : KEYM_getLocalEtcsIdExp() -- OK -- Returning local ETCS_ID: 1
PARSER : Found valid Header
KEY : KEYM_setServerTrNumber() -- OK -- Server Transaction Number: 1
TCP : TCPCLIENT_execReadProcess() -- Read valid 20 bytes Header
TCP : TCPCLIENT_execReadProcess() -- Read valid 167 bytes Body
TCP : TCPCLIENT_readMessage() -- Read entire message
NM : NM_listenForMessage() -- Application timeout stopped
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_PARSING_HKMC_MSG --
SM : -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG -- state running exit status: 0
SM : Received -- ST_MACHINE_E_PARSE_REQ: 8 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE


TMAIN_TEST : Executing run task...


SM : Executing -- ST_MACHINE_STATE_PARSING_HKMC_MSG -- state
KEY : KEYM_getLocalEtcsIdExp() -- OK -- Returning local ETCS_ID: 1
```

```
PARSER : Found valid Header
KEY : KEYM_setServerTrNumber() -- OK -- Server Transaction Number: 1
PARSER : PARSE_CmdAddKeys() -- Starting command parsing
PARSER : PARSE_CmdAddKeys() -- Command Req. Number: 3
PARSER : PARSE_fromBufferToKStruct() -- Starting KStruct 0 data parse
PARSER : PARSE_fromBufferToKStruct() -- KStruct 0 data parse OK
PARSER : PARSE_fromBufferToKStruct() -- Starting KStruct 1 data parse
PARSER : PARSE_fromBufferToKStruct() -- KStruct 1 data parse OK
PARSER : PARSE_fromBufferToKStruct() -- Starting KStruct 2 data parse
PARSER : PARSE_fromBufferToKStruct() -- KStruct 2 data parse OK
PARSER : PARSE_FromBufferToMessageBody() -- OK -- Complete and valid message (
    notification or command) received
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND --
SM : -- ST_MACHINE_STATE_PARSING_HKMC_MSG -- state running exit status: 0
SM : Received -- ST_MACHINE_E_EXEC_REQ: 9 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE


TMAIN_TEST : Executing run task...


SM : Executing -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND -- state
KEY : Executing CMD_ADD_KEYS command
KEY : Executing Command Add Keys
KEY : Key 0 already installed
KEY : Key 1 already installed
KEY : Key 2 already installed
KEY : Printing Key Storage content...
KEY_STORAGE : Storage.NumberOfKeyRelation = 6
KEY_STORAGE : Storage.NumberOfKeys = 3
KEY_STORAGE : Printing KEY_0 Info...
======================KEY_0======================
KEY_INFO : KeyLength = 24
KEY_INFO : KeyId.EtcsExp.EtcsIdType = 1
KEY_INFO : KeyId.EtcsExp.EtcsId = 17 34 51
KEY_INFO : KeyId.SNum = 65244
KEY_INFO : EtcsIdExp.EtcsIdType = 1
KEY_INFO : EtcsIdExp.EtcsId = 17 34 51
KEY_INFO : Kmac = qwertyuiopasdfghjklzxcvb
KEY_INFO : PeerNum = 1
KEY_INFO : EtcsIdExpList.Peer_0
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 10
KEY_INFO : KeyValidPeriod = 20 33 3 21 24 37 3 21
=================================================
KEY_STORAGE : Printing KEY_1 Info...
======================KEY_1======================
KEY_INFO : KeyLength = 24
KEY_INFO : KeyId.EtcsExp.EtcsIdType = 1
KEY_INFO : KeyId.EtcsExp.EtcsId = 17 34 51
KEY_INFO : KeyId.SNum = 65245
```

