UNIVERSITÀ DI PISA

Dipartimento di Informatica

Corso di Laurea in Informatica

Progettazione di meccanismi di protezione nelle comunicazioni intra-veicolari

Tutore interno:

Prof.ssa Anna Bernasconi

Tutori aziendali:

Dott. Gianpiero Costantino

Dott.ssa Ilaria Matteucci

Presentata da:

Carlo Leo

Sessione autunnale

Anno Accademico 2018/2019

Elenco delle figure

Indice

Capitolo 1

Introduzione

La maggior parte delle funzionalità dei veicoli moderni sono regolate attraverso *centraline elettroniche* -Eletronic Control Units (ECU)- le quali, per assolvere al loro compito, necessitano di interagire tra loro scambiandosi una mole considerevole di informazioni. Componenti del veicolo, come ad esempio airbag, freni, sterzo, etc., hanno la necessità di essere interconnessi e comunicare fra loro al fine di garantire il regolare funzionamento delle varie funzionalità di loro competenza, e di conseguenza un corretto funzionamento del veicolo stesso. Le centraline, quindi, formano all'interno del veicolo una vera e propria rete di cui esse stesse sono i nodi che comunicano fra di loro usando un protocollo stabilito: il protocollo **Controller Area Network**, in breve **CAN bus**, il quale risale al 1983 ed oggi è molto diffuso. Questo protocollo di comunicazione è standardizzato in ISO-11898: 2003 [14, 11] come un semplice protocollo basato su due linee di bus. Tuttavia, per ragioni storiche, il **CAN bus** non è pensato per essere sicuro. Infatti, è stato disegnato sotto l'assunzione che la comunicazione avvenga in un ambiente chiuso, ossia non accessibile dall'esterno, secondo il concetto di "*Security-by-Obscurity*". Nel campo della cybersecurity è noto che ipotesi troppo ottimistiche sono da considerare infrante quando si va a valutare l'istanza reale del problema. Non fa eccezione l'attuale panorama automobilistico, dove l'ipotesi sopra citata è da considerare violata per molti motivi. Tra i vari c'è ne sono due da evidenziare con maggiore enfasi poiché riguardano gli interessi dei passeggeri in modo diretto:

- il funzionamento di un'auto è strettamente relazionato alla salvaguardia (*safety*) dei passeggeri: potenziali attaccanti potrebbero danneggiare il veicolo per recare loro danni;

- le auto, con il progredire degli anni, diventano sempre più connesse con il mondo esterno, mantenendo sempre più dati sul conducente, come ad esempio informazioni inerenti allo stile di guida e episodi significativi, dati ambientali acquisiti mediante i vari sensori e telecamere, nonché ricevuti dagli smartphone dei passeggeri. Inoltre, laddove vengono utilizzate tecniche biometriche per autenticare il conducente, vengono trattati anche i dati sensibili. Tutti questi dati sono ovviamente interessanti, ad esempio, per motivi di marketing, per le assicurazioni e anche in vista dell'avvento della guida totalmente autonoma.

Alla luce di queste osservazioni, sta crescendo la consapevolezza della necessità di introdurre meccanismi, tecniche e soluzioni di sicurezza così da rendere sicura la rete intraveicolare formata dalle ECU. I problemi inerenti alla sicurezza sono una delle maggiori e più difficili sfide nel campo automobilistico, dati i vincoli temporali (tempi di reazione), computazionali e di spazio dettati sia dalla capacità computazionali delle ECU, sia dagli standard in materia di *safety*. Inoltre questi problemi riguardano anche l'integrità del veicolo stesso oltre che dei passeggeri. È chiaro che entrambi gli aspetti sono fondamentali nel disegno di un **protocollo di comunicazione**, nel quale devono essere rispettate le seguenti tre proprietà:

1. **Autenticazione**: un destinatario deve essere in grado di verificare se un messaggio è stato mandato da un mittente legittimo.

2. **Integrità**: un destinatario deve essere in grado di verificare se un messaggio è stato alterato durante la trasmissione.

3. **Confidenzialità**: garantisce che il contenuto di un messaggio non può essere noto a terze parti non autorizzate.

Con il passere degli anni e data l'eterogeneità delle soluzione proposte, è stato fondato un consorzio addetto a definire linee guida per la produzione di software nel modo automobi-

listico: **AUTOSAR**. Ad esempio, in termini di sicurezza, al fine di garantire le prime due proprietà sopraelencate in materia di comunicazione intra-veicolare, AUTOSAR consiglia l'utilizzo del **Message Authentication Code (MAC)**, mentre sulla terza proprietà non sono state proposte linee guide per allineare le varie soluzioni proposte, in quanto tutte sono accomunate dall'utilizzo della **crittografia** per garantirla. Nel corso degli ultimi anni, sono state proposte diverse soluzioni nella letteratura relativa alla sicurezza in ambito automobilistico, e.g., [8, 9, 15]. Un esempio è il protocollo **TOUCAN** [23, 24] definito per la comunicazione intra-veicolare in cui sono tate seguite le linee guida di AUTOSAR. In più, come sua peculiarità TOUCAN cifra anche il MAC. Per la realizzazione di entrambe le procedure vi è la necessità di avere un segreto condiviso da tutti gli attori del protocollo di comunicazione: **chiave di sessione**. Ciò è necessario perché, data l'efficienza in tempo e in spazio della crittografia simmetrica rispetto a quella asimmetrica e dati i vincoli presenti nel modo automobilistico, l'utilizzo di cifrari simmetrici risulta essere più vantaggioso e meno problematico di un approccio che utilizza il suo complementare. Infatti tutti i protocolli di comunicazione proposti percorrono la via della crittografia simmetrica. Questa soluzione introduce un nuovo problema, cioè condividere la chiave di sessione tra i vari partecipanti. Nei protocolli proposti che implementano una comunicazione sicura tra le varie ECU non c'è una fase in cui viene effettuata la condivisione della chiave, rendendola così una parte preliminare alla comunicazione vera è propria. Il problema della **condivisione della chiave di sessione** si pone alle base della realizzazione di una comunicazione sicura. Il tirocinio sostenuto, dal titolo *"Progettazione di meccanismi di protezione nelle comunicazioni intra-veicolari"* presso il gruppo di ricerca **Trustworthy and Secure Future Internet (TSFI)** dell'**Istituto di Informatica e Telematica (IIT) del Consiglio Nazionale delle Ricerche (CNR)**, ha previsto la comprensione degli attuali meccanismi di comunicazione nel mondo automobilistico e la progettazione di un protocollo per lo scambio di chiavi di sessione così da aumentare la sicurezza delle comunicazioni intra-veicolari. Quindi, il lavoro svolto è stato incentrato sulla condivisione di una chiave di sessione così da estendere il protocollo TOUCAN, con la condivisione della chiave di sessione che poi tale protocollo userà per la comunicazione sicura fra le ECU. Inoltre, il tirocinio ha previsto lo studio in via sperimentale del cifrario

RSA nel contesto automobilistico, alla luce dei vincoli e dei requisiti che questo dominio applicativo impone.

Capitolo 2

CAN: Controller Area Network

Il *Controller Area Network* (CAN) bus è stato introdotto negli anni ottanta dalla **Robert Bosch GmbH**, come un sistema di messaggistica broadcast con una velocità massima di segnale pari a 1 megabit al secondo. Al contrario di una rete tradizionale come ad esempio USB o ethernet, CAN non invia gradi blocchi di dati da un nodo A a un nodo B. Inoltre, in una rete CAN non vi è una comunicazione punto a punto ma molti messaggi brevi vengono trasmessi a tutta la rete il che garantisce la coerenza dei dati in ogni singolo nodo del sistema.

2.1 Panoramica

Il CAN è un bus di comunicazione seriale sviluppato originariamente per l'industria automobilistica per sostituire il complesso cablaggio con un bus a due fili. Le specifiche richiedono un'elevata immunità alle interferenze elettriche e la capacità di auto diagnostica e riparazione per gli errori sui dati. Queste caratteristiche hanno portato alla popolarità di CAN in una varietà di settori tra cui automazione degli edifici, medicina e produzione. Il protocollo di comunicazione CAN, standardizzato con la ISO-11898: 2003 [11], descrive come le informazioni devono essere passate tra i dispositivi di una rete. Inoltre, è conforme al modello Open Systems Interconnection (OSI) definito in termini di livelli. La comunicazione effettiva tra i dispositivi collegati dal mezzo fisico è definita dallo strato fisico del modello. L'architettura ISO 11898 definisce i due livelli più bassi dei sette livelli

del modello ISO/OSI: livello di collegamento dati e livello fisico.

Il protocollo CAN è un protocollo **carrier-sense, multiple-access con collision detection e arbitraggio basato sulla priorità del messaggio (CSMA/CD+AMP)**. CSMA significa che ogni nodo su un bus deve attendere un periodo di tempo prestabilito prima di tentare di inviare un messaggio. CD+AMP significa che le collisioni vengono risolte attraverso un arbitrato *bit-wise*, basato sulla priorità predefinita di un messaggio data dal suo campo identificativo. Il CAN trasmette dati secondo un modello basato su bit "dominanti"e "recessivi", in cui i bit dominanti sono gli 0 logici e i bit recessivi sono gli 1 logici. Se un nodo trasmette un bit dominante e un altro un bit recessivo, allora il bit "dominante" "vince"fra i due (realizzando di fatto un AND logico). Ad oggi vi sono tre versioni del protocollo che saranno descritte più in dettaglio di seguito: **Standard CAN**, **Extended CAN** e **CAN FD**.

2.2 Standard CAN

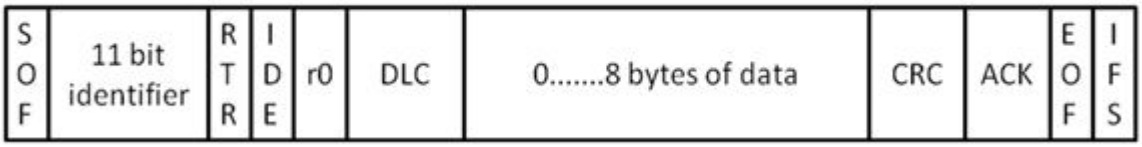

Figura 2.1: Frame Standard CAN

In Figura 2.1 è rappresentata la struttura logica di un frame, dove:

- **SOF**: rappresenta un singolo bit che segna l'inizio di un messaggio e viene utilizzato per sincronizzare i nodi su un bus dopo essere stati inattivi.
- **Identifier**: composto da 11 bit che stabiliscono la priorità del messaggio: più basso è il valore binario più alta è la priorità. Un vincolo imposto al campo dell'identificatore è che i primi 7 bit non possono essere tutti recessivi.
- **RTR**: campo di un solo bit. Assume il valore dominante quando l'informazione è richiesta da un altro nodo. Tutti nodi ricevono la richiesta, ma il campo *identifier*

determina il nodo specifico. La risposta è anch'essa ricevuta da tutti i nodi e quindi può essere utilizzata da un qualsiasi nodo interessato.

- **IDE**: campo di un bit che assume il valore predominante e indica che si sta trasmettendo un frame standard.
- **r0**: bit riservato per sviluppi futuri.
- **DLC**: campo di 4 bit che indica il numero di byte contenuti nel payload.
- **Data**: payload del frame in cui è contenuto il messaggio con ampiezza massima 64 bit.
- **CRC**: controllo di ridondanza ciclico (CRC) a 16 bit (15 bit più delimitatore) contiene il *checksum*, sequenza di bit che, associata al pacchetto trasmesso, viene utilizzata per verificare l'integrità del messaggio stesso che può subire alterazioni durante la trasmissione sul canale di comunicazione.
- **ACK**: il trasmettitore invia un bit recessivo e ogni ricevitore può confermare la ricezione, se ha avuto esito positivo, con un bit dominante. È un campo a due bit: il primo è il bit di riconoscimento, il secondo determina la fine del campo ed assume sempre il valore recessivo.
- **EOF**: campo a 7 bit che indica la fine del frame.
- **IFS**: campo a 7 bit che contiene il tempo richiesto per copiare un frame ricevuto correttamente nella buffer area.

