Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

STRUTTURE INTELLIGENTI
Soluzioni innovative per la diagnostica strutturale: uno studio sperimentale

Relatore:
Chiar.mo Prof.
Fabio Panzieri

Correlatore:
Chiar.mo Prof.
Nicola Testoni

Presentata da:
Francesco Negretti

Sessione II
Anno Accademico 2015/2016

Indice

Elenco delle figure

Capitolo 1

Introduzione

1.1 Scopo

Il progetto "Intelligent Strutures" (IS) in cui si inserisce questa attivitá di tesi nasce con lobiettivo di garantire la sicurezza dei velivoli prima del decollo, durante tutta la vita dellaeromobile.

Le attuali tecnologie per la diagnostica dei velivoli sono molto ingombranti, pesanti e costose, ma soprattutto la loro accuratezza dipende dalle capacitá delloperatore che effettua il monitoraggio, grosso deficit che non é piú tollerabile in un epoca digitale.

Per ispezionare un velivolo e verificarne lintegritá strutturale, sono necessarie 24 ore, un tempo di fuori servizio molto lungo che in termini economici costituisce una grossa perdita per le compagnie aeree.

Lobiettivo di questo progetto é quello di abbattere questi tempi da 24 ore a 15 minuti.

1.1.1 Soluzione Proposta

Lo strumento utilizzato per raggiungere questo obiettivo é una rete di sensori piezoelettrici [12] ad ultrasuoni, da "annegare" nella struttura del velivolo stesso. Tali sensori sono basati su una struttura modulare e facilmente scalabile.

Distribuendo i sensori sulla superficie dell'aeromobile (fig. 1.1), e collegando questi ultimi tra di loro, si crea una rete dallerta capace di rilevare urti e danneggiamenti [11, 14].

Basta infatti tarare le soglie di tolleranza di questa rete per far sí che la struttura sia in grado di rilevare le onde meccaniche generate dagli urti e stimare in poco tempo l'integritá del materiale, effettuando in tal modo un rapido auto-check per verificare che tutto il volume della struttura sia integro.

1.2 Nuove Tecnologie

Figura 1.1: Esempio disposizione Sensori

Questa nuova soluzione proposta minimizza i costi di manutenzione, passando da una metodologia "preventiva" a una metodologia "predittiva". Inoltre abbatte il peso del sistema di controllo stesso: si consideri che un singolo sensore pesa 4 grammi, e sebbene la popolazione di questi sensori conti un migliaio di nodi, lintero volume necessario per controllare un intero velivolo non supera in peso di qualche chilo.
Lidea é quella di trasformare il velivolo, o qualsivoglia struttura da analizzare, in una Struttura Intelligente (fig. 1.1), capace di auto diagnosticarsi secondo un protocollo *ad hoc* scritto per questi sensori.

Lintento é quello di riprodurre una pelle virtuale che emuli il funzionamento dei recettori del corpo umano anche su strutture diverse, permettendo ad esse di ricevere stimoli, tramite i sensori, ed essere quindi in grado di rispondere nella maniera piú adeguata [6].

1.3 Stato dell'arte

1.3.1 Sensoristica

Attualmente in commercio ci sono pochi competitor che propongono soluzioni simili a quelle trattate in questa tesi, e talvolta queste soluzioni presentano problemi

che sono stati risolti all'interno di IS.
In particolar modo spiccano due aziende: la DEA Security, un'azienda italiana con sede a La Spezia che progetta, sviluppa e produce una gamma di sistemi di rivelazione antintrusione perimetrali, e la Mica Mote [3], piattaforma hardware e software open-source sviluppata da ricercatori U.C. Berkeley, che combina sensori, comunicazioni, e informatica in un'architettura wireless.
Nel capitolo dedicato allo stato dell'arte, entrambi questi progetti, saranno analizzati nello specifico e verranno confrontati passo a passo con quanto invece viene proposto in questo elaborato.

