POLITECNICO DI TORINO

Laurea Triennale in Ingegneria Informatica

Relazione sul tirocinio curriculare

EXT-TAURUM P2T Sviluppo di un'architettura CAN sicura in ambiente OSEK automotive

Tutors

Stefano DI CARLO

Franco OBERTI

Fulvio VALENZA

Tirocinante

Matteo PEDONE

Marzo-Giugno 2022

Indice

Capitolo 1

Introduzione

1.1 Contesto lavorativo

Il tirocinio non è stato svolto in un'azienda ma in un laboratorio di ricerca del *DAUIN (Dipartimento di Automatica ed Informatica)* del Politecnico di Torino con sede in *Corso Duca Degli Abbruzzi, 24 Torino.*

1.2 Progetto

L'industria automobilistica non fa più affidamento su sistemi meccanici puri ma beneficia invece di unità di controllo elettroniche (ECU) avanzate al fine di fornire funzionalità nuove e complesse nello sforzo di passare ad auto completamente connesse. Tuttavia, le auto connesse forniscono un pericoloso terreno di gioco per gli hacker. I veicoli stanno diventando sempre più vulnerabili agli attacchi informatici poiché sono dotati di funzionalità e sistemi di controllo più connessi. Questa situazione può esporre asset strategici nella value chain automobilistica. In questo scenario, il Controller Area Network (CAN) è il protocollo di comunicazione più utilizzato nel settore automobilistico. Tuttavia, questo protocollo manca di primitive di crittografia e autenticazione. Di conseguenza, qualsiasi nodo dannoso può causare incidenti catastrofici e perdite finanziarie.

Lo scopo di questo pogetto è la prototipazione di EXT-TAURUM P2T una nuova architettura CAN-FD sicura a basso costo per il dominio automotive che implementa la comunicazione sicura tra ECU, una nuova strategia di provisioning delle chiavi, gestione intelligente del throughput e meccanismi di firma hardware. L'architettura è stata definita a livello concettuale nell'ambito di una collaborazione tra Politecnico di Torino e Punch Torino e si vuole passare a una sua prototipazione in ambiente OSEK.

1.3 Dettagli tecnici

Il tirocinio curriculare consiste in un modulo da 10 CFU (Crediti Formativi Universitari), che corrispondono ad un totale di 250 ore. La modalità di lavoro è di tipo part-time e mista: è stata svolta parzialmente presso un laboratorio di ricerca del *DAUIN (Dipartimento di Automatica ed Informatica)* del Politecnico di Torino e parzialmente da remoto. La data di inizio è il 1 Mrazo 2022, e la data di fine è il 10 Giugno 2022.

Capitolo 2

Descrizione attività svolte

2.1 Strumenti utilizzati

2.1.1 Sistema operativo OSEK

Il numero di ECU (electronic control unit) incorporati nelle automobili è cresciuto rapidamente negli ultimi 20 anni. Oggi, alcune automobili contengono oltre 70 ECU.

Gli investigatori hanno scoperto che è stata spesa una grande quantità di sforzi nello sviluppo e software di debug per il sistema operativo (OS), comunicazione e gestione di rete e input/output (I/O) a scapito dello sviluppo dell'applicazione. Da questa indagine sono cresciuti i consorzi OSEK e VDX, che si sono fusi per sviluppare l'OSEK/VDX open standard per i sistemi operativi.

Sebbene gli standard OSEK/VDX siano stati originariamente sviluppati per l'industria automobilistica, le specifiche risultanti descrivono un piccolo sistema operativo in tempo reale ideale per la maggior parte dei sistemi embedded che sono staticamente definiti: il caricamento dinamico delle attività non è supportato. Questi standard sono ora candidati alla standardizzazione da parte dell'International Standards Organization (ISO) con il numero della norma ISO 17356.

2.1.2 Erika Virtual Machine Power PC

L'installazione di un ambiente completo di sviluppo e debug per una scheda embedded comporta sempre molto lavoro nell'installazione di compilatori, debugger, ambienti di sviluppo, makefile e così via. Erika Virtual Machine Power PC ha lo scopo di fornire una soluzione rapida per tutti questi problemi, fornendo una piattaforma Linux con tutto il software preinstallato e pronto per funzionare.

Nello specifico è stata scelta la versione PowerPC di Erika in quanto viene utilizzata in ambiente automotive e contiene un trace integrato per consentire il debug del codice.

In altre parole, questa macchina virtuale fornisce un ambiente virtuale completo in cui è possibile:

- Modificare l'applicazione OSEK/VDX utilizzando il kernel OSEK/VDX open source ERIKA;
- Compila la tua applicazione usando i compilatori open-source preinstallati;
- Schede target di programma collegate al PC tramite una porta USB indirizzata al Macchina virtuale.

In aggiunta a questo, questa macchina virtuale contiene una versione pre-costruita di ERIKA Enterprise che funziona come domU su Xen Hypervisor.

2.1.3 Lauterbach Trace32

TRACE32 di Lauterbach è un insieme di strumenti di sviluppo modulari per microprocessori, che offre un ambiente di debug integrato per progetti embedded.

TRACE32 permette il debug contemporaneo di più CPU nel progetto, con stream di trace in modalità "mixed". Gli utenti possono osservare i risultati interlacciati in una stessa finestra trace, con una base tempi a livello di sistema che aiuta ad allineare i flussi trace. Una logica di trigger estesa permette il cross-trigger fra le logiche di trace delle CPU, rendendo ancora più facile il debug delle interdipendenze fra i processori.

2.2 Formazione

2.2.1 CAN – Controller Area Network

CAN (Controller Area Network) è un bus seriale di comunicazione dati progettato per applicazioni real-time. Nacque originariamente per l'industria automobilistica, ma si è diffuso presto nell'automazione industriale per le sue caratteristiche di robustezza ed affidabilità.

CAN è stato sviluppato da Robert Bosch nel lontano 1986 su richiesta della Mercedes. L'esigenza di far comunicare i molti dispositivi elettronici presenti all'interno delle automobili (Antilock Braking System, Traction Control, Air Conditioning Control, chiusura centralizzata, sono solo alcuni esempi) e la complessità di questi, avrebbe portato ad un aumento insostenibile di collegamenti dedicati ed una duplicazione dei sensori necessari a piu' dispositivi, con conseguente aumento dei costi di produzione e soprattutto notevole ingombro fisico. Per questi motivi è nato CAN, il quale consente a controllori, sensori e attuatori di comunicare l'uno con l'altro ad una velocità fino a 1Mbit/sec, offrendo anche:

- bassi costi di progettazione e implementazione
- funzionamento in ambienti ostili
- facilità di configurazione e modifica
- rilevamento automatico degli errori di trasmissione

Il protocollo CAN si colloca all'interno della pila ISO/OSI nei livelli 1 fisico e 2 data link, lasciando totale libertà per il livello 7 applicazione.