2.3 Extended CAN

Figura 2.2: Frame Extended CAN

In Figura 2.2 è rappresentata la struttura logica di un frame Extended CAN. Rispetto al frame standard presenta le seguenti variazioni:

- **SRR**: sostituisce il campo RTR di cui preserva la lunghezza. Assume il valore del bit recessivo e indica che si tratta di un frame di tipo Extended CAN.
- **IDE**: bit recessivo che indica la presenza di ulteriori bit d'identificazione.
- **Identifier**: questo campo viene esteso da 11 a 29 bit. I 18 bit aggiunti possono assumere semantiche diverse a seconda dell'utilizzo. Un'esempio è lo standard SAE J1939 [26], un protocollo ad alto livello che utilizza i 29 bit per codificare alcune informazioni aggiuntive, come il destinatario dei messaggi e la presenza di messaggi multipli.
- **r1**: un ulteriore bit riservato per sviluppi futuri.

La differenza sostanziale tra le due versione del protocollo è nel numero di bit nel campo identificatore, quindi nel numero di messaggi che si possono avere. Nella versione standard questo numero è 2^{11} mentre nella seconda versione è 2^{29}. Per un approfondimento di quanto detto in queste prime tre sezioni si consulti [2].

2.4 CAN FD

Il protocollo **CAN-FD** è stato sviluppato con l'obiettivo di aumentare la banda (bandwidth), di una rete CAN mantenendo invariata la maggior parte del software e dell'hardware (in particolare il livello fisico). Come conseguenza, solo i controller del protocollo CAN devono essere aggiornati in modo da poter supportare il nuovo protocollo. È stata quindi messa in atto la retro-compatibilità: i controller nativi di protocollo CAN-FD possono prendere parte alla normale comunicazione CAN attraverso l'utilizzo dei bit riservati nel frame CAN i quali permettono a un nodo (ECU) di capire il protocollo adottato nella comunicazione. L'aumento di banda è stato realizzato mediante la possibilità di avere una velocità da 1 megabit fino a un massimo di 3 megabit al secondo. Tale velocità può essere diversa nelle due fasi che costituiscono l'invio di un frame:

- **fase di arbitraggio** in cui viene inviato l'header e il tailer del frame.
- **fase dati** in cui viene inviato il payload e i campi ad esso connessi.

Inoltre, anche il payload del frame è stato esteso ad una dimensione massima di 64 byte, poiché la precedente dimensione massima, 8 byte, si è rilevata troppo restringente in alcuni casi. Il protocollo è disponibile in due versioni differenziate dal numero di bit utilizzati nel campo *identifier (ID)*: 1) 11 e 2) 29.
Per semplificare la gestioni degli errori, ogni nodo mantiene due contatori: *Transmit Error Counter* e *Receive Error Counter*, il primo indica il numero di frame trasmessi con errore il secondo quelli ricevuti erroneamente. Un nodo parte dallo stato di **errore attivo**, cioè partecipa attivamente alla segnalazioni degli errori sul CAN bus (campo ESI assume valore dominante). Quando uno qualsiasi dei due contatori supera una certa soglia, il nodo passa nello stato di **errore passivo**, in altre parole partecipa passivamente alla segnalazioni di errori sul CAN bus (campo ESI assume valore recessivo). Quando il contatore che aveva causato il cambiamento di stato rientra al di sotto della soglia, il nodo transita nello stato precedente. Le regole per l'incremento e il decremento dei contatori sono varie e piuttosto complicate, ma il principio è semplice. La trasmissione di frame affetti da errore causa un incremento maggiore rispetto alla loro ricezione, questo perché la causa di errore con probabilità molto alta è il mittente. I frame trasmessi o ricevuti privi di errore causano un decremento dei contatori [25].
La struttura logica del frame risulta essere la seguente come mostrato in Figura 2.3

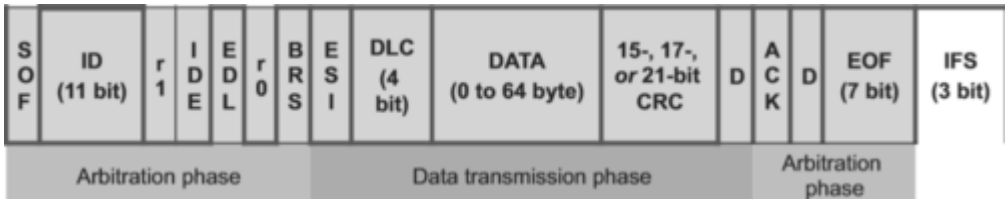

Figura 2.3: Frame Can FD

In aggiunta troviamo i seguenti campi:

- **EDL**: campo di un bit, assume il valore recessivo (1) all'interno di in un frame CAN-FD, dominante (0) altrimenti. Indica se il frame è di tipo CAN FD.

- **BRS**: campo di un bit, indica se la velocità in bit nella fase dati è la stessa della fase di arbitraggio (BRS dominante) o se nella velocità dati viene utilizzata la velocità in bit predefinita (BRS recessiva).

- **ESI**: campo di un bit, è trasmesso in caso di rilevamento di errore come bit dominate (0) da nodi nello stato di *errore attivo*, recessivo (1) da nodi nello stato di *errore passivo*.

Nonostante con questa versione siano stati alleviati i limiti principali di una rete CAN, nella maggior parte delle reti CAN non risulta essere adottato, dato che è stato rilasciato nel 2012 e comporta un aggiornamento/sostituzione delle ECU, motivo per cui il nostro lavoro è stato concentrato sulle prime due versioni, praticamente uguali sulla quantità d'informazione utile trasportata da un singolo frame [4].

Capitolo 3

Panoramica sulla crittografia

La crittografia, etimologicamente *"scrittura nascosta"*, è la pratica e lo studio di tecniche per la comunicazione sicura in presenza di terze parti chiamate **avversari**. Più in generale, la crittografia riguarda la costruzione e l'analisi di protocolli che impediscono a terzi o al pubblico di leggere messaggi privati. Vari aspetti della sicurezza delle informazioni, come riservatezza dei dati, integrità dei dati, autenticazione e non ripudio sono fondamentali per la crittografia moderna.
Il problema centrale che la crittografia vuole risolvere è il seguente: un mittente **Mitt** vuole comunicare con un destinatario **Dest** utilizzando un canale potenzialmente *insicuro*, cioè tale che altri possono intercettare i messaggi che vi transitano per conoscerli o alterarli. Per proteggere la comunicazione, i due agenti devono adottare un metodo di crittografia che permetta a **Mitt** di spedire un *messaggio* m sotto forma di *crittogramma* c, incomprensibile ad un ipotetico crittoanalista X in ascolto sul canale, ma di cui sia facile la decifrazione da parte di **Dest**. Se definiamo con MSG lo *"spazio dei messaggi"* e con CRT lo *"spazio dei crittogrammi"*, le operazioni di cifratura e decifrazione di definiscono formalmente come segue:

- **Cifratura del messaggio**: operazione con cui si trasforma un generico messaggio in chiaro m in un crittogramma c applicando un funzione

$$C : MSG \to CRT$$

- **Decifrazione del crittogramma**: operazione che permette di ricavare il messaggio in chiaro m a partire dal crittogramma c applicando una funzione

$$D : CRT \rightarrow MSG$$

Matematicamente le funzioni C e D sono una l'inverso dell'altra. Il termine *"spazio"* utilizzato per definire MSG e CRT indica un insieme a cui appartengono i messaggi e i crittogrammi senza richiedere che la funzione C o D sia definita sull'intero insieme. Infatti i messaggi effettivamente scambiati costituisco in genere un sottoinsieme di MSG che varia con l'applicazione considerata, e i relativi crittogrammi costituiscono in genere un sottoinsieme di CRT. Le funzioni C e D sono quindi parziali e la funzione C deve essere *iniettiva*, cioè a messaggi diversi devono corrispondere crittogrammi diversi [7]. Il secondo parametro di questi due processi è detto **chiave**, principalmente la crittografia è divisa in due approcci, che si differenziano proprio nel modo in cui essa viene utilizzata: **crittografia simmetrica** e **crittografia asimmetrica**.

3.1 Crittografia Simmetrica

Nella crittografia simmetrica la chiave utilizzata nel processo di cifratura da *Mitt* è la stessa utilizzata nel processo di decifrazione da *Dest*. In pratica, c'è il bisogno di interdire un segreto tra i due partecipanti della comunicazione. Stabilire questo segreto è l'aspetto più critico di questo approccio, in quanto la comunicazione deve avvenire sempre su canale insicuro, e quindi non può essere fatta in chiaro. Ragione per cui sono stati definiti protocolli che hanno lo scopo di far concordare la chiave, uno di questi è il protocollo di *Diffie-Hellman* (DH). Il cifrario simmetrico standard ad oggi utilizzato prende il nome di **AES** per *Advanced Encryption Standard* [29], il quale risulta molto efficiente dal punto di vista computazionale. L'approccio simmetrico è anche utilizzato per garantire l'autenticazione, cioè l'identità del mittente, e l'integrità del crittogramma ricevuto, attraverso l'utilizzo del MAC (Message Authentication Code). Tale tecnica è una funzione in due parametri (chiave e messaggio) che genera un'immagine breve di esso, la quale può essere calcolata solo da un mittente conosciuto dal destinatario previ opportuni accordi. Ovve-

ro, anch'essa richiede che venga concordata una chiave dato che viene utilizzata sia dal mittente nel calcolo del MAC, sia dal destinatario nella verifica.

3.2 Crittografia Asimmetrica

La crittografia asimmetrica non presenta la necessità di concordare una chiave tra mittente e destinatario, in quanto ogni utente dispone di una coppia di chiavi che prendono il nome di chiave pubblica, $\mathbf{K_{Pub}}$, e chiave privata, $\mathbf{K_{Priv}}$: la prima è conosciuta da tutti gli utenti del sistema ed è usata nel processo di cifratura, mentre la seconda è nota solo all'utente in questione ed è utilizzata nel processo di decifrazione. Quando un mittente, Mitt, vuole mandare un messaggio a un destinatario, Dest, si procura la sua chiave pubblica e cifra il messaggio attraverso un processo di cifratura concordato e noto, quando Dest riceve il crittogramma, per risalire al messaggio applica il conseguente processo di decifrazione utilizzando la propria chiave privata.
Sebbene sia stato risolto il problema di concordare una chiave tra mittente e destinatario attraverso l'utilizzo di una coppia di chiavi, questo criterio ne presenta un altro, ovvero procurarsi la chiave pubblica corretta. Dato il fatto che è nota a tutti, un utente potrebbe infatti spacciarsi per un altro e ricevere messaggi non destinati a lui. Per far fronte a questa problematica nella crittografia a chiave pubblica è stato introdotto l'uso dei **certificati** [7], i quali forniscono la chiave pubblica di un utente attestando la sua identità. I cifrari a chiave pubblica fondano la loro sicurezza su funzioni matematiche dette **one-way trap-door** che sono computazionalmente facili[1] da calcolare ma computazionalmente difficile da invertire[2]. Alcuni esempi sono:

- Fattorizzazione.
- Calcolo della radice in modulo (modulo un numero composto).
- Calcolo del logaritmo discreto.

[1]Richiede tempo polinomiale nella dimensione dell'input.
[2]Richiede tempo esponenziale nella dimensione dell'input.

Con il passare degli anni e dato l'aumento delle risorse computazionali, i cifrari basati su queste funzioni per garantire un adeguato livello di sicurezza, richiedono un numero considerevole di bit nella chiave, aumentando così i tempi di elaborazione. Ragione per cui, negli ultimi venti anni, la crittografia a chiave pubblica utilizza la **crittografia su curve ellittiche** in breve ECC (Elliptic Curve Cryptography). La ECC offre prestazioni migliori e maggiore sicurezza, a parità di bit di chiave, rispetto ai sistemi a chiave pubblica di *prima generazione*, i quali sono: il cifrario RSA e il protocollo DH per lo scambio pubblico delle chiavi. Inoltre, anche la **firma digitale** fa uso di cifrari a chiave pubblica.