```
KEY_INFO : EtcsIdExp.EtcsIdType = 1
KEY_INFO : EtcsIdExp.EtcsId = 17 34 51
KEY_INFO : Kmac = nmqwertyuiopasdfghjklzxc
KEY_INFO : PeerNum = 3
KEY_INFO : EtcsIdExpList.Peer_0
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 10
KEY_INFO : EtcsIdExpList.Peer_1
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 11
KEY_INFO : EtcsIdExpList.Peer_2
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 12
KEY_INFO : KeyValidPeriod = 20 33 3 21 24 37 3 21
==============================================
KEY_STORAGE : Printing KEY_2 Info...
=====================KEY_2=====================
KEY_INFO : KeyLength = 24
KEY_INFO : KeyId.EtcsExp.EtcsIdType = 1
KEY_INFO : KeyId.EtcsExp.EtcsId = 17 34 51
KEY_INFO : KeyId.SNum = 65246
KEY_INFO : EtcsIdExp.EtcsIdType = 1
KEY_INFO : EtcsIdExp.EtcsId = 17 34 51
KEY_INFO : Kmac = vbnmqwertyuiopasdfghjklz
KEY_INFO : PeerNum = 2
KEY_INFO : EtcsIdExpList.Peer_0
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 12
KEY_INFO : EtcsIdExpList.Peer_1
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 10
KEY_INFO : KeyValidPeriod = 20 33 3 21 24 37 3 21
==============================================
KEY_STORAGE : End of storage
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_RESPONDING_TO_HKMC --
SM : -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND -- state running exit status: 0
SM : Received -- ST_MACHINE_E_RESPOND_REQ: 10 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_ADD_KEYS_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_CMD_KEY_ALREADY_INSTALL


TMAIN_TEST : Executing run task...


SM : Executing -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state
PARSER : Parsing NOTIF_RESPONSE
SM : Sending Response of -- 26 bytes --
 00  00  00  1A  02  02  44  55  66  01  11  22  33  00  00  00  01  00  01  0B  00
    00  03  03  03  03
```

```
221 NM : NM_writeData() -- WARNING -- Sending data to UNSECURE write function
222 TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
223 SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG --
224 SM : -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state running exit status: 0
225 SM : Received -- ST_MACHINE_E_WAIT_MSG: 14 -- INTERNAL event
226 TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
227 TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
228 TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
229 TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
230 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
231 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
232 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
233 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
234 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
235 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
236 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
237 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
238 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
239 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_ADD_KEYS_COMMAND
240 TMAIN_TEST : EVENT : KM_EVENT_CMD_KEY_ALREADY_INSTALL
241 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
```

Come è possibile notare dal log file da riga 4 a 91 viene inviato il comando di NOTIF_SESSION_INIT che per mette la connessione tra le 2 entità, il comando viene scomposto ed analizzato nelle sue parti per verificarne la corretta formattazione(da riga 9 a 18) ed eseguito (da riga 24 a 38), dopo aver eseguito il comando l'HKCM aspetta la risposta al messaggio NOTIF_SESSION_INIT.
Dopo che la connessione è avvenuta l'HKMC invia il comando di "aggiungi 3 chiavi" e il comando viene nuovamente scomposto e analizzato (da riga 97 a 124) in questo caso vengono anche analizzata la struttura delle chiavi affinché sia conforme con le norma del subset (da riga 103 a 109). Il successivo stato della macchina e quello di esecuzione del comando dove viene verificato se le chiavi sono già installate, se le chiavi risultano presenti nel DB (come in questo caso) viene visualizzato un messaggio che indica che la chiave e già presente, viene anche riportata la struttura della chiave come visto nella tabella 5.6 ogni campo è espresso in esadecimale tranne il campo Kmac che è espresso in formato stringa per una più facile identificazione.
Dopo l'esecuzione del comando avviene la fase in cui l'HKMAC risponde al comando inviato, per una corretta interpretazione si è deciso di far visualizzare il messaggio di risposta e la sua grandezza (riga 219) in questo caso il messaggio è NOTIF_RESPONSE. Analizzando i vari campi del messaggio, i primi 20 byte sono l'header mentre i restanti 6 sono il body, facendo riferimento alla tabella 5.3 il 20esimo byte "0B" corrisponde al comando NOTIF_RESPONSE mentre per i byte del body si deve fare riferimento alla tabella 5.16 che riporta la struttura del messaggio di risposta il 21esimo "00" byte corrisponde a una richiesta andata a buon fine il 22esimo e il 23esimo corrispondono al campo REQ-NUM che indica il risultato della richiesta fatta, vedi tabella 5.17 "00 03" corrisponde alla voce "Richiesta di installare chiavi già presenti nel DB", gli ultimi 3 byte corrispondono al vettore che indica il risultato della richiesta per ogni chiave.

Successivamente si aspetta la risposta dal HKMC e abbiamo inviato il comando di NOTIF_SESSION_END si fa il parsing (scomporre e analizzare) del comando lo si esegue e in riga 378 è possibile notare che lo stato della macchina è passato a STATE_DISCONNECT ovvero la fine della connessione. Successivamente per controllare la corretta installazione delle chiavi abbiamo deciso di verificare il numero di chiavi nel DB senza la necessità di eseguire la fase di connessione.