I pacchetti trasmessi da una stazione CAN non contengono indirizzi di alcun genere, al loro posto troviamo un identificatore del contenuto del messaggio unico sull'intera rete. Un nodo ricevitore può così verificare il contenuto del messaggio e filtrare i soli pacchetti a cui è interessato, ignorando gli altri. Questo modo di operare è detto Multicast.

La caratteristica content-oriented del CAN permette un alto grado di flessibilità e modularità del sistema, consentendo che nuove stazioni che sono solo ricevitori e che hanno bisogno dei soli dati esistenti possano essere aggiunte senza alcuna modifica nè all'hardware nè al software.

Per garantire la comunicazione real-time il CAN ricorre ad un metodo di accesso al mezzo CSMA/CD modificato. Due o piu' stazioni che iniziano a trasmettere competono per l'accesso al bus sul loro valore di priorità determinato proprio dall'identificatore. Quello con il valore numerico più basso vince la competizione per il canale.

Ogni conflitto viene risolto in accordo con il meccanismo del wired-and, secondo il quale lo stato dominante (0 logico) sovrascrive lo stato recessive (1 logico). I

nodi perdenti diventano subito ricevitori e non tenteranno una ritrasmissione non prima che il bus sia diventato libero. Questa politica garantisce il determinismo dell'accesso al bus, e l'assenza di periodi di inattività del canale.

2.2.2 Crittografia chiave simmetrica

La crittografia a chiave simmetrica è la più antica forma conosciuta (il cifrario di Cesare è un cifrario a chiave simmetrica). Rappresenta un metodo semplice per cifrare testo in chiaro dove la chiave di crittazione è la stessa chiave di decrittazione, rendendo l'algoritmo di cifratura molto performante e semplice da implementare. Tuttavia presuppone che le due parti siano già in possesso delle chiavi, richiesta che non rende possibile uno scambio di chiavi con questo genere di algoritmi. Lo scambio avviene attraverso algoritmi a chiave asimmetrica o pubblica, generalmente più complessi sia da implementare che da eseguire, ma che permettono questo scambio in modo sicuro. Dopodiché la comunicazione verrà crittata usando solo algoritmi a chiave simmetrica per garantire una comunicazione sicura, ma veloce.

2.2.3 MAC - Message Authentication Code

In crittografia un message authentication code (*MAC*) è un piccolo blocco di dati utilizzato per garantire l'autenticazione e integrità di un messaggio digitale, generato secondo un meccanismo di crittografia simmetrica: un algoritmo MAC accetta in ingresso una chiave segreta e un messaggio da autenticare di lunghezza arbitraria, e restituisce un MAC (alle volte chiamato anche tag). In ricezione il destinatario opererà in maniera identica sul messaggio pervenuto in chiaro ricalcolando il MAC con lo stesso algoritmo e la stessa chiave: se i due MAC coincidono si ha autenticazione e integrità del messaggio inviato.

Il valore MAC protegge dunque sia l'integrità dei dati del messaggio sia la sua autenticità permettendo al destinatario dello stesso (che deve anch'egli possedere la chiave segreta) di rilevare qualsiasi modifica al messaggio.

Come si desume quindi dalla descrizione, l'algoritmo MAC protegge solo da integrità dei dati, autenticità del mittente del messaggio, ma non dalla confidenzialità delle informazioni contenute nello stesso. Può però essere utilizzato insieme a un algoritmo di crittografia simmetrica che crittografa tutto il messaggio **M** con una chiave **K'**, verrà effettuato quindi il MAC del messaggio **M** crittografato con **K'** usando la chiave **K**. Verrà poi crittografato con un algoritmo di crittografia simmetrica (usando la chiave **K'**) il messaggio **M** e il suo MAC preparato con chiave **K'**.

Le due chiavi **K'** (usata nella crittografia simmetrica) e **K** (usata nella creazione del MAC) non dovranno però essere uguali.

I MAC differiscono dalle firme digitali in quanto sono sia generati che verificati utilizzando la stessa chiave segreta. Questo implica che il mittente e il destinatario del messaggio devono scambiarsi la chiave prima di iniziare le comunicazioni, come nel caso della crittografia simmetrica.

Per questa ragione i MAC non forniscono la proprietà del non ripudio offerta dalle firme digitali: qualunque utente che può verificare un MAC è anche capace di generare MAC per altri messaggi. Invece una firma digitale è generata utilizzando la chiave privata di una coppia di chiavi (crittografia asimmetrica). Dato che la chiave privata è nota solo al suo possessore, una firma digitale prova che un documento è stato firmato esattamente dal suo proprietario e da nessun altro. Ecco che le firme digitali offrono la proprietà del non ripudio.

Esempio:

Il mittente deve inviare un messaggio a un destinatario. Prima di effettuare l'operazione si accorda col destinatario su una chiave segreta.

Stabilita la chiave, utilizza una funzione di MAC per calcolare il MAC del messaggio e infine invia entrambi al destinatario. Questi calcola il MAC del messaggio ricevuto e poi lo confronta con il MAC allegato al messaggio stesso: se coincidono, il messaggio è integro, altrimenti qualcosa è andato storto. (Fig.2.1)