1.3.2 Diagnostica Aeromobili

In questo campo i competitor che propongono soluzioni simili a quelle proposte da IS sono principalmente due: ACELLENT e AernNova.
Il monitoraggio che propone Acellent può essere applicato a diverse applicazioni in innumerevoli settori, tra cui aerospaziale, infrastrutture civili, energia e difesa personale. Il sistema è costituito da tre componenti principali che integrano insieme: sensori, hardware e software. Essi possono essere utilizzati sia in una modalità autonoma attiva o passiva, o una modalità combinata passiva-attiva [8].
AerNnova é un azienda molto grande e giá avviata da tempo, che si occupa della costruzione e del montaggio delle parti degli aerei. All'interno del suo reparto di ricerca e sviluppo é stato sviluppato un progetto di nome Pamela. Benché ancora in fase di studio e non ancora ad alti livelli, questo prodotto procede peró molto velocemente grazie alle numerose risorse a cui puó attingere. Al momento Pamela si compone di diversi nodi sensori che vengono messi su ogni pannello di cui sono composte le strutture dei velivoli (fig. 1.2).

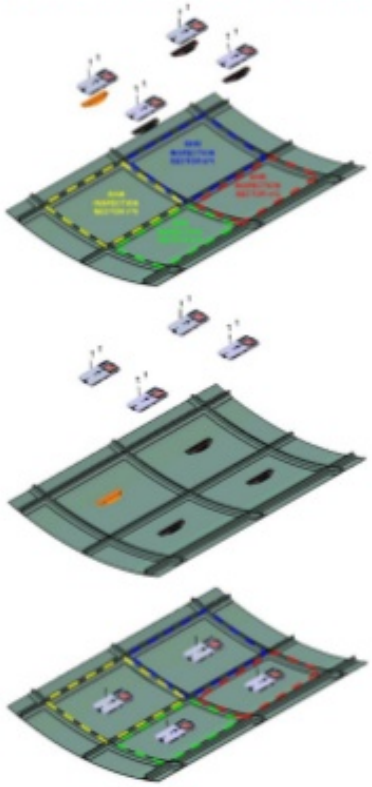

Figura 1.2: Disposizione sensori Pamela

Questi sensori ne monitorizzano integritá ed eventuali danneggiamenti.

1.4 Risultati Attesi

Al momento attuale la comunicazione e la circolazione interna dei pacchetti, che permettono lo scambio di dati tra i sensori e il sistema di controllo [9], é funzionale solamente per la fase di testing, é ancora lontana dall'essere pronta per la commercializzazione.
La sicurezza delle comunicazioni é infatti assente, non essendovi ancora nessuna criptazione dei dati trasmessi: la codifica Manchester, che come vedremo meglio piú avanti viene utilizzata per compensare alcune lacune Hardware, non ha lo scopo di essere una cifratura, e non rientra infatti tra i metodi utilizzati per trasmettere messaggi in sicurezza.
Tutto il funzionamento, e l'affidabilità dei dati, è ad alto rischio di attacco: basterebbe infatti *sniffare* i pacchetti in transito per avere in chiaro le informazioni trasmesse dai sensori.
Queste informazioni sono molto sensibili e non deve quindi essere possibile rubarle.

É necessario intervenire su questo aspetto e criptare i messaggi trasmessi tramite una chiave privata (Fig. 1.3), in modo che solamente il destinatario sappia come decodificare le informazioni e utilizzarle.