```
1 SM : Executing -- ST_MACHINE_STATE_DISCONNECT -- state
```

6.3 Simulazione del comando cancella tutto

```
NOTIF SESS INIT --> DELETE ALL --> ADD 3 KEYS --> NOTIF END UPDATE



Start!



KM : KM_Init() -- Starting State Machine Initialization/Reset
KM : KM_Init() -- State Machine Initialization/Reset OK --
KEY : KEYM_init() -- EXECUTING KEY MANAGER INIT
KM : KM_Init() -- Starting Network Manager Initialization/Reset
NM : NM_disconnect() -- Clear text connection, no need to free SSL CTX
NM : NM_disconnect() -- Clear text connection, no need to free SSL session
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: CLOSED
TCP : TCPCLIENT_genericRead() -- TCP ports already closed. Nothing to do.
KM : KM_Init() -- Network Manager Initialization/Reset OK --
KM : KM_Init() -- Starting Security Manager Initialization/Reset
KM : KM_Init() -- Security Manager Initialization/Reset OK --
TEST : Start initKM()
KM : SUCCESS loading ETCSID in KM_setETCSID
TEST : Start KM_setInfoHomeKMC ()
KM : KM_setNetworkContext() -- OK --
TEST : Start KM_setSecurityContextDN ()
KM : KM_setSecurityContextDN() -- DNs load OK
TEST : Start KM_setSecurityContextPSK ()
SEC : SEC_setPskCredential() -- TLS method init OK --
SEC : SEC_setPskCredential() -- SSL context struct init OK --
SEC : SEC_setContextPSK() -- Set PSK credentials OK
KM : KM_setSecurityContextPSK() -- Set PSK context OK --
TEST : Start KM_setSecurityContextPKI ()
SEC : SEC_setPkiCredential() -- TLS method init OK --
SEC : SEC_setPkiCredential() -- SSL context struct init OK --
SEC : SEC_setPkiCredential() -- load RSA CA certificate PATH: ./support_TLS/x509/ca-
    chain.pem --
SEC : SEC_setPkiCredential() -- load CA certificate OK --
SEC : SEC_loadX509CertFromLocalPath() -- BIO new OK --
SEC : SEC_loadX509CertFromLocalPath() -- BIO read Cert file OK --
SEC : SEC_loadX509CertFromLocalPath() -- Create X509 Cert structure OK --
SEC : SEC_loadCaLocalCert() -- Load the local CA certificate from path OK
SEC : SEC_validateLocalCaCertificate() -- Load CA certificate OK
SEC : SEC_validateCaLocalCertificate() -- Check CA certificate expiration status
SEC : SEC_checkCertRevokeTime() -- Starting local Cert revoke time
SEC : SEC_checkCertRevokeTime() -- Get current time OK -- Time_now : 1665585172
    seconds
SEC : SEC_checkCertRevokeTime() -- Certificate revoke time : 1691562432 seconds
SEC : SEC_checkCertRevokeTime() -- Certificate renew time : 1691526432 seconds
SEC : SEC_checkCertRevokeTime() -- Cert expiration time check: OK
SEC : SEC_validateLocalCaCertificate() -- Check CA certificate expiration status OK
SEC : SEC_verifyCertificateChainAgainstCA() -- CA certificate valid --
SEC : SEC_loadClientKeyAndCert() -- load client certificate PATH: ./support_TLS/x509/
    client1.pem --
SEC : SEC_loadClientKeyAndCert() -- load client certificate OK --
SEC : SEC_loadClientKeyAndCert() -- load local private-key PATH: ./support_TLS/x509/
    client1-key.pem --
SEC : SEC_loadClientKeyAndCert() -- load local private-key OK --
SEC : SEC_loadClientKeyAndCert() -- local certificate and private-key matches --
SEC : SEC_setPkiCredential() -- Load local keys and certificate OK --
SEC : SEC_loadX509CertFromLocalPath() -- BIO new OK --
```