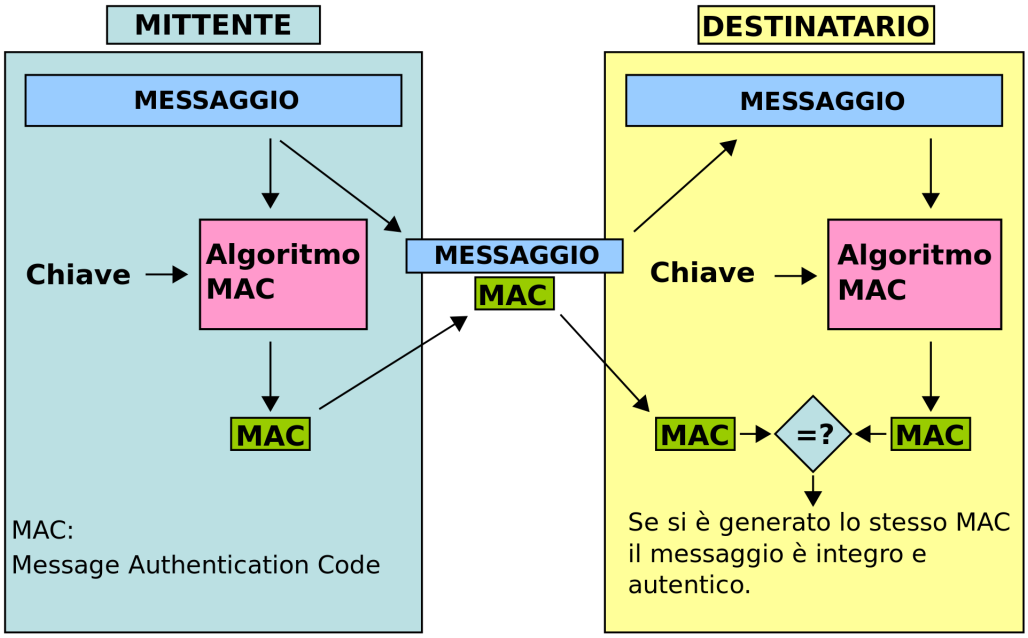

Figura 2.1: MAC

2.2.4 Cifrario a blocchi

Uno dei cifrari a chiave simmetrica più utilizzati è il cifrario a blocchi. Si partiziona il messaggio m in blocchi da k bit, ed ognuno di loro è cifrato in maniera indipendente mediante una tabella di conversione. Si noti che questa tabella (che deve essere conosciuta sia da A che da B), ha un numero di entry pari a 2^k . Valori accettabili di k, che garantiscano una certa sicurezza, sono dell'ordine dei 64 bit. Un attacco brute-force dovrebbe provare tutte le $2^k!$ possibili combinazioni di tabelle per decifrare il cifrario. Risulta molto oneroso tenere in memoria tabelle così grandi, per questo si attua il seguente schema (Fig.2.2), qui spiegato per k = 64 bit.

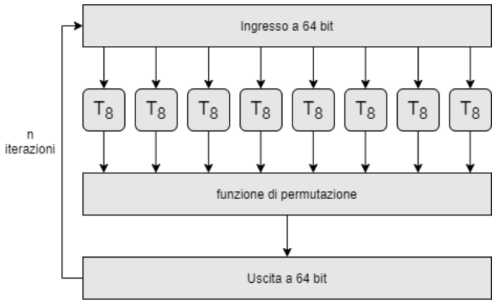

Figura 2.2: Esempio di cifrario a blocchi

Si creano 8 tabelle di cifratura da 8 bit ciascuno, e l'ingresso viene partizionato ulteriormente. Ogni partizione dell'ingresso viene mandato ad una singola tabella, cifrato, e mandato ad una funzione di permutazione. Il processo viene iterato n volte, così da simulare una cifratura a blocchi di 64 bit, per via dell'indipendenza statistica. La chiave simmetrica Ks sarà costituita dalle 8 tabelle di cifratura, dato che di solito la funzione di permutazione è predeterminata. Uno degli algoritmi più usati è AES(Advanced encryption standard), che usa blocchi di 128 bit con chiavi a 128, 192 0 256 bit. La maggiore dimensione delle chiavi è indice di una maggiore sicurezza.

2.2.5 Advanced Encryption Standard

Il 2 gennaio 1997 l'americano NIST (National Institute of Standards and Technology) lanciò un concorso per un nuovo cifrario simmetrico (a chiave segreta) denominato Advanced Encryption System che potesse prendere il posto del DES e avesse una sicurezza almeno pari al Triplo DES. Dopo aver esaminato molti cifrari proposti da crittologi di tutto il mondo il 2 ottobre 2000 il NIST annunciò di aver scelto il cifrario Rijndael progettato da due crittologi belgi Joan Daemen e Vincent Rijmen; l'AES-Rijndael fu certificato nel 2001 come nuovo standard di cifratura. Il nome Rijndael è una sintesi dei nomi dei suoi inventori.

In crittografia, l'**Advanced Encryption Standard** (AES), è un algoritmo di cifratura a blocchi a chiave simmetrica.

Nell'AES il blocco è di dimensione fissa (128 bit) e la chiave può essere di 128, 192 o 256 bit.

AES opera utilizzando matrici di 4×4 byte chiamate stati (states). Quando l'algoritmo ha blocchi di 128 bit in input, la matrice State ha 4 righe e 4 colonne; se il numero di blocchi in input diventa di 32 bit più lungo, viene aggiunta una colonna allo State, e così via fino a 256 bit. In pratica, si divide il numero di bit del blocco in input per 32 e il quoziente specifica il numero di colonne.

C'è un passaggio iniziale:

1. **AddRoundKey** – Ogni byte della tabella viene combinato con la chiave di sessione, la chiave di sessione viene calcolata dal gestore delle chiavi.

Successivamente per cifrare sono previsti diversi round o cicli di processamento: ogni round (fase) dell'AES (eccetto l'ultimo) consiste dei seguenti quattro passaggi:

1. **SubBytes** – Sostituzione non lineare di tutti i byte che vengono rimpiazzati secondo una specifica tabella.
2. **ShiftRows** – Spostamento dei byte di un certo numero di posizioni dipendente dalla riga di appartenenza.
3. **MixColumns** – Combinazione dei byte con un'operazione lineare, i byte vengono trattati una colonna per volta.
4. **AddRoundKey** – Ogni byte della tabella viene combinato con la chiave di sessione, la chiave di sessione viene calcolata dal gestore delle chiavi.

Il numero di round o cicli di processamento/elaborazione crittografica dei quattro passaggi precedenti è 10 con l'ultimo round che salta il passaggio MixColumns. A seguito la descrizione di ogni singolo passaggio.

La fase di decifratura non è identica a quella di cifratura dal momento che gli step sono eseguiti in ordine inverso. Tuttavia, si può definire un cifrario inverso equivalente ai passi dell'algoritmo usato per la cifratura, usando la funzione inversa a ogni step e un differente key schedule.

2.3 Analisi del codice di start-up dell'OS

2.3.1 Core OS

Per poter implementare le funzioni di criptaggio e decriptaggio dei messaggi è necessario conoscere in modo accurato il codice di start-up del sistema operativo OSEK. In particolare è utile analizzarlo per individuare la locazione ottimale in cui aggiungere il relativo codice.

OSEK OS è un sistema operativo attivato da eventi (event-triggered). Ciò fornisce un'elevata flessibilità nella progettazione e manutenzione di sistemi basati su AUTOSAR. L'attivazione di eventi offre libertà per la selezione degli stessi per guidare la pianificazione in fase di esecuzione.

In particolare OSEK OS fornisce le seguenti funzionalità per supportare i concetti in AUTOSAR:

- Pianificazione fissa basata su priorità
- Strutture per la gestione degli interrupt
- Interruzioni solo con priorità maggiore rispetto alle attività
- Protezione contro l'uso errato dei servizi del sistema operativo
- Un'interfaccia di avvio tramite *StartOs* (e *StartupHook*).
- Un'interfaccia di spegnimento tramite *Shutdownos()* e *ShutdownHook*.