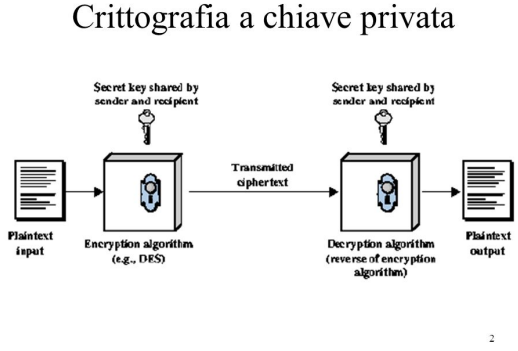

Figura 1.3: Comunicazione tramite chiave privata

Inoltre non è ancora previsto un servizio che permetta di conoscere la correttezza dei pacchetti ricevuti.
Può infatti capitare che durante durante le trasmissioni un pacchetto venga parzialmente o totalmente perso. In questo momento il sistema non è in grado di rilevare se accidentalmente accade uno smarrimento totale o parziale dei dati.
Di conseguenza non è implementato un algoritmo che permetta di ritrasmettere l'intero pacchetto nel caso servisse. I dati sono quindi soggetti ad incertezza fintanto che non si ha il modo di controllare che tutti i rilevamenti fatti dai sensori siano realmente arrivati a destinazione per intero.
Per risolvere questo problema è necessario inserire all'interno dei pacchetti stessi informazioni aggiuntive in modo da avere un riscontro e poter fornire al server una notifica di ricezione corretta/sbagliata che faccia proseguire la trasmissione o ripeterne una parte.
Questa idea si ispira al contenuto dell'header nei pacchetti IP (Fig. 1.4), infatti in esso sono contenute le informazioni utili per conoscere tutti gli aspetti di quel pacchetto, sicuramente non verranno utilizzati gli stessi campi che utilizzano i pacchetti IP, perché sono troppi per le attuali necessità di IS, però una buona parte serve a tale scopo e verrà quindi riutilizzata.

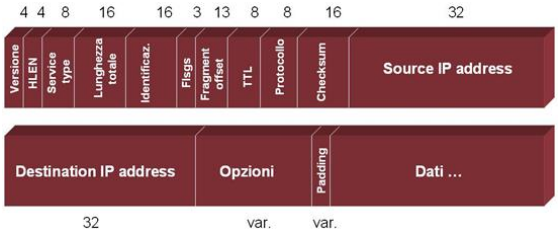

Figura 1.4: Composizione di un pacchetto IP

Al termine di questo lavoro mi aspetto quindi che il sistema di gestione delle comunicazioni rispetti gli standard discussi in precedenza, e che sia pronto per essere testato in ambienti reali. Saranno adeguati i protocolli di comunicazione già esistenti, snellendoli per cercare di migliorare le prestazioni, e ampliando le possibilità di comunicazioni che hanno in questo momento. Inoltre saranno introdotte nuove *features*, il tracciamento dei pacchetti e la criptazione delle comunicazioni.

Saranno presenti molte informazioni aggiuntive all'interno di ogni pacchetto, i normali pacchetti IP ad esempio hanno un campo *Total Length* che indica la lunghezza totale del pacchetto IP: il ricevitore é quindi in grado di calcolare se ci sono state delle perdite di dati in modo da identificare il pacchetto danneggiato, e conoscerne provenienza, destinazione e dimensione. In questo modo sarà impossibile smarrire pacchetti sia in modo parziale, sia totale.
La correttezza delle comunicazioni sarà così garantita e le informazioni sullo stato delle strutture monitorate saranno certe.

Inoltre ogni informazione inviata nella rete sarà protetta dai principali tipi di attacco, infatti con un opportuno algoritmo di crittografia, i pacchetti che transiteranno non saranno decifrabili se non dal server che ne conosce la chiave privata. In questo modo il vincolo di sicurezza sarà rispettato e le informazioni relative allo stato della struttura monitorata saranno, oltre che affidabili grazie al tracciamento dei pacchetti, anche certe. Infatti le informazioni contenute nei pacchetti saranno interpretate solamente una volta giunte a destinazione.

1.5 Struttura della tesi

Nei prossimi capitoli verrá trattato piú nello specifico lo stato dell' arte, ovvero i modi alternativi in cui il problema é affrontato, ne verranno discussi vantaggi e svantaggi confrontando le loro metodologie con quelle proposte da IS e da questo elaborato.

Successivamente un capitolo sará dedicato alla metodologia e all'architettura utilizzata per sviluppare questo progetto, ovvero uno sguardo in generale al funzionamento del sistema nel suo complesso, da quali parti é composto e da come interagiscono tra di loro.
Successivamente il capitolo dedicato all'Implementazione vedrá nello specifico che ambiente di sviluppo é stato utilizzato, quali linguaggi di programmazione e perché, oltre che la visione dettagliata dell'implementazione delle funzionalitá.
Infine, prima delle Conclusioni tratte al termine di questa esperienza di tesi, verranno illustrati i risultati ottenuti dai test di funzionamento del sistema, e si cercherá di dare una interpretazione a questi risultati per vedere se le soluzioni proposte possono essere concorrenziali e competitive con le tecnologie giá esistenti.

Capitolo 2

Stato dell'arte

Come giá accennato nell'introduzione, questa tecnologia, per quanto sia nuova e abbia ampi spazi per emergere in un mercato povero di prodotti analoghi, ha comunque altri prodotti concorrenti che svolgono funzioni simili giá presenti nel panorama industriale.
Nello specifico DEA Security e Mica Mote.