```
SEC : SEC_loadX509CertFromLocalPath() -- BIO read Cert file OK --
SEC : SEC_loadX509CertFromLocalPath() -- Create X509 Cert structure OK --
SEC : SEC_verifyLocalCertificateChain() -- Create X509 store object OK --
SEC : SEC_verifyLocalCertificateChain() -- Create X509 store context OK --
SEC : SEC_verifyLocalCertificateChain() -- Load CA certificate into X509 struct OK --
SEC : SEC_verifyLocalCertificateChain() -- X509 store context init OK --
SEC : SEC_verifyLocalCertificateChain() -- Certificate is valid
SEC : SEC_setPkiCredential() -- Verify local certificate against CA OK --
SEC : SEC_setContextPKI() -- OK
KM : KM_setSecurityContextPKI -- Set PKI context OK --
STORAGE : STORAGE_openFile() -- File open OK: ./keys-storage.bin
STORAGE : STORAGE_readData() -- Reading file: ./keys-storage.bin
STORAGE : STORAGE_readData() -- Read file OK: ./keys-storage.bin
TEST : testFillKeyStorageFromFile() -- OK -- Key storage load from file OK
KEY : Load page : 0
KEY : Key loaded : 3
TEST : debugInit() -- key storage page 0 settingKeyRelation OK
TEST : debugInit() -- Key storage checksum:
00 00 00 00 2A 8F A6 B3 36 62 6A 35 4F 69 2F ED DC 54 5D 4B
SM : Received -- KM_ACTION_SET_CONNECTION_REQUEST: 2 -- EXTERNAL action
SM : KM_ACTION_SET_CONNECTION_REQUEST propagated
TMAIN_TEST : Executing run task...
```

```
SM : Executing -- ST_MACHINE_STATE_INIT -- state
SM : SM_init_State() done.
SM : -- ST_MACHINE_STATE_INIT -- state running exit status: 0
SM : Received -- ST_MACHINE_E_INIT_DONE: 2 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
```

```
TMAIN_TEST : Executing run task...
```

```
SM : Executing -- ST_MACHINE_STATE_IDLE -- state
SM : -- ST_MACHINE_STATE_IDLE -- state running exit status: 0
SM : Received -- ST_MACHINE_E_CONNECT_REQ: 5 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
```

```
TMAIN_TEST : Executing run task...
```

```
SM : Executing -- ST_MACHINE_STATE_CONNECTING_TO_HKMC -- state
NM : NM_openConnection() -- Timer enabled
NM : NM_openConnection() -- OK -- Try to open TCP port
TCP : TCPCLIENT_openTcpPort() -- Socket creation OK --
TCP : TCPCLIENT_openTcpPort() -- set Server IP: 127.0.0.1 OK --
TCP : TCPCLIENT_openTcpPort() -- establish TCP/IP connection to the SSL Server OK --
NM : NM_openConnection() -- TCP port open OK
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
```

```
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_TLS_HANDSHAKE --
SM : -- ST_MACHINE_STATE_CONNECTING_TO_HKMC -- state running exit status: 0
SM : Received -- ST_MACHINE_E_TLS: 15 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE



TMAIN_TEST : Executing run task...



SM : Executing -- ST_MACHINE_STATE_TLS_HANDSHAKE -- state
NM : NM_startSecureHandshake() -- Starting TLS handshake --
NM : NM_tryHandshake() -- Starting TLS handshake
NM : NM_tryHandshake() -- NO SECURITY PROTOCOL SELECTED. NOTHING TO DO
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_RESPONDING_TO_HKMC --
SM : -- ST_MACHINE_STATE_TLS_HANDSHAKE -- state running exit status: 0
SM : Received -- ST_MACHINE_E_RESPOND_REQ: 10 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE



TMAIN_TEST : Executing run task...



SM : Executing -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state
KEY : Generate NOTIF_SESSION_INIT message
PARSER : Parsing NOTIF_SESSION_INIT
SM : Sending Response of -- 23 bytes --
00 00 00 17 02 02 44 55 66 01 11 22 33 00 00 00 00 00 00 09 01 02 FF
NM : NM_writeData() -- WARNING -- Sending data to UNSECURE write function
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG --
SM : -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state running exit status: 0
SM : Received -- ST_MACHINE_E_WAIT_MSG: 14 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE



TMAIN_TEST : Executing run task...