2.3.2 Applicazione OS

Un sistema operativo real-time deve essere in grado di supportare una raccolta di oggetti del sistema operativo (attività, interruzioni, allarmi, hook, ecc.) che formano un'unità funzionale coesa. Questa raccolta di oggetti è denominata **applicazione OS**.

Il sistema operativo è responsabile della pianificazione delle risorse di elaborazione disponibili tra le applicazioni del sistema operativo che condividono il processore. Se vengono utilizzate le applicazioni OS, tutte le tabelle Task, ISRS, Risorse, Contatori, Allarmi e Pianificazione devono appartenere a un'applicazione OS. Tutti gli oggetti che appartengono alla stessa applicazione OS hanno accesso l'uno all'altro. L'accesso da altre applicazioni OS può essere concesso durante la configurazione. Per accesso si intende consentire l'utilizzo di questi oggetti all'interno dei servizi API.

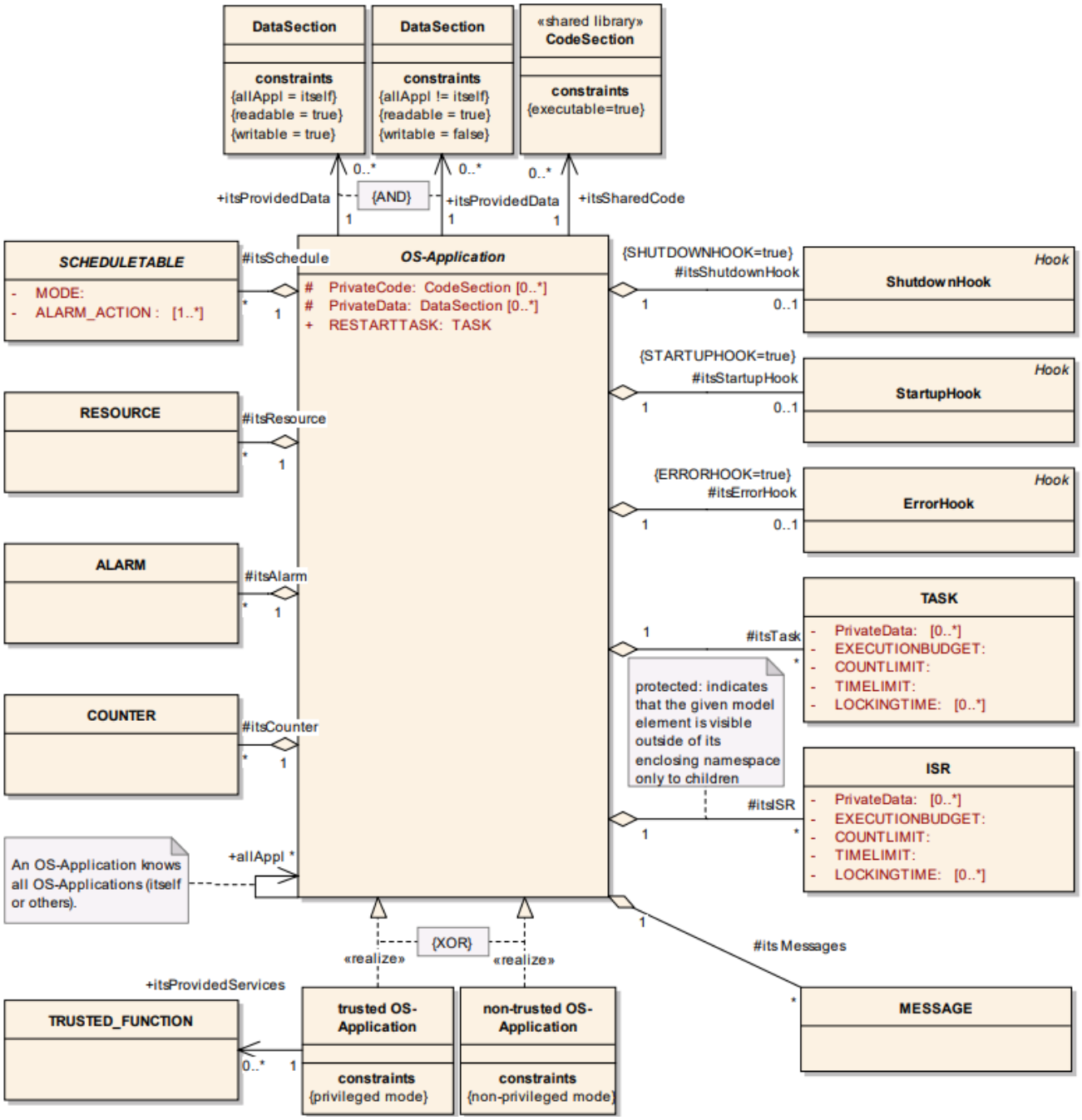

Figura 2.3: Modello di Applicazione OS

Esistono due classi di applicazioni OS:

1. *Le applicazioni Trusted-OS* possono essere eseguite con funzionalità di monitoraggio o protezione disabilitate in fase di esecuzione. Potrebbero avere accesso illimitato alla memoria. l'API del sistema operativo e non è necessario che il loro comportamento di temporizzazione venga applicato in fase di esecuzione. Possono essere eseguiti in modalità privilegiata se supportati dal processore.

2. *Le applicazioni Non-Trusted OS* non possono essere eseguite con le funzioni di monitoraggio o protezione disabilitate in fase di esecuzione. Hanno accesso limitato alla memoria, accesso limitato all'API del sistema operativo e il loro comportamento temporale viene applicato in fase di esecuzione. Non possono essere eseguiti in modalità privilegiata se supportati dal processore

Si presume che il sistema operativo stesso sia affidabile.

StartupHook

StartupHook viene chiamato dal componente RTA-OSEK durante la chiamata *StartOS* dopo che il kernel è stato inizializzato, ma prima che lo scheduler sia in esecuzione. La Figura 2.4 mostra l'esecuzione dello Startup Hook relativo all'inizializzazione del componente RTA-OSEK.

Figura 2.4: Uso di StartupHook

StartupHook viene spesso utilizzato per l'inizializzazione di OSEK COM o inizializzazione dell'hardware di destinazione (configurazione e inizializzazione delle fonti di interrupt, per esempio).

Allarmi e Contatori

A differenza di altri sistemi operativi in tempo reale, OSEK/VDX non ha un concetto di timer; viene invece definito un allarme che copre le funzioni di un timer e la necessità unica di un sistema di controllo integrato per intraprendere un'azione in base al verificarsi di una serie di eventi. Questo si ottiene creando un oggetto contatore che viene incrementato ogni volta che si verifica un evento. L'**allarme** viene attivato quando il **contatore** corrispondente raggiunge un valore preimpostato.