Capitolo 4

Specifiche AUTOSAR per la Sicurezza delle Comunicazioni e lo Scambio di Chiavi

Lo sviluppo di software per applicazioni automobilistiche è costantemente aumentato negli ultimi decenni. Nel settore automobilistico, il software è un'area chiave per l'innovazione e, di contro, anche per i costi di sviluppo. L'elettronica e il software coprono oltre il 90% di tutte le innovazioni e determinano fino al 40% dei costi di sviluppo di un veicolo, di cui, dal 50% al 70% sono dedicati al software delle centraline elettroniche. Data l'importanza dello sviluppo software per l'innovazione automobilistica e i costi ad esso connessi, la standardizzazione di un'architettura software, di una metodologia, di una piattaforma software e di interfacce applicative può aiutare a supportare la gestione della crescente complessità dei sistemi e delle loro integrazioni, oltre a limitare i costi di sviluppo. Sotto questo scenario è stata fondata **AUTOSAR** [12] una partnership di sviluppo mondiale di entità interessate e coinvolte a vario titolo nel settore automobilistico. È stata fondata nel 2003 con lo scopo di creare e stabilire un'architettura software aperta e standardizzata per il sistema automobilistico. Fornisce linee guida per la specifica e implementazione di una serie di moduli software di base, definisce le interfacce delle applicazioni e crea una metodologia di sviluppo comune basata su un formato di scambio standardizzato. I moduli software di base resi disponibili dall'architettura software stratificata AUTOSAR

possono essere utilizzati in veicoli di diversi produttori e per componenti elettronici di diversi fornitori, riducendo così le spese per la ricerca e lo sviluppo e padroneggiando la crescente complessità delle architetture elettroniche e software per auto. AUTOSAR è stato ideato per spianare la strada a sistemi elettronici innovativi che migliorano ulteriormente le prestazioni, la sicurezza e la compatibilità ambientale e per facilitare lo scambio e l'aggiornamento di software e hardware per tutta la durata del veicolo. Mira a essere preparato per le prossime tecnologie e migliorare l'efficienza in termini di costi senza scendere a compromessi in termini di qualità.

4.1 Struttura del software

Nello sviluppo del software, AUTOSAR ha previsto una struttura a strati in modo da poter lavorare a diversi livelli d'astrazione. È noto che, questa soluzione, è ampiamente utilizzata nel mondo dell'informatica poiché offre numerosi vantaggi, in particolare la possibilità di dividere il problema di partenza in sotto-problemi in modo da scrivere vari moduli software indipendenti tra loro che collaborano per risolvere il problema stesso.
In particolare, l'**architettura AUTOSAR** consiste in tre livelli d'astrazione, i quali vengono eseguiti su un microcontrollore (ECU): i) **Application**, ii) **Run-time Environment**, e iii) **Basic software** [28].
L'**application layer** è la parte in cui le varie componenti di una funzionalità non di base vengono implementate. Queste componenti software comunicano tra di loro e con il software di base *via RTE* e sono indipendenti dall'hardware sottostante.
Il livello **Runtime Environment (RTE)** realizza la comunicazione tra le componenti software e il software di base. Le varie componenti software comunicano tra loro e/o con i moduli software di base esclusivamente tramite RTE. Questo consente ai componenti software di essere indipendenti da qualsiasi ECU specifica e da altri componenti software. Viene generato un RTE per ogni partizione (o *core*) di una ECU. Le comunicazioni posso essere categorizzate in base alla locazione dei partecipati:

- **Inter-ECU communication**: si ha quando le componenti software risiedono su ECU diverse.

- **Intra-ECU communication**: si ha quando le componenti software risiedono sulla medesima ECU. Qui si vanno a differenziare altri due scenari:
  - **Inter-Partition communication**: le componenti software vengono eseguite su diverse partizioni della stessa ECU.
  - **Intra-Partition Communication**: la comunicazione avviene all'interno della stessa partizione.

Un importante passo nella standardizzazione è stato quello di definire il **Basic software BSW**. Esso è costituito da moduli software che definiscono funzionalità di base per una ECU. Tali funzionalità si possono raggruppare secondo il servizio offerto in:

- **Input/output**: offre accesso standardizzato ai vari sensori e periferiche.
- **Memory**: offre accesso standardizzato alla memoria non volatile.
- **Crypto**: accesso standardizzato alle primitive crittografiche inclusi acceleratori hardware interni/esterni.
- **Communication**: offre meccanismi per la comunicazione tra ECU all'interno dello stesso veicolo.
- **Off-board Communication**: offre meccanismi per la comunicazione *Vehicle-to-X* (X sta per infrastruttura) e tra ECU su veicoli differenti.
- **System**: offre servizi simili a quelli offerti da un sistema operativo, permette di interagire con una ECU tramite chiamate di sistema.

Anche lo BSW presenta una struttura a strati, logicamente è diviso come segue [27]:

- **Micro-controller abstraction layer MCAL**: è il livello software più basso. Contiene i driver interni che hanno accesso diretto al microcontrollore e alle periferiche interne. Il suo compito è rendere il livello software indipendente dal microcontrollore.
- **ECU abstraction layer**: si trova sopra al MCAL e si estrae dallo schema della ECU. È implementato per una ECU specifica (quindi dipendente dall'hardware) e

offre un'API per l'accesso a periferiche e dispositivi indipendentemente dalla loro posizione (onchip/offchip) e dalla loro connessione al microcontrollore (pin della porta, tipo di interfaccia) per creare livelli software più elevati indipendentemente dal layout hardware della ECU.

- **Service layer**: è il livello più alto. È di fondamentale importanza per il software applicativo, in quanto il suo compito è fornire servizi di base alle applicazioni.

- **Complex Drivers Layer**: si espande dall'hardware al RTE, fornisce la possibilità di integrare funzionalità per scopi speciali, ad esempio funzionalità che non sono contenute all'interno delle specifiche AUTOSAR.

Alla fine troviamo la struttura complessiva mostrata in Figura 4.1.

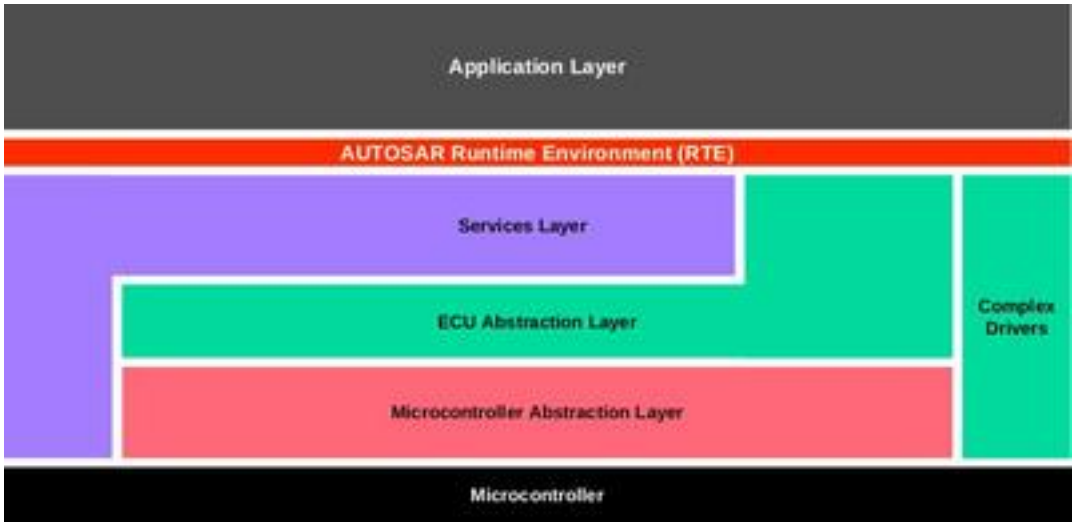

Figura 4.1: Architettura Autosar

Questo stile architetturale ha facilitato molto lo sviluppo di applicazioni in quanto ha offerto la possibilità di sviluppare software indipendentemente dall'hardware, inoltre la definizione di vari *basic software* ha omogeneizzato e reso disponibili funzionalità che sono alla base di un qualsiasi applicazione si voglia realizzare. Così, l'industria automobilistica sta affrontando la sfida dell'importanza, in rapida crescita, del software e delle funzionalità basate su esso oltre che della sua complessità, dato che la ricerca ha dimostrato che la complessità del software è una delle principali ragioni del ritardo del progetto

[3]. Inoltre l'essere dipendenti dall'hardware nella scrittura del codice rappresentava un problema anche sotto l'aspetto economico, dal momento che cambiando, anche di poco, le specifiche di una ECU, il codice doveva essere scritto nuovamente. Per la verifica del software lo standard ISO262622 (titolato *Road vehicles – Functional safety*) ha chiaramente raccomandato l'analisi statica del software per ridurre gli errori a run-time, dato che risulta più problematico correggere eventuali errori a run-time [16].
AUTOSAR ha delineato le linee guida anche per quanto riguarda la comunicazione sicura all'interno del veicolo. In particolare, per il problema che vogliamo affrontare, oltre ad aver definito delle linee guida per la gestione delle chiavi, Autosar ha definito anche due *basic software*: **crypto stack** e **communication stack**; il primo entra in gioco completamente, mentre del secondo ci interessa un modulo in particolare: **Secure Onboard Communication (SecOC)** [22].

4.2 Secure Onboard Communication

L'autenticazione e l'integrità dei dati sensibili sono necessarie per proteggere la funzionalità corretta e sicura del veicolo: ciò garantisce che i dati ricevuti provengano dalla giusta ECU e abbiano il valore corretto. Il modulo SecOC definito in AUTOSAR mira a definire ed introdurre meccanismi di autenticazione efficienti -sotto il profilo delle risorse- e praticabili per i dati critici a livello di PDU (Protocol Data Unit), cioè dati da muovere fra livelli diversi della struttura ISO/OSI. I meccanismi di autenticazione devono essere perfettamente integrati con gli attuali sistemi di comunicazione AUTOSAR. L'impatto sul consumo di risorse deve essere minimizzato al fine di consentire la protezione come componente aggiuntivo per i sistemi legacy. La specifica si basa sul presupposto che vengano utilizzati principalmente approcci di autenticazione simmetrici con particolare riferimento all'utilizzo del *Message Authentication Code (MAC)*. Gli approcci di autenticazione simmetrici raggiungono lo stesso livello di sicurezza con chiavi molto più piccole rispetto agli approcci asimmetrici e possono essere implementate in modo compatto ed efficiente nel software e nell'hardware. Tuttavia, la specifica fornisce il livello necessario di astrazione in modo da poter utilizzare sia approcci simmetrici che approcci di autenticazione asim-

metrici. La Figura 4.2 mostra l'integrazione del modulo SecOC come parte dello stack di comunicazione AUTOSAR.

In questo design, PduR (PDU router in figura) è responsabile del routing dei PDU relativi alla sicurezza in entrata e in uscita verso il modulo SecOC. Il modulo SecOC deve quindi aggiungere o elaborare le informazioni rilevanti per la sicurezza, nonché eseguire il calcolo e la verifica del meccanismo di autenticazione, e propagare i risultati sotto forma di PDU al PduR, sarà esso il responsabile dell'inoltro successivo. Inoltre, il modulo SecOC si avvale dei servizi crittografici forniti dal *crypto stack* e interagisce con il modulo RTE per consentire la gestione e l'aggiornamento della chiave di sessione. Il modulo SecOC viene utilizzato in tutte le centraline in cui è necessaria una comunicazione sicura [22].