SM : Executing -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG -- state
NM : Start Application Timer: 15 sec
NM : Running Application Timer
```

```
TCP : TCPCLIENT_readMessage() -- Start reading message
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
TCP : TCPCLIENT_genericRead() -- Read success. Read 23 bytes.
TCP : TCPCLIENT_readMessage() -- Read message duration: 0 milliseconds
KEY : KEYM_getLocalEtcsIdExp() -- OK -- Returning local ETCS_ID: 1
PARSER : Found valid Header
KEY : KEYM_setServerTrNumber() -- OK -- Server Transaction Number: 0
TCP : TCPCLIENT_execReadProcess() -- Read valid 20 bytes Header
TCP : TCPCLIENT_execReadProcess() -- Read valid 3 bytes Body
TCP : TCPCLIENT_readMessage() -- Read entire message
NM : NM_listenForMessage() -- Application timeout stopped
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_PARSING_HKMC_MSG --
SM : -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG -- state running exit status: 0
SM : Received -- ST_MACHINE_E_PARSE_REQ: 8 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE



TMAIN_TEST : Executing run task...




SM : Executing -- ST_MACHINE_STATE_PARSING_HKMC_MSG -- state
KEY : KEYM_getLocalEtcsIdExp() -- OK -- Returning local ETCS_ID: 1
PARSER : Found valid Header
KEY : KEYM_setServerTrNumber() -- OK -- Server Transaction Number: 0
PARSER : PARSE_FromBufferToMessageBody() -- OK -- Complete and valid message (
    notification or command) received
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND --
SM : -- ST_MACHINE_STATE_PARSING_HKMC_MSG -- state running exit status: 0
SM : Received -- ST_MACHINE_E_EXEC_REQ: 9 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE



TMAIN_TEST : Executing run task...




SM : Executing -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND -- state
KEY : Executing NOTIFY_SESSION_INIT command
NM : Application timeout 50 sec has been requested by HKMC.
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_RESPONDING_TO_HKMC --
SM : -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND -- state running exit status: 0
SM : Received -- ST_MACHINE_E_RESPOND_REQ: 10 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
```

```
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND



TMAIN_TEST : Executing run task...




SM : Executing -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state
SM : Nothing to write
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG --
SM : -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state running exit status: 0
SM : Received -- ST_MACHINE_E_WAIT_MSG: 14 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE



TMAIN_TEST : Executing run task...




SM : Executing -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG -- state
NM : Start Application Timer: 50 sec
NM : Running Application Timer
TCP : TCPCLIENT_readMessage() -- Start reading message
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
TCP : TCPCLIENT_genericRead() -- Read success. Read 20 bytes.
TCP : TCPCLIENT_readMessage() -- Read message duration: 0 milliseconds
KEY : KEYM_getLocalEtcsIdExp() -- OK -- Returning local ETCS_ID: 1
PARSER : Found valid Header
KEY : KEYM_setServerTrNumber() -- OK -- Server Transaction Number: 1
TCP : TCPCLIENT_execReadProcess() -- Read valid 20 bytes Header
TCP : TCPCLIENT_execReadProcess() -- Read valid 0 bytes Body
TCP : TCPCLIENT_readMessage() -- Read entire message
NM : NM_listenForMessage() -- Application timeout stopped
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_PARSING_HKMC_MSG --
SM : -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG -- state running exit status: 0
SM : Received -- ST_MACHINE_E_PARSE_REQ: 8 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
```

```
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE



TMAIN_TEST : Executing run task...



SM : Executing -- ST_MACHINE_STATE_PARSING_HKMC_MSG -- state
KEY : KEYM_getLocalEtcsIdExp() -- OK -- Returning local ETCS_ID: 1
PARSER : Found valid Header
KEY : KEYM_setServerTrNumber() -- OK -- Server Transaction Number: 1
PARSER : PARSE_FromBufferToMessageBody() -- OK -- Complete and valid message (
    notification or command) received
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND --
SM : -- ST_MACHINE_STATE_PARSING_HKMC_MSG -- state running exit status: 0
SM : Received -- ST_MACHINE_E_EXEC_REQ: 9 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE



TMAIN_TEST : Executing run task...