2.1 DEA Security

DEA Security è uno dei leader europei nella progettazione e produzione di sistemi perimetrali per la rilevazione delle intrusioni. Tra i prodotti proposti da DEA vi è il sistema interrato SISMA CP, la linea di sistemi per recinzioni metalliche SERIR e TORSUS, i sistemi per pavimentazioni SISMA CA e SISMA CA PF, e in particolar modo i rivelatori sismici per la protezione di infissi, vetrate, pareti e casseforti.

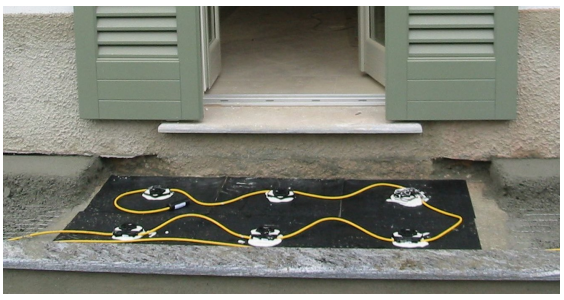

Figura 2.1: Sistema rilevamento intrusione DEA Security

Nonostante questa azienda si occupi di sistemi anti-intrusione, molti punti sono in comune con quello che è stato sviluppato in IS, infatti vengono utilizzati reti di sensori collegati tra loro tramite cavi (Fig. 2.1), e gestiti da un unica centralina per rilevare vibrazione tramite sensori piezoelettrici.
In sostanza utilizzano quindi la stessa tecnologia della rete proposta da IS, applicata peró in un campo diverso, quello della sicurezza intesa come *security*. La differenza sostanziale a livello di hardware tra le soluzioni proposte é la dimensione dei nodi sensori. I loro prodotti, pur variando a seconda del modello, hanno dimensioni di 51 x 31 x 22 mm, mentre i sensori proposti da IS sono 30 x 23 x 4 mm.
Nonostante il grosso miglioramento per quanto riguarda le dimensioni, questa caratteristica non é del tutto rilevante nel campo della sicurezza, infatti il miglior deterrente é la conoscenza del rischio, perché quindi nascondere i sensori quando eventuali malintenzionati potrebbero essere giá scoraggiati solamente dalla consapevolezza di un sistema di sicurezza? Nonostante quindi la somiglianza nelle tecnologie e la riduzione delle dimensioni il campo Security non si presta bene a questo tipo di tecnologia, decisamente piú promettente il campo Safety.

2.2 Mica Mote

Nel campo Safety, quindi per quello che riguarda la conservazione di strutture e materiali, troviamo invece la nascente Mica Mote.
Questa piattaforma si chiama MICA perché la sua attuazione elettronica finale assomiglia agli omonimi minerali silicati che si separano in foglie sottili. Allo stesso modo, l'hardware elettronico MICA è una serie di schede di sensori processore / radio e messi insieme per formare un sensore intelligente wireless integrato.
Il comparto hardware è costituito da una piccola scheda radio a bassa potenza (Fig. 2.2), un processore (noto come processore Mote / radio o MPR) e una o più schede di sensori (sensore mote). La combinazione dei due tipi di schede formano un sensore wireless in rete.

Figura 2.2: Mica2

Il sensore wireless viene alimentato tramite due batterie di tipo AA, e il consumo di energia determina la durata della batteria. I processori, la radio, e un carico tipico del sensore consumano circa 100 mW: pertanto sono consigliate batterie a lunga durata.
Per gestire la rete di sensori viene utilizzato un sistema operativo molto leggero, OpenSource, sviluppato appositamente per dispositivi wireless a bassa potenza, noto come TinyOS[1], che permette alle reti la gestione dell'alimentazione, e la trasmissione dei dettagli di misurazione del sensore.
Questo progetto presenta diversi punti critici su cui lavorare: alcuni di essi sono stati risolti nella versione proposta da IS e da questo elaborato.
In primo luogo l'alimentazione é un grosso deficit. Infatti per quanto piccoli vengano fatti i sensori necessitano sempre e comunque di una batteria (Fig. 2.2), che va ad intaccare la dimensione ed il peso stesso del blocco sensore, e ha una durata di carica finita, dopodiché va sostituita. Questo problema é risolto grazie alla tecnologia Data Over Power utilizzata da IS, infatti, come vedremo nel capitolo 3 sono stati ridotti al minimo il numero dei cavi, é possibile evitare di avere batterie e i sensori possono essere immersi direttamente nella struttura, non necessitando di altri tipi di manutenzione.