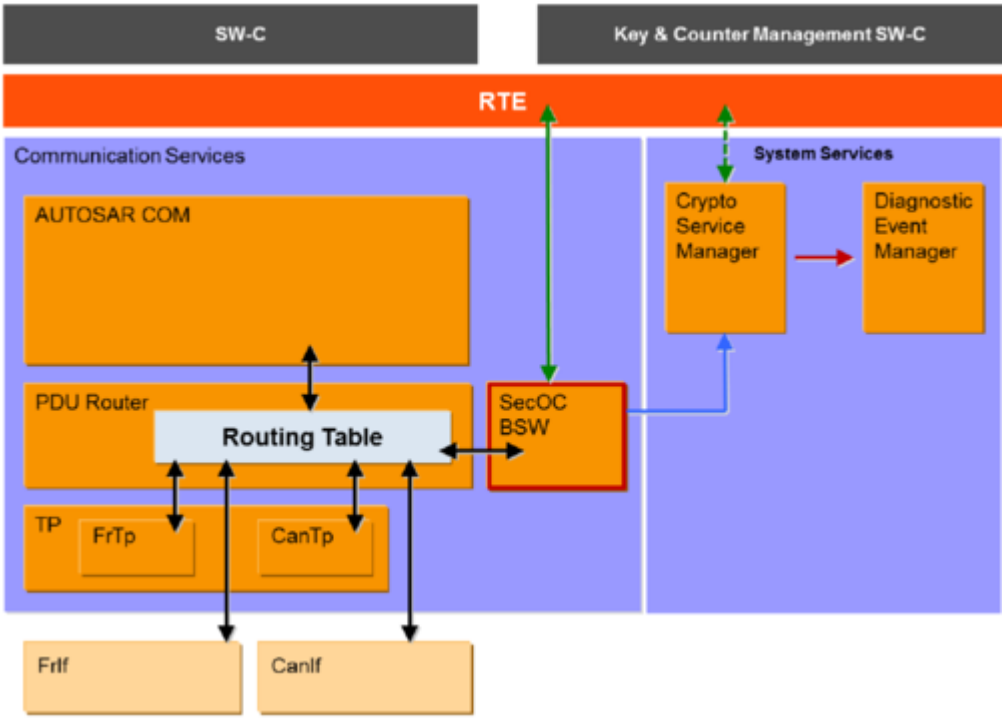

Figura 4.2: Design di sistema

4.3 Crypto Stack

Il crypto stack offre un accesso standardizzato ai servizi crittografici per applicazioni e funzioni di sistema. I servizi crittografici sono, ad esempio, il calcolo degli hash, la verifica

delle firme asimmetriche e la crittografia simmetrica dei dati. Questi servizi dipendono dalle primitive crittografiche e dagli schemi crittografici sottostanti. Il CSM deve consentire a diverse applicazioni di utilizzare lo stesso servizio ma utilizzando primitive e/o schemi sottostanti diversi. Ad esempio, un'applicazione potrebbe dover utilizzare il servizio hash per calcolare un digest SHA2[1] e un'altra potrebbe dover calcolare un digest SHA1[2]. Oppure un'applicazione potrebbe aver bisogno di verificare una firma che è stata calcolata con lo schema di firma RSASSA-PKCS1-V1_ e utilizzare SHA1 come primitiva hash sottostante, mentre un'altra applicazione potrebbe aver bisogno di verificare una firma calcolata con uno schema diverso che utilizza SHA2 come primitiva hash. Lo stack crittografico consente di configurare quali servizi sono necessari e di creare diverse configurazioni per ciascun servizio in cui è possibile scegliere schemi e primitive. Inoltre, poiché il calcolo di molti dei servizi crittografici è computazionalmente gravoso, è necessario prevedere disposizioni per programmare questi lunghi calcoli, cioè tali computazione devono essere configurabili in modo da poter essere eseguiti in modo sincrono o asincrono.
Crypto Stack fornisce servizi con funzionalità di crittografia, basate su librerie software o su moduli hardware. Inoltre, sono possibili configurazioni miste, ad esempio se un modulo hardware non è in grado di fornire da solo le funzionalità necessarie. I seguenti algoritmi crittografici sono supportati dal crypto stack:

- Generazione di un numero random:
  - Deterministic Random Number Generator (DRNG).
  - True Random Number Generator (TRNG).
- Crittografia simmetrica:
  - AES, con chiavi da 128 e 256 bit.
  - PRESENT, con chiavi da 128 bit.

[1]SHA2 è un collezione di quattro funzioni hash crittografiche che dato come input una stringa di bit che rappresenta un messaggio, producono rispettivamente un immagine (digest) di 224,256,384 e 512 bit.

[2]SHA1 è una funzione hash crittografica che dato come input una stringa di bit che rappresenta un messaggio, produce un immagine (digest) di 160 bit.

- Crittografia asimmetrica e gestione delle firme:
  - RSA, con chiavi da 1024, 2048, 3072 e 4096 bit.
  - Edwards-curve Digital Signature Algorithm (EDSA).
  - Diffie-Hellman sulla curva ellitica *Curve25519*.
- Hash
  - SHA-2 a 224, 256, 384 e 512 bit.
  - SHA-3 a 224, 256, 384 e 512 bit.
  - BLAKE a 224, 256, 384 e 512 bit.
  - RIPEMD-160.

Il crypto stack si espande su tutta l'architettura dei livelli AUTOSAR. Al livello più basso, *Microcontroller Abstraction Layer*, si trovano i **crypto driver (CRYPTO)**, questi moduli contengono le effettive implementazioni delle diverse istanze hardware e software crittografiche. Al livello intermedio, *Hardware Abstraction Layer*, vi è la **crypto interface (CRYIF)** mentre al *Service Layer* troviamo il **crypto service manager CSM** [17].

4.3.1 Crypto Service Manager (CSM)

Il Crypto Service Manager (CSM) controlla l'accesso simultaneo di uno o più client a uno o più servizi di crittografia sincrona/asincrona. Offre code prioritarie per gestire i *job* che non possono essere elaborati direttamente dal CRYPTO dedicato. Le richieste al CSM per le routine crittografiche sono rappresentate come **job**.
Le funzionalità fornite dal CSM riguardano le seguenti aree:

- Calcolo dell'HASH.
- Generazione e verifica del MAC.
- Generazione e verifica della firma digitale.
- Encryption e decryption utilizzando algoritmi simmetrici o asimmetrici.

- Generazione numeri random.
- Operazioni per la gestione delle chiavi.

I servizi del CSM sono generici e CSM consente a diverse applicazioni di utilizzare lo stesso servizio con algoritmi crittografici diversi. Ciò è garantito dalla possibilità di configurare e inizializzare i servizi individualmente. Le routine crittografiche effettive sono incapsulate dal CSM. Non è necessario che l'entità che utilizza il servizio si preoccupi se la routine è implementata in software o hardware o quale modulo CRYPTO mantiene effettivamente la routine crittografica richiesta. CSM fornisce uno strato di astrazione a tutte le funzionalità crittografiche disponibili nello stack crittografico. Poiché il calcolo dei servizi crittografici potrebbe essere molto intenso dal punto di vista computazionale, l'elaborazione dei lavori deve essere considerata come sincrona o asincrona. Il CSM può avere diverse code in cui i lavori asincroni vengono elaborati in base alla loro priorità. Ogni coda del CSM è mappata su un singolo oggetto Crypto Driver, consentendo così l'accesso alle primitive crittografiche dell'oggetto Crypto Driver selezionato. Dopo che CRYPTO ha rifiutato una richiesta di servizio CSM perché è occupato, il lavoro specifico viene inserito nella coda CSM appropriata, tenendo conto della sua priorità. Quando viene utilizzata l'elaborazione sincrona, il servizio CSM verrà eseguito immediatamente nel contesto del chiamante. Il risultato della routine crittografica sarà disponibile direttamente al ritorno della funzione. I job asincroni vengono successivamente elaborati dal CRYPTO dedicato, nel contesto di una funzione principale pianificata o nell'hardware. Se il particolare oggetto driver CRYPTO rifiuta il lavoro perché è occupato, il CSM inserisce la richiesta di servizio nella rispettiva coda dei lavori CSM. Il CRYPTO notifica a CRYIF il completamento di un lavoro asincrono utilizzando una funzione di callback del CRYIF. E CRYIF inoltra i risultati mediante una funzione di callback del CSM [20].

4.3.2 Crypto Interface (CRYIF)

Il CRYIF è racchiuso dal CSM di livello superiore e dal CRYPTO di livello inferiore. Riceve richieste dal CSM e le mappa all'operazione crittografica appropriata nel CRYPTO. CRYIF inoltra le richieste fornite da CSM al CRYPTO specifico. Le notifiche che

riceve dal livello inferiore informano dell'esito nel caso in cui la richiesta fosse asincrona. Il CRYIF può gestire diversi moduli CRYPTO. Potrebbero esistere, ad esempio, un modulo CRYPTO per un modulo hardware crittografico esterno e un modulo CRYPTO che contiene una libreria software crittografica [19].

4.3.3 Crypto Driver (CRYPTO)

Un CRYPTO contiene in genere le implementazioni crittografiche effettive e supporta l'archiviazione delle chiavi, la configurazione delle chiavi e la gestione delle chiavi per i servizi di crittografia. Può avere uno o più oggetti Crypto Driver, con aree di lavoro separate. Ogni oggetto Crypto Driver può fornire arbitrariamente molte primitive crittografiche. Una primitiva crittografica è un'istanza di un algoritmo crittografico configurato. Un oggetto Crypto Driver può eseguire solo una cripto-primitiva in un dato istante. Il concetto di diversi moduli CRYPTO e diversi oggetti Crypto Driver consente implementazioni diverse e simultanee degli stessi servizi crittografici. Possono esistere varianti di moduli CRYPTO con diversi obiettivi di ottimizzazione. Ad esempio, lo stesso algoritmo di hash potrebbe essere implementato in due diversi moduli CRYPTO, uno con una soluzione hardware più veloce (più costosa) e l'altro con una soluzione software più lenta (più economica). L'esempio seguente descrive un possibile scenario che utilizza diversi moduli CRYPTO: esistono due implementazioni CRYPTO di due diversi fornitori. Un CRYPTO è un'astrazione di una soluzione hardware ("CRYPTO_ HW") e l'altro CRYPTO è una soluzione software pura ("CRYPTO_ SW"). CRYPTO_ SW è una libreria crittografica che fornisce servizi di hash e un generatore di numeri casuali (pseudo). Per consentire l'elaborazione di entrambi i servizi in parallelo, CRYPTO_ SW ha due oggetti Crypto Driver, uno per i servizi di hash ("CDO_ HASH") e uno per il generatore di numeri casuali ("CDO_ RNG"). Se alcune routine crittografiche non devono essere eseguite in parallelo, devono essere collocate nello stesso oggetto Crypto Driver [18].

4.4 Gestione delle chiavi

AUTOSAR fornisce delle linee guide riguardo alla struttura per la gestione delle chiavi. Sono state definite l'entità che ne fanno parte e lo stile architetturale, quindi, sono stati definiti i ruoli dei vari partecipanti. Sul tipo di comunicazione sono state proposte più soluzioni lasciando così la possibilità, al momento dell'implementazione, di valutare pro e contro di ogni soluzione. Lo stile architetturale proposto è del tipo **client-server**, vi è un **master key** addetto alla distribuzione e all'aggiornamento della *chiave di sessione*, che funge da server, i cui **client** sono le diverse ECU che si interfacciano con esso attraverso un modulo dedicato. Per la collocazione del *key-master* sono definite due soluzioni mutuamente esclusive:

1. Localmente all'interno del veicolo.
2. Da remoto utilizzando un servizio cloud.

La prima soluzione prevede l'installazione di una ECU che offre il servizio sopra citato, la seconda invece apre due scenari diversi. Il primo presume che ogni centralina abbia la propria connessione con il *key-master*. Ciò è realizzabile solo in veicoli che si trovavano in una fascia alta di mercato, in quanto offre ECU con risorse computazionali e funzionalità maggiori. Il secondo invece prevede una connessione per l'intero veicolo, effettuata e mantenuta dal *sistema di infotainment* che, una volta ricevuta la chiave da remoto, deve effettuare la distribuzione di essa all'interno dell'autovettura riconducendosi alla soluzione numero uno. Il **KeyM** è un modulo del *crypto stack* che è delegato alla gestione della chiave di sessione utilizzando le primitive che operano su di essa offerte dal CSM. Consiste in due sotto-moduli:

- **Crypto key sub-module**: rappresenta il client effettivo dato che viene utilizzato per inizializzare, aggiornare e mantenere la chiave crittografica per una centralina interfacciandosi con il *key master*. Il *crypto key sub-module* una volta ricevuta la chiave di sessione, la fornisce al CSM così da poterla utilizzare nei *job* crittografici.