Risorse

La maggior parte dei sistemi operativi multitasking/multiprocessing fornisce un metodo per condividere una risorsa tra compiti o processi. I metodi tipici sono semafori e mutex.

Interrupt

In molte applicazioni integrate, gli interrupt sono interfacce critiche con eventi asincroni esterni. Esempi di interruzioni che si possono trovare nei sistemi embedded sono orologi in tempo reale, sensori che inviano un flusso di impulsi e interruzioni basate su un'azione dell'utente.

Task

Una delle parti fondamentali del sistema operativo e che sarà molto utile al fine del progetto descritto in questa relazione sono le tasks. Le **Tasks** all'interno del sistema operativo OSEK/VDX hanno una serie di attributi che influiscono sulle operazioni del sistema e la dimensione del codice.

Una task è base o estesa, ha una priorità definita staticamente, potrebbe essere o meno preventiva, e potrebbe essere in grado di sospendere la sua esecuzione in attesa di un evento.

La combinazione di questi e altri attributi crea una classe di conformità, come definito nella specifica OSEK/VDX.

Una ***task base*** viene eseguita fino al completamento a meno che non venga anticipata da un'attività con priorità più alta o da un'interruzione (se abilitato). La task può disabilitare la prelazione e gli interrupt. Le task con priorità minore sono inibite mentre vengono eseguite task base con priorità più alta;

Il sistema operativo OSEK/VDX non consente la pianificazione round-robin delle attività con lo stesso livello di priorità, come si trova in alcuni sistemi più grandi. Per consentire la suddivisione del tempo la pianificazione renderebbe impossibile prevedere le prestazioni del sistema, il che è indesiderabile in un ambiente critico per la sicurezza in tempo reale. Le attività di base possono esistere in uno dei tre stati: SOSPESO, PRONTO e IN ESECUZIONE.

Transizioni tra gli stati si verificano in base a quattro possibili eventi: attivazione, avvio, prelazione e terminazione. Il diagramma di transizione dello stato per un'attività di base è mostrato in Figura 2.5

Poiché OSEK/VDX è un sistema operativo definito staticamente, tutte le tasks devono essere definite in fase di compilazione.

Quando il sistema operativo viene avviato da *StartOS()*, viene avviata una task di base nello stato SOSPESO o PRONTO, a seconda della configurazione nel file OIL.

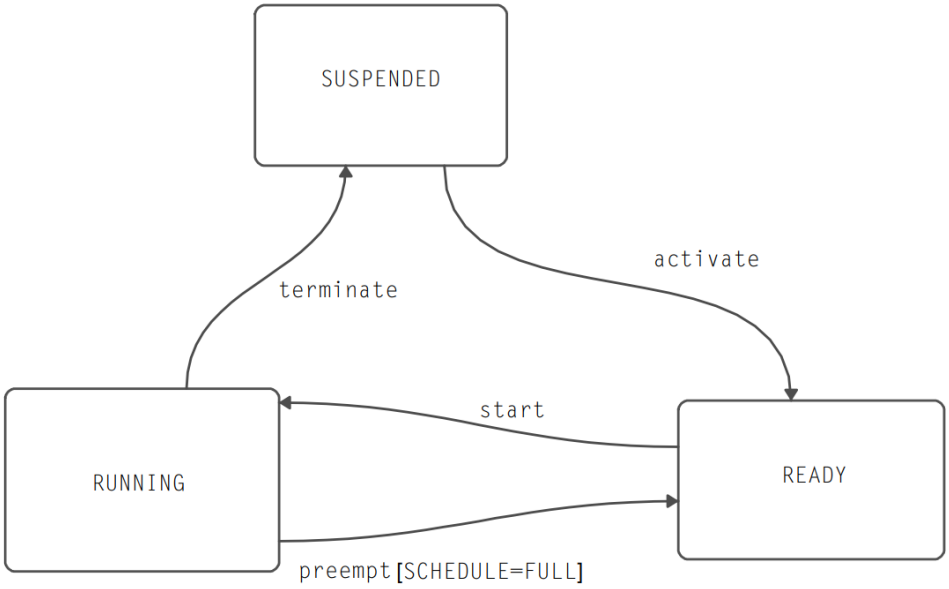

Figura 2.5: Diagramma di transizione dello stato della task base

Se una task è definita come AUTOSTART task parte nello stato PRONTO; in caso contrario, si avvia nello stato SOSPESO. Il vantaggio di una task base è l'utilizzo della quantità minima di risorse necessarie per supportarla. Perché una task base inibisce non solo le attività a priorità più bassa, ma anche altre attività a la stessa priorità, la quantità di spazio di stack richiesta è ridotta al minimo. Di solito le tasks base sono utilizzate nei casi in cui non è richiesta la sincronizzazione tra processi.

Le ***task estese*** sono simili alle tasks base tranne per il fatto che hanno uno stato aggiuntivo nel loro diagramma di transizione dello stato: WAITING. Hanno anche due eventi aggiuntivi: wait e trigger. La figura 2.6 mostra il diagramma di transizione dello stato per compiti estesi. Nello stato WAITING, la task estesa attende che si verifichi un evento e continua dal punto in cui è entrato nello stato. Mentre si trova nello stato WAITING, l'attività consuma ancora risorse come lo spazio dello stack. Poiché le attività estese possono funzionare fuori sequenza, non possono utilizzare stack singoli, come fanno le task base. Di conseguenza, le risorse richieste sono maggiori.