SM : Executing -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND -- state
KEY : Executing CMD_DELETE_ALL_KEYS command
KEY : Executing Command Delete All Keys
KEY : All keys in the Storage have been successfully deleted
KEY : Key Storage updated
KEY : Printing Key Storage content...
KEY_STORAGE : No keys in the storage
KEY_STORAGE : End of storage
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_RESPONDING_TO_HKMC --
SM : -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND -- state running exit status: 0
SM : Received -- ST_MACHINE_E_RESPOND_REQ: 10 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
```

```
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_DELETE_ALL_KEYS_COMMAND



TMAIN_TEST : Executing run task...



SM : Executing -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state
PARSER : Parsing NOTIF_RESPONSE
SM : Sending Response of -- 23 bytes --
00 00 00 17 02 02 44 55 66 01 11 22 33 00 00 00 01 00 01 0B 00 00 00
NM : NM_writeData() -- WARNING -- Sending data to UNSECURE write function
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG --
SM : -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state running exit status: 0
SM : Received -- ST_MACHINE_E_WAIT_MSG: 14 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_DELETE_ALL_KEYS_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE



TMAIN_TEST : Executing run task...



SM : Executing -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG -- state
NM : Start Application Timer: 50 sec
NM : Running Application Timer
TCP : TCPCLIENT_readMessage() -- Start reading message
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
TCP : TCPCLIENT_genericRead() -- Read success. Read 187 bytes.
TCP : TCPCLIENT_readMessage() -- Read message duration: 0 milliseconds
KEY : KEYM_getLocalEtcsIdExp() -- OK -- Returning local ETCS_ID: 1
PARSER : Found valid Header
KEY : KEYM_setServerTrNumber() -- OK -- Server Transaction Number: 2
TCP : TCPCLIENT_execReadProcess() -- Read valid 20 bytes Header
TCP : TCPCLIENT_execReadProcess() -- Read valid 167 bytes Body
```

```
426 TCP : TCPCLIENT_readMessage() -- Read entire message
427 NM : NM_listenForMessage() -- Application timeout stopped
428 SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_PARSING_HKMC_MSG --
429 SM : -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG -- state running exit status: 0
430 SM : Received -- ST_MACHINE_E_PARSE_REQ: 8 -- INTERNAL event
431 TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
432 TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
433 TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
434 TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
435 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
436 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
437 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
438 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
439 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
440 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
441 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
442 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
443 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
444 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_DELETE_ALL_KEYS_COMMAND
445 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
446 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
447
448
449
450
451 TMAIN_TEST : Executing run task...
452
453
454
455
456 SM : Executing -- ST_MACHINE_STATE_PARSING_HKMC_MSG -- state
457 KEY : KEYM_getLocalEtcsIdExp() -- OK -- Returning local ETCS_ID: 1
458 PARSER : Found valid Header
459 KEY : KEYM_setServerTrNumber() -- OK -- Server Transaction Number: 2
460 PARSER : PARSE_CmdAddKeys() -- Starting command parsing
461 PARSER : PARSE_CmdAddKeys() -- Command Req. Number: 3
462 PARSER : PARSE_fromBufferToKStruct() -- Starting KStruct 0 data parse
463 PARSER : PARSE_fromBufferToKStruct() -- KStruct 0 data parse OK
464 PARSER : PARSE_fromBufferToKStruct() -- Starting KStruct 1 data parse
465 PARSER : PARSE_fromBufferToKStruct() -- KStruct 1 data parse OK
466 PARSER : PARSE_fromBufferToKStruct() -- Starting KStruct 2 data parse
467 PARSER : PARSE_fromBufferToKStruct() -- KStruct 2 data parse OK
468 PARSER : PARSE_FromBufferToMessageBody() -- OK -- Complete and valid message (
        notification or command) received
469 SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND --
470 SM : -- ST_MACHINE_STATE_PARSING_HKMC_MSG -- state running exit status: 0
471 SM : Received -- ST_MACHINE_E_EXEC_REQ: 9 -- INTERNAL event
472 TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
473 TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
474 TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
475 TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
476 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
477 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
478 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
479 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
480 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
481 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
482 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
483 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
484 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
485 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_DELETE_ALL_KEYS_COMMAND
486 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
```

```
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE



TMAIN_TEST : Executing run task...