- **Certificate sub-module**: consente di configurare i certificati, fornendo interfacce per archiviarli e verificarli. I certificati sono memorizzatati secondo una struttura gerarchica all'interno della quale troviamo le seguenti tre categorie:
  - **Root**.
  - **Intermediate**.
  - **End user**.

  Le prime due categorie vengono memorizzate nella fase di costruzione del veicolo o della centralina. I certificati *end-user* sono quelli che vengono verificati dal modulo durante la comunicazione, se la verifica ha avuto esito positivo, vengono inserirti lungo la catena opportuna. I certificati seguono lo standard **x.509** [1].

Tutto questo è descritto accuratamente in [21].

4.4.1 Conclusioni

Sul come la chiave deve essere scambiata tra il *key master* e una ECU non è stato detto niente, a parte il fatto che deve essere comunicata in maniera sicura. Questo è il contesto sul quale ci siamo concertati sapendo che è un problema molto critico e sull'esito di questa fase si basano tutti i meccanismi di autenticazione successivi, ragione per cui si può pensare di pagare qualcosa in più dal punto di vista computazionale al fine di raggiungere una forte robustezza. La crittografia simmetrica per questo problema non è d'aiuto in quanto una sua prerogativa fondamentale è quella che vi sia un segreto condiviso tra le due parti oggetto del dialogo: **la chiave di sessione**. Data la natura del problema, la soluzione proposta in questo tirocinio prevede l'utilizzo di un cifrario asimmetrico per questa fase in modo da rendere possibile una comunicazione cifrata senza la necessità di avere un segreto condiviso tra le parti che verrà spiegata e dettagliata nei Capitoli 6 e 7.

Capitolo 5

Plug-and-Secure Communication for CAN

Un protocollo per lo scambio di una chiave di sessione è stato proposto in [13] e prende il nome di **Plug-and-Secure Communication for CAN**, il quale risulta molto efficiente dal punto di vista della sua specifica data ma, presenta difficoltà implementative in quanto prevede che due nodi si debbano sincronizzare prima di iniziare a comunicare, cosa che non è facilmente realizzabile fra ECU.

5.1 Specifica del Protocollo

Nella spiegazione del protocollo si utilizzano i nomi classici presenti nella letteratura crittografica:

- **Alice**: rappresenta la prima ECU coinvolta nella comunicazione.
- **Bob**: rappresenta la seconda ECU coinvolta nella comunicazione.
- **Eve**: rappresenta l'attaccante.

L'idea base per questo approccio è che Alice e Bob si accordino su un segreto condiviso, i.e., la chiave, per mezzo di una discussione pubblica usando messaggi standard CAN. In particolare, entrambi i nodi simultaneamente trasmettono appropriate CAN frames,

così che Eve sia solo in grado di vedere la sovrapposizione di entrambi i messaggi, senza conoscere il contenuto esatto di ognuno di loro. Tuttavia, dato che Alice e Bob sanno cosa hanno trasmesso e dato che possono vedere la sovrapposizione di entrambi i messaggi, possono facilmente concludere cosa l'altro nodo ha trasmesso e così stabilire un segreto condiviso (chiave) che Eve non conosce.

Per una concreta realizzazione, si è fatta fede a una proprietà caratteristica del CAN bus: **il bit 0 è dominante e il bit 1 è recessivo**. Infatti, se Alice e Bob simultaneamente trasmettono un certo bit (com'è richiesto dall'approccio in questione), ci sono in totale quattro casi che potrebbero verificarsi rappresentati in Figura 5.1.

Alice	Bob	Effective Bit on CAN Bus
0	0	0
0	1	0
1	0	0
1	1	1

Figura 5.1: Risultato sul CAN bus

In pratica il bus realizza l'**AND logico** dei due bit trasmessi, come mostrato in Figura 5.1. L'attuale procedura per l'accordo sul segreto condiviso (chiave) tra Alice e Bob è un approccio multi-step:

1. Alice e Bob generano indipendentemente l'uno dall'altro una stringa random di bit $\mathbf{R_{Bob}}$ e $\mathbf{R_{Alice}}$ di lunghezza predefinita **N**.

$$R_{Alice} = 0\,1\,1\,0\,1\,0\,0\,1\,0\,1$$
$$R_{Bob} = 1\,0\,1\,1\,0\,1\,0\,1\,1\,0$$

Figura 5.2: Esempio con N pari a 10

2. Alice e Bob estendono queste stringhe di bit in modo tale che dopo ogni bit ci sia il bit inverso, $\mathbf{S_{Bob}}$ e $\mathbf{S_{Alice}}$ hanno lunghezza **2N**.

$$S_{Alice} = 01\ 10\ 10\ 01\ 10\ 01\ 01\ 10\ 01\ 10$$
$$S_{Bob} = 10\ 01\ 10\ 10\ 01\ 10\ 01\ 10\ 10\ 01$$

Figura 5.3: Esempio con N pari a 10

3. Alice e Bob trasmettono simultaneamente $\mathbf{S_{Alice}}$ e $\mathbf{S_{Bob}}$, la stringa risultante sul bus $\mathbf{S_{eff}}$ è data da $\mathbf{S_{Bob}}$ **and** $\mathbf{S_{Alice}}$.

$$S_{eff} = 00\ 00\ 10\ 00\ 00\ 00\ 01\ 10\ 00\ 00$$

Figura 5.4: Esempio con N pari a 10

4. Alice e Bob determinano tutte le che coppie in $\mathbf{S_{eff}}$ che includono **il bit 1**[1].

5. Alice e Bob eliminano i bit nelle loro stringhe originali $\mathbf{R_{Alice}}$ e $\mathbf{R_{Bob}}$ corrispondenti con le coppie determinate nel passo 4. I risultati sono due sequenze accorciate di bit $\mathbf{K_{Alice}}$ e $\mathbf{K_{Bob}}$. Si nota che questo è fatto perché, se il bit effettivo sul bus è 1, è chiaro che entrambi hanno trasmesso un 1. Allo stesso modo, dato che i due bit in una coppia sono sempre uno l'inverso dell'altro è chiaro che entrambi i nodi hanno trasmesso 0.

$$K_{Alice} = 0\ 1\ \cancel{1}\ 0\ 1\ 0\ \cancel{0}\ \cancel{1}\ 0\ 1 = 0\ 1\ 0\ 1\ 0\ 0\ 1$$
$$K_{Bob} = 1\ 0\ \cancel{1}\ 1\ 0\ 1\ \cancel{0}\ \cancel{1}\ 1\ 0 = 1\ 0\ 1\ 0\ 1\ 1\ 0$$

Figura 5.5: Esempio con N pari a 10

6. Il risultato di Alice $\mathbf{K_{Alice}}$ è ora esattamente l'inverso del risultato di Bob $\mathbf{K_{Bob}}$, che alla fine è il segreto condiviso (chiave).

[1]Nella Figura 5.4 sono le coppie 3,7 e 8 (assumendo che si inizi a contare da 1).

Chiaramente cosa rimane dopo il passo 5 sono i bit che sono differenti nelle stringhe iniziali. Quando simultaneamente trasmettono $\mathbf{S_{Bob}}$ e $\mathbf{S_{Alice}}$, otterranno sempre '00'per le coppie corrispondente a questi bit.

Così, anche potendo leggere $\mathbf{S_{eff}}$, Eve ha conoscenza solo che Alice e Bob hanno i bit inversi nella sequenza di partenza nelle posizioni indicate dalle coppie, ma non è in grado di dire chi ha 0 e chi ha 1. Alice e Bob, invece, conoscono i bit trasmessi da loro stessi, e possono concludere che il rispettivo nodo ha trasmesso il bit inverso valutando la sequenza finale. $\mathbf{K_{Bob}}$ e $\mathbf{K_{Alice}}$ sono sconosciute a Eve, ma sono conosciute da Alice e Bob.

Con lo schema proposto è possibile stabilire un segreto condiviso (chiave) tra Alice e Bob per mezzo di una semplice discussione pubblica: semplicemente trasmettendo e ricevendo frames CAN e interpretando la sovrapposizione dei frame sul bus nel modo giusto. Di conseguenza, la complessità coinvolta è estremamente bassa, specialmente se andiamo a compararla con schemi già esistenti, come ad esempio DH. Tuttavia, potrebbe essere fatto in maniera completamente automatica, ed è quindi chiaramente superiore rispetto alla distribuzione manuale delle chiavi.

L'integrazione concreta dell'idea centrale in una soluzione in piena regola con meccanismi di protocollo adeguati è ancora in corso. In particolare, per una soluzione completa sono necessari meccanismi addizionali, ad esempio per attivare la trasmissione sincronizzata di Alice e Bob, per avviare l'intera procedura e, ad esempio, per adattare i nodi coinvolti. In una realizzazione pratica, la trasmissione simultanea delle stringhe di bit $\mathbf{S_{Bob}}$ e $\mathbf{S_{Alice}}$ preferibilmente si farebbe nel payload di un frame CAN, rappresentando così una deviazione dallo standard CAN, dove la trasmissione simultanea potrebbe verificarsi solo durante la fase di arbitraggio quando vengono trasmessi gli identificatori CAN.

Una cosa importante da notare è che il numero di bit in un payload in un frame CAN è limitato a 64 bit nel caso di CAN standard e 512 bit nel caso di CAN FD. Inoltre, la lunghezza del segreto condiviso effettivo che possiamo generare con una esecuzione della procedura proposta per una data lunghezza N delle stringhe random iniziali $\mathbf{R_{Bob}}$ e $\mathbf{R_{Alice}}$ non è costante, ma dipende da quanti valori di $\mathbf{R_{Bob}}$ e $\mathbf{R_{Alice}}$ sono uguali. Chiaramente, questo può variare tra 0 e N, con un valore atteso di N/2. Data che nel passo due le sequenze iniziali sono estese di un fattore 2, saranno trasmessi sul bus 2N bit, l'efficienza

complessiva α che si riferisce alla lunghezza del segreto condiviso (parte effettivamente usata delle stringhe iniziali), dopo un round dell'approccio proposto, rispetto ai 2N bit richiesti per stabilire questo segreto, è in generale data da

$$0 \leq \alpha \leq 1/2 \tag{5.1}$$

con un valore atteso di 1/4.

Questo significa che in media, quattro bit di payload dovranno essere trasmessi simultaneamente da Alice e Bob in maniera tale da stabilire un bit segreto.

Dato che per ottenere una sicurezza all'avanguardia di solito sono richieste chiavi simmetriche di 128 bit o anche 256 bit, è abbastanza chiaro che per gli standard CAN e CAN FD una singola esecuzione dell'approccio proposto non è abbastanza per generare un numero sufficiente di bit segreti. Comunque, anche per affrontare questo problema sono richiesti meccanismi di protocollo adeguati, che idealmente consentirebbero la generazione di chiavi di lunghezza arbitraria. Questo potrebbe essere fatto eseguendo ripetutamente la procedura e combinando i bit segreti ottenuti ad ogni round nel modo opportuno.