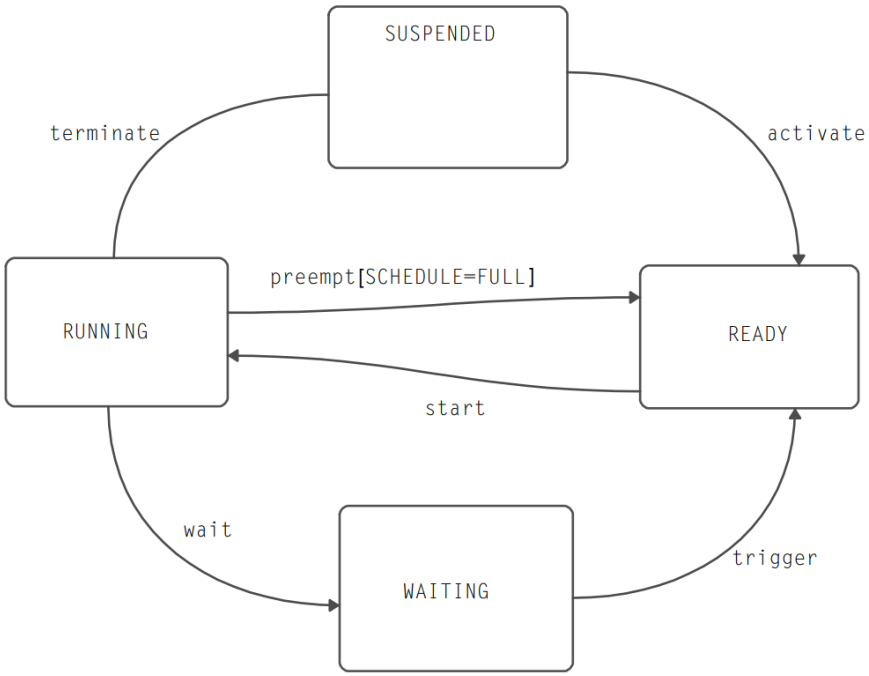

Figura 2.6: Diagramma di transizione dello stato della task estesa

La definizione se una task è di base o estesa viene effettuata indirettamente nel file di configurazione OIL. All'interno della struttura di definizione dell'oggetto TASK è presente l'attributo opzionale EVENT, che se esiste, definisce una task estesa. Altrimenti, è una task base. Alcune implementazioni potrebbero avere un attributo facoltativo impostato sul tipo di task, ma questo non è richiesto dalle specifiche OSEK/VDX.

2.4 Encrypt e Decrypt

Dopo aver esaminato i vari componenti dell'OS e in particolare aver osservato le tasks, è bene sottolineare come si è studiato il modo in cui usarle per lo scopo del tirocinio.

Ogni task viene chiamata dal sistema operativo in base a un riferimento temporale. Si può dedurre quindi che ci saranno task che verranno chiamate prima di altre.

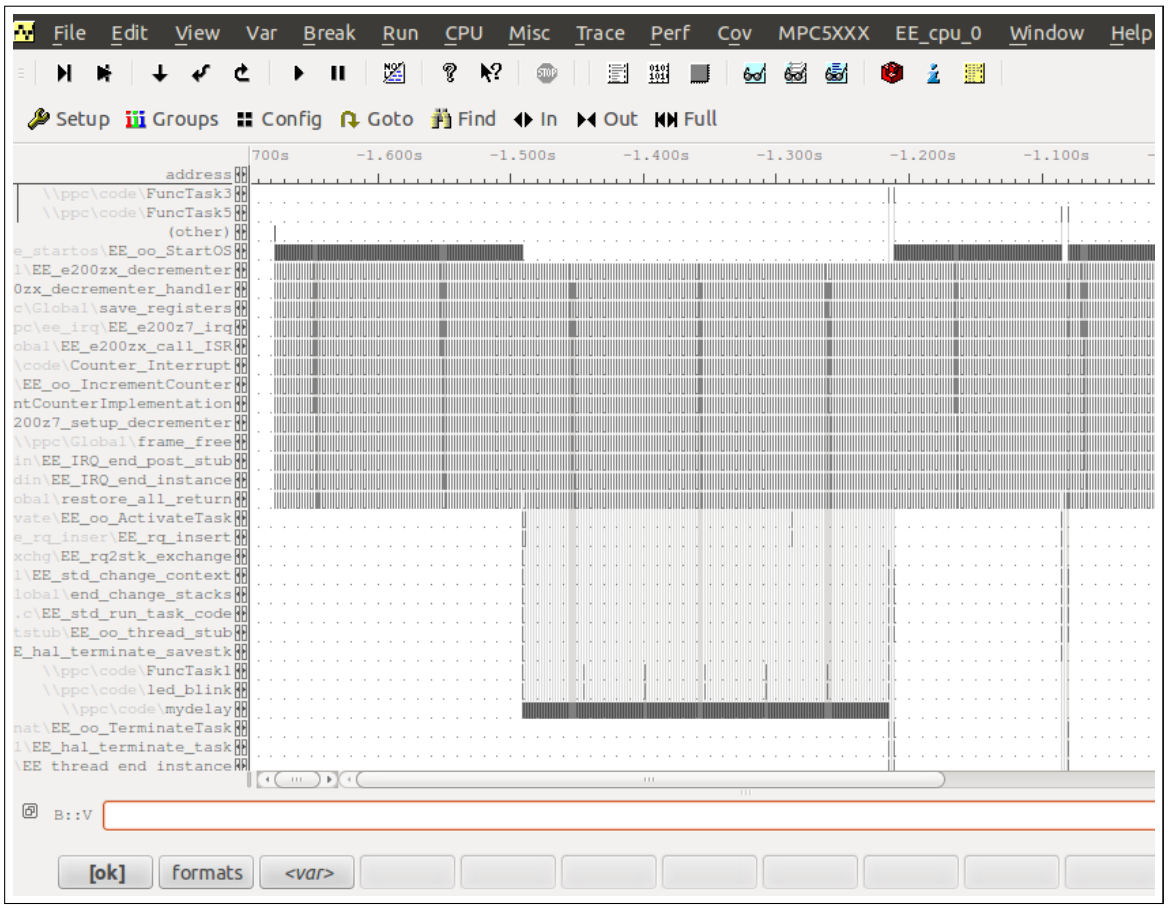

Figura 2.7: Chart ottenuto dal debug su Lauterbach Trace32

Nella figura 2.7 sono riportate le varie chiamate alle funzioni da parte del sistema operativo, a partire da *startOS()*. Le chiamate vengono identificate da segmenti verticali. Come si può osservare dalle prime due righe, le chiamate alla task3 vengono effettuate prima di quelle alla task 5.

Per questo motivo si è deciso di inserire la funzione di **criptaggio** nella task3 e quella di **decriptaggio** nella task5.

Nota: La scelta di queste due task è stata subordinata solo da motivi temporali. Quindi, come sono state scelte task3 e task5, potevano essere scelte altre due task qualsiasi, o scriverne di nuove, a patto che vengani rispettati i vincoli temporali citati in precedenza.

2.4.1 Implementazione

Si sottolinea come questo lavoro è conforme alle specifiche del **NIST** sull'ADVANCED ENCRYPTION STANDARD (AES).

Il NIST (National Institute of Standards and Technology) è un'agenzia del governo degli Stati Uniti d'America che si occupa della gestione delle tecnologie.

Tra i vari compiti del NIST vi è quello della pubblicazione dei Federal Information Processing Standard (FIPS), al fine di definire gli standard obbligatori del governo americano. I FIPS servono infatti a delineare con precisione il Data Encryption Standard e l'Advanced Encryption Standard.