SM : Executing -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND -- state
KEY : Executing CMD_ADD_KEYS command
KEY : Executing Command Add Keys
KEY : Key 0 added successfully. Valid period: 1421031518250315
KEY : Key Storage updated
KEY : Key 1 added successfully. Valid period: 1421031518250315
KEY : Key Storage updated
KEY : Key 2 added successfully. Valid period: 1421031518250315
KEY : Key Storage updated
KEY : Printing Key Storage content...
KEY_STORAGE : Storage.NumberOfKeyRelation = 6
KEY_STORAGE : Storage.NumberOfKeys = 3
KEY_STORAGE : Printing KEY_0 Info...
======================KEY_0======================
KEY_INFO : KeyLength = 24
KEY_INFO : KeyId.EtcsExp.EtcsIdType = 1
KEY_INFO : KeyId.EtcsExp.EtcsId = 17 34 51
KEY_INFO : KeyId.SNum = 65244
KEY_INFO : EtcsIdExp.EtcsIdType = 1
KEY_INFO : EtcsIdExp.EtcsId = 17 34 51
KEY_INFO : Kmac = qwertyuiopasdfghjklzxcvb
KEY_INFO : PeerNum = 1
KEY_INFO : EtcsIdExpList.Peer_0
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 10
KEY_INFO : KeyValidPeriod = 20 33 3 21 24 37 3 21
=================================================
KEY_STORAGE : Printing KEY_1 Info...
======================KEY_1======================
KEY_INFO : KeyLength = 24
KEY_INFO : KeyId.EtcsExp.EtcsIdType = 1
KEY_INFO : KeyId.EtcsExp.EtcsId = 17 34 51
KEY_INFO : KeyId.SNum = 65245
KEY_INFO : EtcsIdExp.EtcsIdType = 1
KEY_INFO : EtcsIdExp.EtcsId = 17 34 51
KEY_INFO : Kmac = nmqwertyuiopasdfghjklzxc
KEY_INFO : PeerNum = 3
KEY_INFO : EtcsIdExpList.Peer_0
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 10
KEY_INFO : EtcsIdExpList.Peer_1
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 11
KEY_INFO : EtcsIdExpList.Peer_2
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 12
KEY_INFO : KeyValidPeriod = 20 33 3 21 24 37 3 21
=================================================
KEY_STORAGE : Printing KEY_2 Info...
======================KEY_2======================
KEY_INFO : KeyLength = 24
```

```
KEY_INFO : KeyId.EtcsExp.EtcsIdType = 1
KEY_INFO : KeyId.EtcsExp.EtcsId = 17 34 51
KEY_INFO : KeyId.SNum = 65246
KEY_INFO : EtcsIdExp.EtcsIdType = 1
KEY_INFO : EtcsIdExp.EtcsId = 17 34 51
KEY_INFO : Kmac = vbnmqwertyuiopasdfghjklz
KEY_INFO : PeerNum = 2
KEY_INFO : EtcsIdExpList.Peer_0
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 12
KEY_INFO : EtcsIdExpList.Peer_1
KEY_INFO : EtcsIdExpList.EtcsIdType = 1
KEY_INFO : EtcsIdExpList.EtcsId = 0 0 10
KEY_INFO : KeyValidPeriod = 20 33 3 21 24 37 3 21
==========================================
KEY_STORAGE : End of storage
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_RESPONDING_TO_HKMC --
SM : -- ST_MACHINE_STATE_EXECUTING_HKMC_COMMAND -- state running exit status: 0
SM : Received -- ST_MACHINE_E_RESPOND_REQ: 10 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_DELETE_ALL_KEYS_COMMAND
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_ADD_KEYS_COMMAND



TMAIN_TEST : Executing run task...




SM : Executing -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state
PARSER : Parsing NOTIF_RESPONSE
SM : Sending Response of -- 26 bytes --
00 00 00 1A 02 02 44 55 66 01 11 22 33 00 00 00 02 00 02 0B 00 00 03 00 00 00
NM : NM_writeData() -- WARNING -- Sending data to UNSECURE write function
TCP : TCPCLIENT_checkTcpPortStatus() -- Socket status: OPEN
SM : Operation result: OK ; Next state: -- ST_MACHINE_STATE_WAITING_FOR_HKMC_MSG --
SM : -- ST_MACHINE_STATE_RESPONDING_TO_HKMC -- state running exit status: 0
SM : Received -- ST_MACHINE_E_WAIT_MSG: 14 -- INTERNAL event
TMAIN_TEST : EVENT : KM_EVENT_INIT_STATE
TMAIN_TEST : EVENT : KM_EVENT_IDLE_STATE
TMAIN_TEST : EVENT : KM_EVENT_CONNECTING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_TLS_HANDSHAKE_STATE
TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
```