Un altro aspetto importante che sarebbe efficientemente gestito applicando tale approccio è quello relativo all'aggiornamento periodico delle chiavi. Questo è generalmente utile per limitare il tempo durante il quale una certa chiave è usata o equivalentemente il numero di messaggi che sono sicuri utilizzando una particolare chiave. Un aggiornamento periodico delle chiavi è altamente raccomandato nella sicurezza in generale. Per aggiornare una chiave, però, già un numero limitato di bit è sufficiente, dato che potrebbero essere combinati con la chiave vecchia in un modo appropriato. Questo potrebbe essere fatto usando una funzione hash crittografica sicura, per esempio. Così, la procedura proposta potrebbe essere regolarmente inserita nelle regolari comunicazioni CAN in modo tale da generare la chiave e aggiornare la chiave usata in accordo alla crescita del livello di sicurezza. La procedura proposta, tuttavia, ha delle limitazioni. Infatti, questa non può essere estesa a una configurazione a più nodi in modo semplice. Ovvero non può generare **chiavi di gruppo**. Come già sottolineato in precedenza, le stringhe di bit $\mathbf{S_{Bob}}$ e $\mathbf{S_{Alice}}$ sono preferibilmente trasmesse nel payload di un frame CAN. Senza nessuna misura addizionale, comunque, questo potrebbe portare a problemi nella realizzazione

pratica.

In particolare, con una implementazione diretta dell'approccio proposto la sovrapposizione dei frame sul bus potrebbe violare la **regola di stuffing**[2]. Anche se individualmente le due stringhe di bit aderiscono a tale regola, ciò non può essere assicurato anche per la stringa effettiva sul bus. Inoltre c'è da considerare che i due nodi si devono sincronizzare prima di iniziare la comunicazione, quindi bisogna affiancare al protocollo anche un meccanismo di sincronizzazione; di conseguenza per una sua implementazione effettiva, si deve lavorare ancora per trovare soluzioni alle problematiche implementative trovate [13].

[2]In informatica il bit stuffing è una tecnica che consiste nell'aggiungere dei bit a zero ad un flusso di dati numerici

Capitolo 6

La nostra soluzione

6.1 Analisi dei requisiti

L'analisi dei requisiti è una fase preliminare allo sviluppo del software che mira a far comprendere quali sono le funzionalità che si devono implementare, nonché, quali funzionalità sono richieste dal committente. In questa fase non si vanno a delineare solo le funzionalità che si devono garantire, ma si comprendono anche quali sono i vincoli che impone il contesto e quali proprietà non funzionali il software deve avere. In merito ai requisiti individuati è stato progettata e sviluppata la nostra proposta per lo scambio della chiave di sessione fra ECU.

6.1.1 Requisiti funzionali

Analizzando il problema sono stati delineate le seguenti funzionalità:

1. Minimizzare l'uso della banda di rete.
2. Avere stabilità di esecuzione.
3. Scambiare la chiave all'intero di frame Standard CAN o Extended CAN.
4. Garantire la confidenzialità.
5. Avere breve durata.

6. Rispettare le linee guida AUTOSAR.

7. Aggiornare periodicamente la chiave di sessione.

6.1.2 Requisiti non funzionali

L'ambiente di esecuzione impone al software di avere le seguenti proprietà:

1. Utilizzo minimo della memoria.

2. Utilizzo minimo della capacità di calcolo.

3. Robustezza contro l'attacco *Man in the Middle*.

6.2 Panoramica

Come detto in precedenza nella Sezione 4.4.1, occorrerebbe avere una chiave concordata per poter scambiare la chiave di sessione mediante l'uso di un cifrario simmetrico, cosa che converrebbe maggiormente dati i vincoli presenti in questo ambiente, ma implicherebbe la necessità di concordare una chiave per poter scambiare una chiave. In pratica, andrebbe a creare un'altra istanza del problema per risolvere il problema stesso. Un ragionamento diverso può essere fatto osservando che questa fase è di vitale importanza per tutte le soluzioni che implementano una comunicazione sicura, i quali necessitano di una chiave per farlo, e considerando che principalmente avviene in fase di *bootstrap*, ovvero in fase di accensione del veicolo, e ad intervalli regolari per aggiornare la chiave, si può pensare di pagare qualcosa in più in merito al tempo e al consumo di risorse. Ragione per cui, la nostra soluzione consiste in un **approccio ibrido** a due fasi:

1. **Fase asimmetrica**: per lo scambio della chiave di sessione.

2. **Fase simmetrica**: per la comunicazione intra-veicolare.

In linea di principio la fase asimmetrica va a rompere il bisogno ricorrente di avere una segreto condiviso per poter scambiarsi informazioni in modo confidenziale.

6.3 Scelte di progetto

Quanto detto finora ci dice quali meccanismi di comunicazione compongono il protocollo ma non abbiamo detto nulla né sulle entità in gioco, né sullo stile architetturale. Sotto questi punti di vista, la soluzione proposta sposa quanto detto sulla gestione delle chiavi, andando a definire in modo preciso i punti in cui è stata mantenuta libertà di scelta. La rete intra-veicolare è composta da uno o più CAN bus (sotto-reti) connessi tra loro attraverso un gateway. Le ECU vengono collegate al proprio CAN bus in base alle funzionalità che devono gestire, quindi all'interno di un'auto si vanno a formare diverse **aree funzionali**. Questo ci suggerisce di inserire un key master all'interno di ogni area funzionale, il quale fornisce la chiave di sessione a tutte le ECU appartenenti alla stessa sotto-rete, così da renderla unica al suo interno. Questo è giustificato dal fatto che una ECU scambia messaggi solo con i dispositivi collegati al medesimo bus. Come prima cosa si è optato per una **collocazione locale** del key master fondamentalmente per i seguenti motivi:

- **Economico**: in quanto si è voluto che tale soluzione fosse realizzabile non solo su veicoli di fascia alta di mercato, i quali offrono ECU con risorse sufficienti per supportare la soluzione che prevede un collocazione in remoto del key master, ma anche su quei veicoli che mettono a disposizioni controllori di base.

- **Stabilità**: optando per l'altra soluzione, la quale prevede che vi sia una collocazione in remoto del key master, sarebbe nata la necessità di instaurare e mantenere una connessione Internet con esso mediante una SIM. L'instabilità risiede proprio nella connessione, poiché la presenza del segnale non può essere garantita dappertutto, o semplicemente allo stesso modo, non può essere garantita una potenza del segnale per una connessione con velocità adeguata. Se si venisse a verificare uno di questi due scenari, la comunicazione con il key master richiederebbe molto più tempo.

- **Carico computazionale**: per mantenere una connessione Internet c'è bisogno che la varie ECU oppure il sistema di infotainment implementino lo stack TCP/IP, il quale richiede uno sforzo computazionale non banale. Inoltre c'è da considerare che non si può fare a meno di avere una connessione affidabile che implica l'uso, a livello

trasporto, del protocollo TCP, che garantisce la consegna effettiva e ordinata dei dati scambianti. È noto che questo protocollo richieda una quantità significativa di risorse per implementare quanto appena detto.

In sintesi collocare il key master localmente all'interno del veicolo è sembrato più adatto ai vincoli che questo panorama impone.
La fase asimmetrica del protocollo entra in gioco nella comunicazione fra una ECU e il key master, implicando che i due debbano disporre di una **chiave pubblica** e una **chiave privata** per realizzarla, tali chiavi devono essere aggiornate ogni qualvolta vi deve essere una comunicazione. Fondamentalmente vi sono due soluzioni per questa fase:

- **Protocollo Difffie Hellman DH**.
- **RSA**.

Considerando il fatto che la chiave deve essere uguale per tutte le ECU, il protocollo DH non è adatto come soluzione in quanto prevede:

- Due partner eseguano un calcolo, nel quale utilizzano una parametro generato casualmente.
- Si scambino il risultato del calcolo eseguito.
- Ognuno di loro utilizza il calcolo dell'altro per generare la chiave di sessione.

Nel nostro caso l'utilizzo del protocollo DH porterebbe alla situazione in cui ogni ECU, utilizzando la comunicazione con il key master il proprio *segreto*, arrivi a stabilire una chiave diversa, cosa che farebbe fallire tutte le comunicazioni successive con le varie ECU. Non è possibile stabilire un segreto comune per tutte le ECU perché ci sarebbe il problema di comunicarlo, dato che è altamente improbabile che ognuna di loro generi in maniera random la stessa sequenza di bit.
Quindi, questa fase prevede l'utilizzo del cifrario RSA che non presenta il problema spiegato sopra, dato che la chiave non viene concordata attraverso lo scambio di calcoli intermedi, ma viene inviata dal key master alla singola ECU in un crittogramma (messaggio cifrato), in questo modo è possibile comunicare la medesima chiave a tutte le ECU collocate

lungo lo stesso CAN bus. Per quanto riguarda la *fase simmetrica* è previsto l'utilizzo del **protocollo TOUCAN**, il quale utilizza MAC a 24 bit per garantire autenticazione e integrità e cifra il payload per garantire la confidenzialità . Il MAC è calcolato mediante **l'algoritmo di Chaskey** [10] che utilizza chiavi a 128 bit, come l'algoritmo **Speck** [6] utilizzato per la cifratura di tutto il frame, il quale produce un immagine di 64 bit. Con il fine di minimizzare il numero di frame scambiati durante questa fase, le chiavi generate da una ECU, per la comunicazione con il key master, hanno una lunghezza pari a 64 bit in modo da poter comunicare la propria chiave pubblica, nel caso del RSA, al interno di due frame. Sebbene sia noto che il cifrario RSA per fornire un buon livello di sicurezza debba essere impiegato con chiavi dell'ordine della migliaia di bit, il nostro è stato uno studio sperimentale di tale cifrario nel contesto automobilistico.
Il key master invece genera chiavi a 128 bit impiegando un numero di frame pari a due per l'invio alla singola ECU. Questi calcoli sono giustificati dalla assunzione che nella comunicazione venga utilizzano il *protocollo Extended CAN* (Capitolo 2) il quale prevede **64 bit di payload** e **29 bit nel campo identifier**, ai quali viene data la semantica SAE J1939 [26]. Quindi se γ è il numero di ECU che fanno parte di una singola sotto-rete, abbiamo che il numero di messaggi richiesti in questa fase è dato dal prodotto $\mathbf{4} \times \boldsymbol{\gamma}$, utilizzando una notazione asintotica possiamo concludere che il numero di messaggi cresce come $\boldsymbol{\Theta(\gamma)}$.

6.4 RSA: funzionamento

Il cifrario RSA fu presentato nel 1979 ed è un cifrario a chiave pubblica che basa la sua sicurezza sul **problema della fattorizzazione**. Nel caso specifico: dati due numeri primi p e q è computazionalmente facile calcolare il prodotto $n = p * q$, ma dato n è computazionalmente difficile risalire a p e q. Precisamente il cifrario richiede i seguenti paramenti:

- **p** e **q** due numeri primi molto grandi.
- Calcolare $\mathbf{n} = \mathbf{p} \times \mathbf{q}$ e la funzione di Eulero $\Phi(n) = (\text{p-1}) \times (\text{q-1})$.

- Scegliere un intero **e** minore di $\Phi(n)$ e primo con esso.
- Calcolare l'intero **d** inverso di **e** modulo $\Phi(n)$ (l'esistenza e l'unicità di d è assicurata per il teorema di Eulero)[1].

Di questi la **chiave pubblica** è la coppia <**n,e**>, mentre la **chiave privata** è **d**.
Ogni messaggio è codificato come una sequenza binaria, che viene trattata come un numero intero m. Per impiegare il cifrario deve risultare $m < n$, il che è sempre possibile dividendo il messaggio in blocchi di al più **t** bit dove [7]:

$$t \leq \lfloor \log_2(n) \rfloor \tag{6.1}$$

Il funzionamento del cifrario è dato dalle seguenti operazioni matematiche:

1. **Cifratura**: il mittente si procura la chiave pubblica del destinatario, la coppia <**n,e**>, e calcola $\mathbf{c} = m^e \mod n$, invia **c**.
2. **Decifrazione**: il destinatario all'arrivo del crittogramma **c** per risalire al messaggio utilizza la propria chiave privata il numero **d** calcolando $\mathbf{m} = c^d \mod n$.