[1]https://github.com/tinyos/tinyos-main

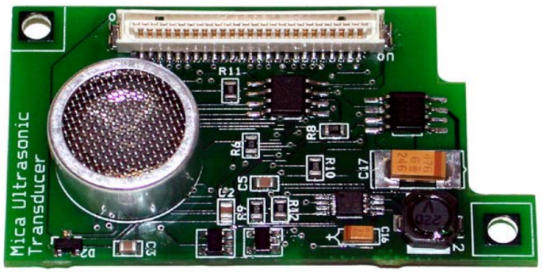

Figura 2.3: Ultrasonic Transceiver

Un secondo punto critico sono le dimensioni vere e proprie del nodo sensore (Fig. 2.3, infatti le misure prodotti Mica Mote sono ancora intorno ai 50mm, circa il doppio di quanto sviluppato da IS.
Infine l'impiego di dispositivi wireless necessita di un sistema operativo che li controlli per potere avere accesso diretto ad ogni nodo: in strutture con cosí basse risorse hardware (16KB di memoria e 1KB di RAM, vedremo questa problematica nel capitolo 3), un sistema operativo, per quanto leggero come TinyOS [4] utilizza una quantitá troppo elevata di queste risorse.
Il collegamento cablato come quello presente nella rete di sensori descritta in questo elaborato, permette di avere il sistema operativo solo sul gestore del sistema, mentre tutti i dispositivi (Gateway e Nodi Sensori) comunicano tra loro tramite una routine che si ripete ciclicamente, risparmiando quindi grosse quantitá di risorse utilizzabili diversamente.

2.3 Acellent

I prodotti Acellent forniscono soluzioni per mantenere sotto controllo l'integrità strutturale e monitoraggio della salute strutturale di aeromobili e di altre strutture civili ed industriali. Il sistema é composto da sensori SMART Layer, hardware e software ScanGenie per rilevare i danni nascosti e delaminazioni a livello globale nelle grandi strutture (Fig 2.4).
Le due modalità di funzionamento generali dei sistemi di monitoraggio strutturale sono "on-board" e "off-board". I dati sono registrati durante il volo in modalità on-board, mentre vengono raccolti solo quando l'aeromobile è a terra in modalità off-board. Un sistema di bordo richiede che il sistema di monitoraggio completo (sensori, hardware e software) sia a bordo dell'aeromobile e sia operativo durante il volo, mentre un sistema off-board richiede solo i sensori installati in modo permanente a bordo dell'aeromobile. Con un sistema off-board, l'hardware e il software

sono a terra e vengono collegati al velivolo durante l'ispezione.