```
611 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
612 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
613 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_NOTIF_SESSION_INIT_COMMAND
614 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
615 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
616 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
617 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
618 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_DELETE_ALL_KEYS_COMMAND
619 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
620 TMAIN_TEST : EVENT : KM_EVENT_WAITING_FOR_HKMC_MSG_STATE
621 TMAIN_TEST : EVENT : KM_EVENT_PARSING_HKMC_MSG_STATE
622 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_COMMAND_STATE
623 TMAIN_TEST : EVENT : KM_EVENT_EXECUTING_HKMC_ADD_KEYS_COMMAND
624 TMAIN_TEST : EVENT : KM_EVENT_RESPONDING_TO_HKMC_STATE
```

In questa simulazione si sono voluti testare 2 comandi consecutivi, il primo "DELETE_ALL_KEYS" il quale cancella tutte le chiavi presenti nel DB, mentre il secondo comando è quello riportato nella sezione precedente ma questa volta il messaggio di risposta sarà diverso poiché le chiavi non sono presenti all'interno del DB.
La simulazione ha le stesse fasi della precedente, ovvero :

- Inizializzazione del KM;
- IDLE;
- Connessione con l'HKMC;
- Attesa risposta dal HKCM;

Una volta avviata la connessione il viene inviato il comando richiesto, "DELETE_ALL_KEYS", viene scomposto ed analizzato per assicurarne la correttezza ed eseguito, il comando è visibile alla riga 348. Il messaggio di risposta (riga 384) in questo caso è di dimensioni 23 byte, il 20esimo byte "OB" indica l'invio di un messaggio di notifica, vedi tabella 5.3, mentre i successivi 3 corrispondono alla corretta esecuzione del comando come visibile in tabella 5.16. Finché non viene inviato il comando di fine sessione l'HKMC aspetta il prossimo messaggio, che sarà aggiungi 3 chiavi, una volta che il messaggio è stato inviato l'HKMC esegue il comando, visibile in riga 499. Le righe 501, 503 e 505 indicano che le chiavi sono state installate correttamente mentre le righe 502,504 e 506 mostrano che questa volte le chiavi non era già presenti quindi la cancellazione delle chiavi è andata a buon fine. Il messaggio di risposta, riga 599, è diverso dal caso precedenti infatti adesso abbiamo il 20esimo byte "0B" che indica l'invio di un messaggio di notifica, mentre i byte 21, 22 e 23 , "00 00 03" indica che ci sono state 3 richieste. Gli ultimi 3 byte "00 00 00", i quali riportano l'esito per ogni chiave, indicano che l'installazione delle chiavi è andata a buon fine, nel caso precedente gli ultimi 3 byte erano "03 03 03" che stavano ad indicare che le chiavi erano già presenti nel DB. Successivamente viene mandato il messaggio di fine aggiornamento e le entità si disconnettono.

CAPITOLO 7

Conclusioni

Come è possibile notare nella sezione simulazione i componendi rispondono bene a tutti i comandi e con i relativi messaggi di notifica. E' possibile utilizzare l'applicazione per lo scambio di chiavi tra le entità on-bord e track side. Questa tecnologia oltre che nell'ambito railway può essere usata anche per connessione sicure tramite scambio di chiavi cifrate di diversi mezzi, come per le linee areree per permettere di comunicare in maniera sicura e cifrata, in modo che nessuno, se non chi autenticato, possa intromettersi nella trasmissione. E' inoltre possibile anche applicabile l'applicazione anche ai centri di gestione delle chiavi che svolgono attività per la generazione e il controllo di certificati per garantire l'autenticità delle entità comunicanti nelle comunicazioni TLS. L'applicazione può anche essere usata molto in generale per avere in qualsiasi contesto comunicazione sicure e cifrate.
.

Bibliografia

[UNSING] *On-line Key Management FFFIS, Subset 137 v1. 0. 0 (2015)*

[UNSING] *Off-line Key Management FFFIS, Subset 137 v1. 0. 0 (2015)*

[Railwaysignalling.eu] *The ERTMS/ETCS signalling system*

[UNISING] *EuroRadioFIS*

[https://www.rfc-editor.org/rfc/rfc1320] *[RFC 1320]*