La correttezza del cifrario è data dal seguente teorema.

Teorema 6.4.1 ([7]) *Per un qualunque intero $m < n$ si ha: $(m^e \mod n)^d \mod n = m$, ove n,e,d sono i parametri del cifrario RSA.*

6.5 Struttura della fase asimmetrica

La fase asimmetrica ha luogo in due momenti: i) in fase di bootstrap all'accensione del veicolo e ii) ad intervalli regolari per l'aggiornamento della chiave di sessione. Entrambi sono strutturati allo stesso modo in merito al comportamento dei partecipanti. Nel dettaglio, il key master deve:

- Generare la chiave di sessione, una sola volta all'inizio della fase.
- Ricevere la chiave pubblica di una ECU e cifrare la chiave di sessione.

[1]Per n > 1 e per ogni a primo con n si ha che $a^{\Phi(n)} \equiv 1 \mod n$

- Dividere il crittogramma ottenuto nel payload di due frame CAN.
- Inviare i due frame specificando nel primo che il messaggio è composto da due frame, questo è possibile utilizzando per il campo ID la specifica SAE J1939 [26].

Una ECU deve:

- Calcolare la propria chiave pubblica e privata.
- Comunicare la chiave pubblica al key master.
- Decriptare i due frame CAN ricevuti dal key master i quali contengono la chiave di sessione, controllando che l'ID corrisponda a quello atteso.
- Ricomporre la chiave simmetrica. La specifica SAE J1939 [26] garantisce che venga ricostruita nel modo corretto.

Al fine di evitare errori durante la comunicazione della chiave pubblica da parte delle ECU al key master, ognuna di loro nel campo ID del frame ha un valore diverso, il quale è ordinato in modo strettamente crescente partendo dalla più distante da esso fisicamente. In questo modo, per come funziona l'arbitraggio sul CAN bus (minore è il valore del campo ID maggiore è la priorità), ogni ECU riesce a comunicare correttamente la propria chiave pubblica. Nella comunicazione della chiave di sessione invece il key master inserirà nel campo ID un valore che identifica la ECU a cui è diretto e se stesso secondo la specifica SAE J1939 [26]. In questo modo sarà decifrato solo dalla ECU in grado di farlo.
Inoltre, si assume che l'ID associato key master non possa essere replicato da un'altra ECU. In pratica, ogni ECU ha il proprio ID che non può essere modificato da remoto. Questa assunzione serve a contrastare un attacco di tipo *Man in the Middle*, in quanto nessun'altra ECU può fingere di essere il key master all'interno della sua zona di competenza.

Capitolo 7

Implementazione

Questo tirocinio si è concentrato nello sviluppo della sola parte asimmetrica del protocollo, realizzando un prototipo che traduce in realtà la comunicazione tra una singola ECU e il key master.

7.1 Ambiente

La comunicazione tra la singola ECU e il key master è stata realizzata mediante l'uso di un notebook HP[1] e un Raspberry Pi 2 Model B[2], utilizzando come linguaggio di programmazione il linguaggio C. La comunicazione è realizzata attraverso *socket AF INET* utilizzando il pattern architetturale client-server, facendo impersonare al Raspberry la ECU (quindi il client) e al notebook il key master (quindi il server). Il frame CAN è stato modellato attraverso la *struct*, Listing 7.1, facendo riferimento alla versione Extended CAN, prendendo solo i campi fondamentali: *ID*, *DLC* e *Payload*.

[1]sistema operativo linux mint, 8 GB di RAM e processore intel i5 dual core 2.5 GHz

[2]sistema operativo raspdebian, 1 GB di RAM e processore quad core Cortex A7 900MHz

```
1         /*
2          * rappresenta il campo ID di un frame Extended CAN mediante
3          * l'uso di una bitfield di 29 bit
4          */
5          typedef struct id{
6             unsigned int value : 29;
7         }id;
8
9         /*
10         *  rappresenta il campo DLC di un frame Extended CAN mediante
11         *  l'uso di una bitfield di 4 bit
12         */
13        typedef struct dlc{
14            unsigned int value : 4;
15        }dlc;
16        /*
17         * struttura che rappresenta un frame Extended CAN nei
18         * sui campi piu' importanti
19         */
20        typedef struct can_frame{
21            id ID;
22            dlc DLC;
23            uint8_t payload[8];
24        }can_frame;
25
```

Listing 7.1: Rappresentazione frame Extended CAN

Per la realizzazione di questa fase sono state implementate tutte le funzionalità necessarie all'utilizzo del cifrario RSA spiegate dettagliatamente nella Sezione 7.3.

7.2 Comunicazione

La comunicazione realizza quanto descritto nella Sezione 6.5. Quando il server viene avviato, genera la chiave simmetrica e attende una richiesta di connessione da parte del client. All'avvio, il client genera la propria coppia di chiavi e stabilisce una connessione

con il server. In seguito invia due frame CAN che contengono rispettivamente n ed e che formano la chiave pubblica del cifrario RSA. Accettata la connessione e ricevuta la chiave pubblica, il server cifra la chiave di sessione generata in precedenza (K), dispone il crittogramma ottenuto nel payload di due frame CAN e prosegue con il loro invio; terminato l'invio dei frame attende nuove richieste di connessione. Il client riceve i due frame CAN contenenti la chiave di sessione, ne decifra il payload e ottiene la chiave di sessione corretta, che verrà stampata a video. Il client a questo punto termina. La verifica dell'ID del key master contenuto in entrambi i frame è stata implementata definendo una costante all'interno del programma che ne stabilisce il valore, quindi alla ricezione dei frame il client controlla che il campo ID combaci con tale valore, come mostrato nel frammento di codice sottostante.

```
1    while (!done) {
2        ISNULL(read_frame(sockfd), frame_in, "read frame fail")
3        if (frame_in->ID.value == KEYMASTERID) { //controllo dell'ID
4            //decifro il payload
5            uint8_t *tmp = decipher(frame_in->payload,
6                                    param->n, param->d);
7            //costruisco la chiave di sessione
8            for (int i = 0; i < SIZEPAYLOAD; i++) {
9                symmetric_key[j] = tmp[i];
10               j += 1;
11           }
12           free(frame_in);
13           free(tmp);                    //chiave di sessione
14           if (j == BYTEKEY) done = 1; //ricevuta completamente
15       } else break;
16   }
17
```

Listing 7.2: Ricezione chiave di sessione

Quando la comunicazione termina troviamo per il server un output del tipo mostrato in Figura 7.1, per il client è mostrato in Figura 7.2.

Figura 7.1: Output server

Figura 7.2: Output client

L'aggiornamento della chiave di sessione nel server è stato implementato utilizzando il segnale SIGALRM, che viene mandato al processo allo scadere di un timer impostato in precedenza. All'arrivo del segnale il server rigenera la chiave di sessione ed avvia nuovamente il timer.

7.3 Implementazione dell'RSA

Per utilizzare correttamente il cifrario il numero intero che codifica il messaggio deve risultare minore di **n** (Sezione 6.4). Questo diventa problematico quando questa proprietà si deve garantire in un sistema con più utenti, data la presenza di più chiavi pubbliche. Una soluzione a questa problematica, che è stata adottata nella implementazione, è quella di definire un limite inferiore al valore di n [7]. In questo modo codificando il messaggio con interi minore del limite inferiore prestabilito è possibile comunicarlo sotto forma di crittogramma a tutti gli utenti del sistema; nel nostro caso a tutte le ECU. Il limite fissato nel programma è, per semplicità, il massimo intero rappresentabile a 32 bit. Questo comporta che per poter cifrare i due interi che rappresentano la chiave di sessione, essi siano ridotti modulo il limite inferiore stabilito. Questo implica che i 128 bit generati casualmente diventino 64 bit, poiché i due interi che rappresentano la chiave, dopo la riduzione in modulo, hanno i 32 bit meno significativi a 0; quindi i corrispettivi bit nella chiave di sessione assumeranno il medesimo valore. In una implementazione reale il valore che

stabilisce il limite inferiore può essere fissato in modo che non si verifichi quanto descritto in precedenza, oppure si potrebbe utilizzare una tecnica migliore per rendere i due interi che rappresentano la chiave cifrabili con tutte le chiavi pubbliche delle varie ECU.
Il cifrario RSA, per dare un buon livello di sicurezza, richiede che la chiavi siano dell'ordine della migliaia di bit. D'altra parte per rispettare i vincoli e i requisiti imposti dal dominio applicativo in oggetto, nell'implementazione non è stata seguita tale prerogativa tenendo conto della limitata capacità computazionale delle ECU e assumendo di aggiornare frequentemente la chiave di sessione.

7.3.1 Generazione delle chiavi

Le chiavi sono state generate tenendo conto di quando detto nella Sezione 6.3.
I numeri primi p e q vengono generati nel modo seguente:

- Viene generato un numero random di 32 bit, dato che la chiave deve avere una lunghezza di 64 bit, escludendo quelli pari poiché certamente composti.
- Ne viene testata la primalità mediante il test di **Miller e Rabin**, ripetendolo un numero di volte pari a 30, perché essendo un algoritmo randomizzato in questo modo la probabilità di errore è prossima a 10^{-18}, quindi è altamente improbabile che si venga a generare un numero composto. Inoltre la sua complessità è O(log(N)), dove N è la lunghezza in bit del numero in questione [7].
- Se il test ha avuto esito positivo si procede a generare l'altro[3], altrimenti si ritorna al primo punto.

Generati p e q, si calcola n facendone il prodotto e si controlla che il risultato sia maggiore del limite inferiore prestabilito che nel programma è il massimo intero rappresentabile a 32 bit. In caso di esito negativo si rigenerano i numeri primi e si ripete il controllo su n, invece in caso di esito positivo si procede alla generazione dei parametri mancanti. Il parametro e è calcolato generando un numero random modulo Φ(n) ($e < \Phi$(n)), finché non viene generato un intero primo con Φ(n), ovvero un numero tale per cui il massimo comune

[3]Nella generazione del secondo si controlla che non sia uguale al primo

divisore con Φ(n) risulti essere uno. Per verificare questa proprietà viene calcolato il massimo comune divisore tra il numero generato e Φ(n), attraverso l'algoritmo di Euclide esteso la cui complessità è polinomiale. In questo modo, non solo troviamo il parametro e, ma anche la chiave privata d, dato che l'algoritmo ci restituisce anche l'inverso di e modulo Φ(n).
Alla fine sono stati generati tutti i parametri del cifrario, in un tempo polinomiale.

7.3.2 Cifratura e Decifrazione

Entrambi i processi consistono in un elevamento a potenza in modulo, ragione per cui, in entrambe le fasi, viene eseguito lo stesso algoritmo, ovviamente cambiando i paramenti, l'algoritmo delle **quadrature successive**. Tale algoritmo si basa sullo scrivere l'esponente come somma di potenze di due, operazione immediata utilizzando la rappresentazione binaria, tutte le operazioni vengono eseguite modulo n. Se z è il parametro dell'algoritmo, questo richiede $\Theta(\log(z))$, dunque queste due operazioni richiedono tempo polinomiale.

7.4 Valutazioni e Performance

Il protocollo è stato testato nell'ambiente descritto nella Sezione 7.1 e sono stati presi i tempi per l'esecuzione della fase asimmetrica. Tutti i tempi sono stati presi sul Raspberry descritto nella Sezione 7.1. I tempi d'esecuzione della fase asimmetrica sono riportati nella Tabella 7.1. Da notare che una ECU ha una capacità di calcolo inferiore rispetto a un Raspberry, quindi, con molta probabilità in ambiente reale tali tempi potrebbero essere non adatti.