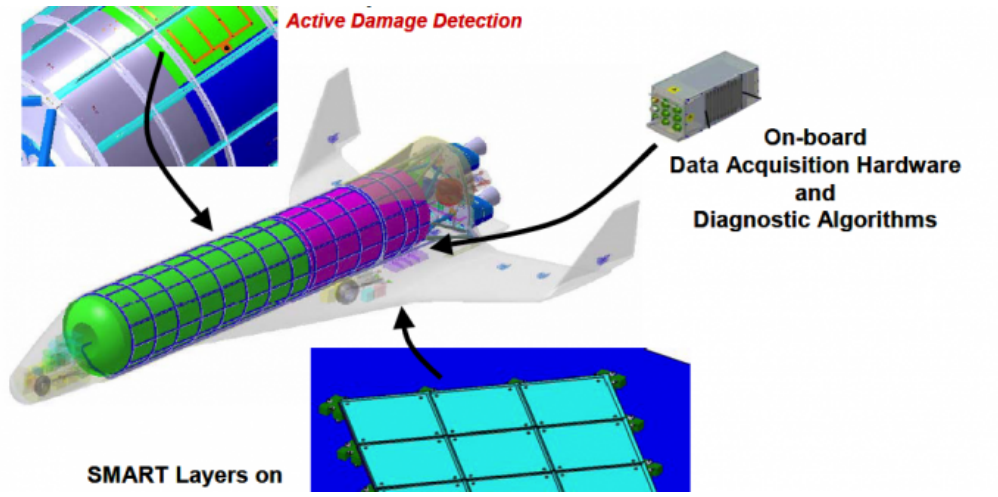

Figura 2.4: Sensori diagnostica Acellent

Questo sistema presenta peró alcuni difetti, infatti utilizzando una tecnica di localizzazione attiva che si basa sull'eccitazione sequenziale di un solo sensore alla volta, è possibile localizzare il danno solo dopo che questo è avvenuto, rendendo pertanto impossibile identificare l'istante e l'energia dell'impatto.
Con il sistema di IS invece la notifica é in tempo reale, perché il sistema in modalitá passiva riesce a gestire tutti i canali contemporaneamente in ogni istante, e quindi a rilevare un urto appena avviene.
Infine, nel loro sistema l'elettronica che interagisce con i sensori e li interroga, é posizionata distante dai sensori, a volte anche a 3/4 metri. Questa scelta da origine al problema del Cross-Talk, ovvero un interferenza elettromagnetica che si puó verificare se si hanno troppe trasmissioni cablate vicine: i rilevamenti risultano quindi alterati e risulta necessaria una onerosa fase di post-elaborazione per eliminare tale effetto negativo.
Il sistema di IS invece utilizza il metodo Sensor Near Elettronic, ovvero ogni sensore ha al suo fianco un piccolo apparato elettronico che ne gestisce il rilevamento e lo trasmette giá elaborato. In questo modo il segnale ricevuto dal sistema elettronico è sempre libero da interferenze e risulta possibile ridurre significativamente il numero di campioni da trasmettere al sistema di controllo.

2.4 AernNova - Pamela

Il sistema Pamela, sviluppato da AerNova, come giá accennato in precedenza, ha delle funzionalitá molto simili al progetto sviluppato da IS: a differenza di Acellent, loro usano un collaudato metodo Sensor Near Elettronci perfettamente funzionante, e hanno sia la parte attiva di rilevamento sia quella passiva.
I difetti del loro sistema sono peró di un altro tipo, i sensori completi di apparato elettronico hanno infatti un peso medio di 300gr l'uno, ognuno di questi sensori riesce a monitorare $1m^2$ di struttura e consuma 20W in attivo e 10W in passivo. Facendo una stima, ad esempio su un Boeing 787, saranno necessari circa 1250 sensori (ha una superficie alare di 520,24 m), quindi tutto il sistema avrebbe un peso intorno a 400kg e un consumo di 25kW. Questo aereo, che é un velivolo giá predisposto in un ottica futuristica orientata al funzionamento totalmente elettrico (Fig. 2.5), ha una produzione di 500kW [1]: il sistema Pamela, se adottato, assorbirebbe quindi oltre il 5% dell'energia; in un aereo tradizionale, con generatori meno preformanti, l'assorbimento sarebbe superiore al 10%.