Si vuole ora spiegare in modo dettagliato le varie parti di codice utili alla stesura delle due funzioni di encrypt e decrypt. L'algoritmo utilizzato è stato AES 128bit, di cui si è parlato nella sezione 2.2.5.

Nota:

Nb = numero di colonne di State;

Nk = numero di word da 32 bit nella chiave

Nr = numero di round nella cifratura

S-box = Tabella usata nella sostituzione di più byte e nella routine Key Expansion per eseguire una sostituzione uno per uno di un valore di byte.

Cipher

```
1  static void Cipher(state_t* state, const uint8_t* RoundKey)
2  {
3    uint8_t round = 0;
4
5    AddRoundKey(0, state, RoundKey);
6
7    for (round = 1; ; ++round)
8    {
9      SubBytes(state);
10     ShiftRows(state);
11     if (round == Nr) {
12       break;
13     }
14     MixColumns(state);
```

```
15      AddRoundKey(round, state, RoundKey);
16    }
17
18    AddRoundKey(Nr, state, RoundKey);
19  }
```

All'inizio della funzione Cipher, l'input è copiato nell'array State. Dopo un'aggiunta iniziale della prima Round Key, l'array State viene trasformato dopo esser stato elaborato in 10 round, con il round finale diverso leggermente dai primi giri Nr -1, in quanto non include la funzione MixColumns().

Le singole funzioni: SubBytes(), ShiftRows(), MixColumns() e AddRoundKey() – elaborano lo stato e sono descritte nelle seguenti sottosezioni. Nel codice l'array RoundKey[] contiene la chiave che viene usata. In AES 128bit ha una lunghezza di 128 bit.

Alla fine viene chiamata nuovamente la funzione AddRoundKey().

AddRoundKey

```
1  static void AddRoundKey(uint8_t round, state_t* state, const uint8_t*
       RoundKey)
2  {
3    uint8_t i,j;
4    for (i = 0; i < 4; ++i)
5    {
6      for (j = 0; j < 4; ++j)
7      {
8        (*state)[i][j] ^= RoundKey[(round * Nb * 4) + (i * Nb) + j];
9      }
10   }
11 }
```

Questa funzione aggiunge la Round Key allo stato attarverso una funzione XOR.

KeyExpansion

```
1  static void KeyExpansion(uint8_t* RoundKey, const uint8_t* Key)
2  {
3    unsigned i, j, k;
4    uint8_t tempa[4]; // Used for the column/row operations
5
6    for (i = 0; i < Nk; ++i)
7    {
8      RoundKey[(i * 4) + 0] = Key[(i * 4) + 0];
```

```
9       RoundKey[(i * 4) + 1] = Key[(i * 4) + 1];
10      RoundKey[(i * 4) + 2] = Key[(i * 4) + 2];
11      RoundKey[(i * 4) + 3] = Key[(i * 4) + 3];
12    }
13
14    for (i = Nk; i < Nb * (Nr + 1); ++i)
15    {
16      {
17        k = (i - 1) * 4;
18        tempa[0]=RoundKey[k + 0];
19        tempa[1]=RoundKey[k + 1];
20        tempa[2]=RoundKey[k + 2];
21        tempa[3]=RoundKey[k + 3];
22
23      }
24
25      if (i % Nk == 0)
26      {
27          /*Function RotWord()*/
28        {
29          const uint8_t u8tmp = tempa[0];
30          tempa[0] = tempa[1];
31          tempa[1] = tempa[2];
32          tempa[2] = tempa[3];
33          tempa[3] = u8tmp;
34        }
35          /*Fuction SubWord()*/
36        {
37          tempa[0] = getSBoxValue(tempa[0]);
38          tempa[1] = getSBoxValue(tempa[1]);
39          tempa[2] = getSBoxValue(tempa[2]);
40          tempa[3] = getSBoxValue(tempa[3]);
41        }
42
43        tempa[0] = tempa[0] ^ Rcon[i/Nk];
44      }
45
46      j = i * 4; k=(i - Nk) * 4;
47      RoundKey[j + 0] = RoundKey[k + 0] ^ tempa[0];
48      RoundKey[j + 1] = RoundKey[k + 1] ^ tempa[1];
49      RoundKey[j + 2] = RoundKey[k + 2] ^ tempa[2];
50      RoundKey[j + 3] = RoundKey[k + 3] ^ tempa[3];
51    }
52  }
```

L'algoritmo usa la chiave di cifratura Key e produce una routine di espansione della chiave per generare Nb(Nr+1) round keys, che verranno usate in ogni round per decriptare lo State.

Nel primo round, dalla linea 6 alla 12, la round key è la chiave stessa. Tutti gli

altri round dipendono dalle precedenti round keys.

Come è definito dalle specifiche del NIST, sono presenti due sotto-funzioni che in questa implementazione sono state inserite nell'intera funzione KeyExpansion: RotWord(), SubWord().

- La funzione RotWord(), dalla linea 29 alla 33, sposta di una posizione a sinistra una word.

  Ad esempio: [a0,a1,a2,a3] diventa [a1,a2,a3,a0]

- La funzione SubWord(), dalla linea 37 alla 40, prende una word alla quale applica la matrice S-box per ognuno dei quattro byte.

SubBytes

```
1  static void SubBytes(state_t* state)
2  {
3    uint8_t i, j;
4    for (i = 0; i < 4; ++i)
5    {
6      for (j = 0; j < 4; ++j)
7      {
8        (*state)[j][i] = getSBoxValue((*state)[j][i]);
9      }
10   }
11 }
```

La funzione SubBytes sostituisce i valori nella matrice State con valori nella matrice S-box.

ShiftRows

```
1  static void ShiftRows(state_t* state)
2  {
3    uint8_t temp;
4
5    temp           = (*state)[0][1];
6    (*state)[0][1] = (*state)[1][1];
7    (*state)[1][1] = (*state)[2][1];
8    (*state)[2][1] = (*state)[3][1];
9    (*state)[3][1] = temp;
10
11   temp           = (*state)[0][2];
12   (*state)[0][2] = (*state)[2][2];
```

```
13    (*state)[2][2] = temp;
14
15    temp           = (*state)[1][2];
16    (*state)[1][2] = (*state)[3][2];
17    (*state)[3][2] = temp;
18
19    temp           = (*state)[0][3];
20    (*state)[0][3] = (*state)[3][3];
21    (*state)[3][3] = (*state)[2][3];
22    (*state)[2][3] = (*state)[1][3];
23    (*state)[1][3] = temp;
24  }
```

La funzione ShiftRows fa uno shift a sinistra delle varie word all'interno di State. Ogni shift è fatto con un differente offset.