Iterazione	Tempo (ms)
1	310
2	370
3	320
4	310
5	370
-	tempo medio 336

Tabella 7.1: Tempi fase asimmetrica

Lo studio sperimentale condotto sull'applicabilità del cifrario RSA in ambito automobilistico ha permesso di concludere che il cifrario può essere facilmente adattato a tale contesto. Tuttavia, presenta una limitata robustezza dal punto di vista della sicurezza. Questo perché, per soddisfare i requisiti e date le limitazioni dovute al protocollo di comunicazione adottato nelle reti intra-veicolari (Capitolo 2), sono state usate chiavi a 64 bit il che è in netta contrapposizione con gli standard odierni che prevedono per l'RSA l'utilizzo di chiavi dell'ordine della migliaia di bit. Il nostro è stato uno studio preliminare di un problema tutt'oggi aperto, il cui scopo è stato indicare la strada per una possibile soluzione effettiva.

Capitolo 8

Conclusioni e Lavori Futuri

Il mondo automobilistico offre all'informatica una serie di sfide nel campo della cybersecurity non facili da vincere dato lo stato delle risorse disponibili e la sua giovane età. Come campo si prospetta molto prosperoso sotto tutti i punti di vista, soprattutto risulta essere un campo che necessita di uno studio intenso per dare una soluzione alla sue problematiche principali. In questo tirocinio ci siamo concentrati sulla problematica relativa allo scambio di una chiave di sessione nella rete intra-veicolare. Tale chiave permetterà quindi di avere comunicazioni CAN sicure tra le centraline di un veicolo secondo le specifiche AUTOSAR. In particolare è stato progettato un protocollo per lo scambio della chiave di sessione, nel quale è stato valutato in via sperimentale l'utilizzo del cifrario RSA e di cui è stato implementato un prototipo.

8.1 Lavori futuri

Una prerogativa che lascia spazio a sviluppi futuri è senz'altro migliorare la sicurezza del cifrario utilizzato nella parte asimmetrica del protocollo. Data la dimensione del payload del frame, 64 bit, e il requisito di minimizzare l'uso della banda di rete, sono state utilizzate chiavi pubbliche a 64 bit. Per la crittografia a chiave pubblica di prima generazione sono raccomandate chiavi dell'ordine della migliaia di bit per avere una sicurezza adeguata. Infatti, esiste un algoritmo sub-esponenziale chiamato *index calculus* per calcolare il logaritmo discreto che può essere usato sia per il protocollo DH sia per RSA.

Avendo un numero così ridotto di bit a disposizione, uno spunto interessante sarebbe quello di utilizzare in questa fase la crittografia su curve ellittiche (ECC), la quale a parità di bit di chiave offre una sicurezza molto maggiore dato che gli attacchi noti sono tutti di natura puramente esponenziale. Un algoritmo noto per lo scambio di messaggi cifrati su curve ellittiche, che viene lasciato come spunto, è il **cifrario a chiave pubblica di ELGamal** [7].

Oltre alla sicurezza, un'altra misura importante di cui tener conto quando si confrontano sistemi crittografici diversi è la loro efficienza. Considerando che per chiavi di pari lunghezza il costo computazionale dell'RSA e dei sistemi basati su ECC è confrontabile, il fatto di poter usare chiavi più corte a parità di sicurezza assicura vantaggi anche in termini di efficienza alla crittografia su ellittiche: elaborazioni più veloci, implementazioni in hardware più compatte, consumi energetici contenuti e quindi minore dissipazione di calore, tutti vantaggi che rendono l'uso di ECC particolarmente adatto ai dispositivi embedded.

Ringraziamenti

Ebbene si sono arrivato all'atto finale di questi tre anni ancora stento a crederci. Come prima cosa mi sento di ringraziare i tutori aziendali Gianpiero e Ilaria che non mi hanno fatto mancare niente in questa esperienza al CNR, sempre gentili e disponibili ad ogni mia necessità. Un caloroso ringraziamento va anche alla mia tutrice interna, la prof.ssa Anna Bernasconi, con la quale ho sostenuto l'esame che mi ha spinto a seguire questa strada per la prova finale, una persona cordiale, sempre disponibile, sempre a disposizione degli studenti. Colgo l'occasione per ringraziare tutti i professori che hanno fatto parte del mio percorso accademico, ognuno ha contributo oltre che nella formazione professionale anche nella persona che sono diventato. Se sono arrivato qui oggi parte del merito è sicuramente dei miei compagni di viaggio Pietro, Iulian e Jacopo, che mi hanno offerto un sostanzioso sostegno morale, oltre quello didattico, sopportandomi nei momenti di panico per i vari esami sostenuti. Assieme a loro ho vissuto tre anni ricchi di emozioni, che porterò per sempre nel mio cuore. Un grazie va tutti i miei amici che mi sono stati accanto anche a distanza, in particolare a quelli di una vita Adriano, Mario, Lorenzo e Gerardo sempre pronti a darmi conforto in qualsiasi momento. Ora passo a ringraziare la colonna portante della mia vita: la famiglia. Inizio con i miei zii Vito e Orietta che considero come secondi genitori e a cui voglio un bene dell'anima, ai loro figli, nonché miei cugini, Giuseppe e Maria con i quali siamo praticamente fratello e sorella. Poi tocca a nonna Palma, per gli amici nonna Pà, per la quale non esistono parole per ringraziala per tutto quello che ha fatto per me. Esempio di come la vita va vissuta, di quali valori contano davvero, un guerriero per definizione che, nonostante l'età continua, a farsi in quattro per i suoi nipoti. Grazie a te roccia della nostra famiglia per tutto quello che fai e che hai fatto, sei l'uomo che vorrei diventare da grande, altre mille righe non basterebbero per ringraziarti,

caro nonno Giuseppe. Un pensiero va anche ai miei bisnonni Maria e Giovanni, dei quali sento la presenza costantemente. Ringrazio anche nonno Carlo e nonna Filomena per il sostegno ricevuto. Loro che mi sopportano da quando ero piccolo sempre attente e vigili nella mia vita, sono le mie sorelle Milena e Pamela con le quali sento di avere un legame inscindibile. Un grazie va anche a mio cognato Daniele sempre pronto a indicarmi la strada migliore da intraprendere, e alla piccola Elisa, la mia nipotina, della quale sono perdutamente innamorato. Se mi chiamano"pioppino"un motivo ci sarà, io e te alla fine ci siamo sempre detti poco ma tra di noi c'è quel legame unico che lega un padre a un figlio, grazie non solo per avermi dato la possibilità di inseguire i miei sogni ma anche perché ci sei sempre stato e ci sarai per sempre; ti voglio bene papà. Come ultima persona voglio ringraziare lei, la donna della mia vita, la persona che mi capisce più di tutti anche solo ascoltando il mio tono di voce, sei motivo di orgoglio mamma, grazie per essermi sempre stata vicino e per avermi fatto da guida quando dovevo prendere in mano la mia vita, sei simbolo di tenacia e forza di volontà; ti voglio bene. Alla fine vorrei ringraziare il sottoscritto che nonostante gli errori commessi, ha avuto la forza di rialzarsi e mettersi in gioco lottando fino a raggiungere il suo primo obiettivo.

Bibliografia

[1] Microsoft. *X.509 Public Key Certificates*. 5/31/2018. URL: https://docs.microsoft.com/en-us/windows/win32/seccertenroll/about-x-509-public-key-certificates.

[2] Steve Corrigan. *Introduction to the Controller Area Network (CAN)*. 2002.

[3] Gia Vo, Richard Lai e Mohit Garg. "Building Automotive Software Component within the AutoSAR Environment - A Case Study". In: ago. 2009. DOI: 10.1109/QSIC.2009.34.

[4] Florian Hartwich. *CAN with Flexible Data-Rate*. 2012.

[5] C. Reuber O.Hartkopp e R.Schilling. "MaCAN-message authenticated CAN". In: a cura di Proc. 10th Int. Conf. Embedded Security in Cars (ESCAR). 2012.

[6] Ray Beaulieu et al. *The SIMON and SPECK Families of Lightweight Block Ciphers*. Cryptology ePrint Archive, Report 2013/404. https://eprint.iacr.org/2013/404. 2013.

[7] P. Ferragina F. Luccio A. Bernasconi. *Elementi Di Crittografia*. University-Press, 2014.

[8] Alessandro Bruni et al. "Formal Security Analysis of the MaCAN Protocol". In: *Integrated Formal Methods*. A cura di Elvira Albert e Emil Sekerinski. Cham: Springer International Publishing, 2014, pp. 241–255. ISBN: 978-3-319-10181-1.

[9] Ryo Kurachi et al. "CaCAN-centralized authentication system in CAN (controller area network)". In: *14th Int. Conf. on Embedded Security in Cars (ESCAR 2014)*. 2014.

[10] Nicky Mouha et al. “Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers”. In: *Selected Areas in Cryptography – SAC 2014*. A cura di Antoine Joux e Amr Youssef. Cham: Springer International Publishing, 2014, pp. 306–323.

[11] International Organization for Standardization. *Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling*. https://www.iso.org/standard/63648.html. 2015.

[12] Silverio Martınez-Fernández et al. “A Survey on the Benefits and Drawbacks of AUTOSAR”. In: *Proceedings of the First International Workshop on Automotive Software Architecture*. WASA ’15. Montreal, QC, Canada: ACM, 2015, pp. 19–26. ISBN: 978-1-4503-3444-0. DOI: 10.1145/2752489.2752493. URL: http://doi.acm.org/10.1145/2752489.2752493.

[13] Andreas Mueller e Timo Lothspeich. “Plug-and-secure communication for CAN”. In: (2015).

[14] ISO 15765-2:2016. *Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services*. 2016. URL: https://www.iso.org/standard/66574.html.

[15] Andreea-Ina Radu e Flavio D. Garcia. “LeiA: A Lightweight Authentication Protocol for CAN”. In: *Computer Security – ESORICS 2016*. A cura di Ioannis Askoxylakis et al. Cham: Springer International Publishing, 2016, pp. 283–300. ISBN: 978-3-319-45741-3.

[16] Alfredo Imparato et al. “A Comparative Study of Static Analysis Tools for AUTOSAR Automotive Software Components Development”. In: ott. 2017, pp. 65–68. DOI: 10.1109/ISSREW.2017.21.

[17] Autosar. *Requirements on Crypto Stack*. 2018.

[18] Autosar. *Specification of Crypto Driver*. 2018.

[19] Autosar. *Specification of Crypto Interface*. 2018.

[20] Autosar. *Specification of Crypto Service Manager*. 2018.

[21] Autosar. *Specification of Key Manager*. 2018.

[22] Autosar. *Specification of Secure Onboard Communication*. 2018.

[23] Giampaolo Bella et al. "TOUCAN: A proTocol tO secUre Controller Area Network". In: *Proceedings of the ACM Workshop on Automotive Cybersecurity*. AutoSec '19. Richardson, Texas, USA: ACM, 2019, pp. 3–8. ISBN: 978-1-4503-6180-4. DOI: 10.1145/3309171.3309175. URL: http://doi.acm.org/10.1145/3309171.3309175.

[24] Pietro Biondi et al. "Implementing CAN bus security by TOUCAN". In: *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Mobihoc 2019, Catania, Italy, July 2-5, 2019*. 2019, pp. 399–400. DOI: 10.1145/3323679.3326614. URL: https://doi.org/10.1145/3323679.3326614.

[25] URL: https://www.kvaser.com/about-can/the-can-protocol/can-error-handling/.

[26] Society of Automotive Engineers. *"SAE J1939 Standards Collection on the Web"*. URL: https://web.archive.org/web/20070807004249/http://www.sae.org/standardsdev/groundvehicle/j1939a.htm.

[27] *automotive.wiki*. URL: https://automotive.wiki/index.php/Basic_Software_Module.

[28] Autosar. *Layered Software Architecture*.

[29] Federal Information Processing e Announcing The. *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*.