MixColumns

```
1  static void MixColumns(state_t* state)
2  {
3    uint8_t i;
4    uint8_t Tmp, Tm, t;
5    for (i = 0; i < 4; ++i)
6    {
7      t   = (*state)[i][0];
8      Tmp = (*state)[i][0] ^ (*state)[i][1] ^ (*state)[i][2] ^ (*state)
       [i][3] ;
9      Tm  = (*state)[i][0] ^ (*state)[i][1] ; Tm = xtime(Tm);  (*state)
       [i][0] ^= Tm ^ Tmp ;
10     Tm  = (*state)[i][1] ^ (*state)[i][2] ; Tm = xtime(Tm);  (*state)
       [i][1] ^= Tm ^ Tmp ;
11     Tm  = (*state)[i][2] ^ (*state)[i][3] ; Tm = xtime(Tm);  (*state)
       [i][2] ^= Tm ^ Tmp ;
12     Tm  = (*state)[i][3] ^ t ;              Tm = xtime(Tm);  (*state)
       [i][3] ^= Tm ^ Tmp ;
13   }
14 }
```

La funzione MixColumns mischia le colonne della matrice State.

InvCipher

```
static void InvCipher(state_t* state, const uint8_t* RoundKey)
{
  uint8_t round = 0;

  AddRoundKey(Nr, state, RoundKey);

  for (round = (Nr - 1); ; --round)
  {
    InvShiftRows(state);
    InvSubBytes(state);
    AddRoundKey(round, state, RoundKey);
    if (round == 0) {
      break;
    }
    InvMixColumns(state);
  }
}
```

La funzione InvCipher() è la funzione inversa rispetto a Cipher(). Essa si occupa del decriptaggio di un testo cifrato. Oltre alla funzione già vista AddRoundKey(), utilizza InvShiftRows(), InvSubBytes(), InvMixColumns(), che sono le funzioni inverse di quelle analizzate in precedenza.

In InvCipher() si osserva che, come in Cipher(), viene chiamata inizialmente la funzione AddRoundKey(). Poi vengono effettuati Nr rounds. I primi Nr-1 sono identici. L'ultimo non comprende InvMixColumn().

InvShiftRows

```
static void InvShiftRows(state_t* state)
{
  uint8_t temp;

  temp = (*state)[3][1];
  (*state)[3][1] = (*state)[2][1];
  (*state)[2][1] = (*state)[1][1];
  (*state)[1][1] = (*state)[0][1];
  (*state)[0][1] = temp;

  temp = (*state)[0][2];
  (*state)[0][2] = (*state)[2][2];
  (*state)[2][2] = temp;

```

```
15    temp = (*state)[1][2];
16    (*state)[1][2] = (*state)[3][2];
17    (*state)[3][2] = temp;
18
19    temp = (*state)[0][3];
20    (*state)[0][3] = (*state)[1][3];
21    (*state)[1][3] = (*state)[2][3];
22    (*state)[2][3] = (*state)[3][3];
23    (*state)[3][3] = temp;
24  }
```

La funzione InvShiftRows(), ruota ogni word a destra, ognuna con un offset differente.

InvSubBytes

```
1  static void InvSubBytes(state_t* state)
2  {
3    uint8_t i, j;
4    for (i = 0; i < 4; ++i)
5    {
6      for (j = 0; j < 4; ++j)
7      {
8        (*state)[j][i] = getSBoxInvert((*state)[j][i]);
9      }
10   }
11 }
```

La funzione InvSubBytes() sostituisce i valori nella matrice State.

InvMixColumns

```
1  static void InvMixColumns(state_t* state)
2  {
3    int i;
4    uint8_t a, b, c, d;
5    for (i = 0; i < 4; ++i)
6    {
7      a = (*state)[i][0];
8      b = (*state)[i][1];
9      c = (*state)[i][2];
10     d = (*state)[i][3];
11
```

```
12      (*state)[i][0] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply
    (c, 0x0d) ^ Multiply(d, 0x09);
13      (*state)[i][1] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply
    (c, 0x0b) ^ Multiply(d, 0x0d);
14      (*state)[i][2] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply
    (c, 0x0e) ^ Multiply(d, 0x0b);
15      (*state)[i][3] = Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply
    (c, 0x09) ^ Multiply(d, 0x0e);
16    }
17  }
```

La funzione InvMixColumns(), come la sua inversa, mischia le colonne della matrice State.

Capitolo 3

Conclusione

L'esperienza di tirocinio svolta presso il Dipartimento di Automatica ed Informatica del Politecnico di Torino è stato un primo passo verso il mondo del lavoro. Nonostante non fossi in azienda, ho dovuto affrontare un progetto che non era più solo a scopo didattico, ma che era utile per la realizzazione di un lavoro vero e proprio. Ciò mi è stato molto utile in quanto mi ha permesso di apprendere vari concetti e acquisire soft skills che non sarebbero state possibili migliorare seguendo un corso tradizionale proposto dall'università.

Concetti appresi Grazie a questo tirocinio mi sono avvicinato alla branca dell'informatica che più mi intriga: la sicurezza informatica, topic che non è approfondito bene nel percorso di studio da me affrontato.

In particolare è stato utile studiare concetti interessanti come la crittografia e l'autenticazione delle firme digitali, e alcuni anche messi in pratica come l'algoritmo di crittografia di tipo AES, sia per il criptaggio che per il decriptaggio di un testo.

Altre nozioni apprese sono la struttura di un sistema operativo, e i suoi vari componenti che sono utili per il funzionamento ottimale dello stesso.

Una ulteriore abilità acquisita è stata quella di interfacciarmi con un ambiente di debug integrato per progetti embedded come Trace32 di Lauterbach.

Soft skills apprese Il tirocinio svolto è stato utile per sperimentare per la prima volta un ambiente diverso, quello lavorativo.

Mi ha permesso di mettere alla prova le mie skills di organizzazione autonoma, del rispetto delle deadlines imposte dal monte ore dello stage, la pianificazione del lavoro, problem solving, l'interazione con esperti, in particolare il tutor Oberti Franco che mi ha seguito da vicino.

Ho potuto quindi migliorare la mia capacità di comunicazione e organizzazione.

Considerazioni finali Mi ritengo soddisfatto del lavoro svolto. Per la prima volta ho potuto testare le mie abilità su un progetto reale e non più didattico. Lo reputo un percorso formativo, un primo passo che ha arricchito il mio curriculum con un'esperienza piena di nuove sfide e realtà. Sono convinto che la scelta di svolgere un tirocinio sia stata la scelta migliore rispetto a seguire un corso tradizionale.

Ora ho una consapevolezza diversa di ciò che mi aspetta alla fine degli studi e mi sento cresciuto sotto più punti di vista.