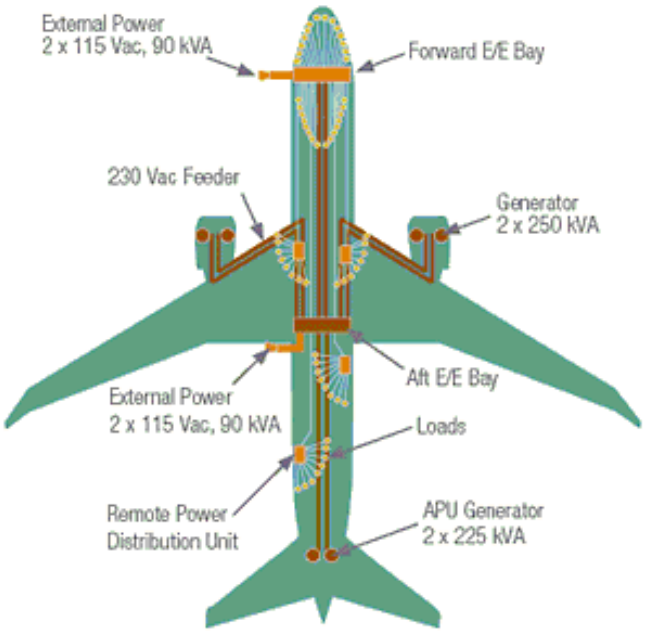

Figura 2.5: Distribuzione Energia Boeing 787

Ogni singolo nodo sensore sviluppato in IS ha invece un peso di 4g, un consumo massimo di potenza pari a 40mW ed é in grado di monitorare una superficie di 50x50 cm. Per monitorare un metro quadrato è quindi necessario utilizzare 4 sensori per un peso totale di 16g (contro i 300g/m^2 di AernNova), ed un consumo totale di 160mW contro i 20W del sistema di AernNova. Per riportarci all'esempio di prima, quindi su un boeing 787 tutta la rete di sensori avrebbe un peso di 20kg e un consumo di 200W, parametri che rientrano entro limiti accettabili per le compagnie aeree.

Capitolo 3

Architettura Proposta

3.1 Realizzazione

Su tutta la struttura che necessita di essere monitorata vengono installati sensori (Fig. 3.1), posizionati a una distanza tra loro compresa tra i 50 e i 70 cm a seconda della forma della struttura e del materiale: il Sistema Intelligente é in grado di monitorare lo stato di salute e di cogliere ogni impatto o danneggiamento tramite l'elettronica di elaborazione presente vicino ad ogni sensore (Fig. 3.5) [13].

3.1.1 Hardware

Figura 3.1: Composites-embedded SHM system

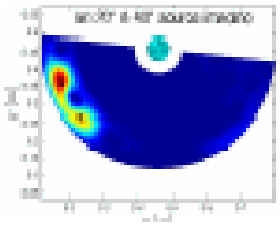

Figura 3.2: Rilevamento di un urto

Quando avviene un impatto, esso viene localizzato dai sensori, che rilevano il propagarsi delle vibrazioni prodotte dall'urto nel materiale (Fig. 3.2). Questi sensori sono collegati tra loro e ad un Gateway (Fig. 3.4) secondo uno schema di tipo Data Over Power, metodo di alimentazione simile al Power Over Ethernet, che permette di alimentare e scambiare dati utilizzando soli i cavi di alimentazione come illustrato in Fig. 3.3 [**?**].

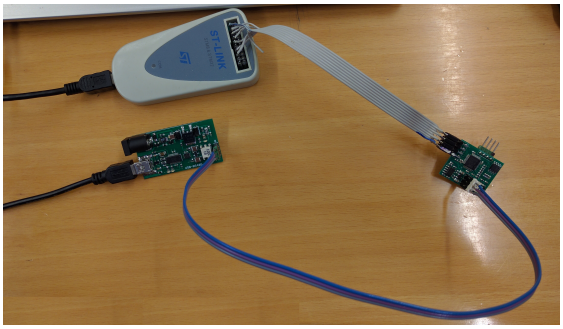

Figura 3.3: Schema collegamento sensori-gateway

Come già accennato in precedenza, i pacchetti che vengono scambiati sulla rete [5] contengono comandi oppure dati. I comandi sono delle word speciali che danno un ordine di esecuzione (per esempio *esegui la diagnosi della struttura*) oppure richiedono delle informazioni al sistema (per esempio *invia tutti i dati che hai raccolto*).

Con questa idea innovativa abbiamo quindi visto come sia possibile, grazie agli strumenti offerti dallo sviluppo tecnologico attuale e al lavoro di ricerca, ottimizzare una procedura che altrimenti richiederebbe molto tempo e sforzo.

Inoltre, grazie alla sua scalabilità e alla sua duttilità, lintero sistema è applicabile in diversi ambiti, come il controllo di interi edifici, automobili da gara, sottoposte allo stress della corsa, oppure bombole di gas e qualsivoglia struttura che ha bisogno di essere sempre sotto controllo.

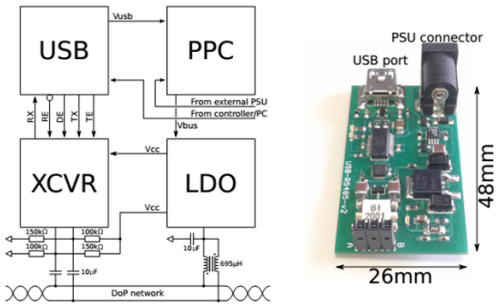

Figura 3.4: Gateway

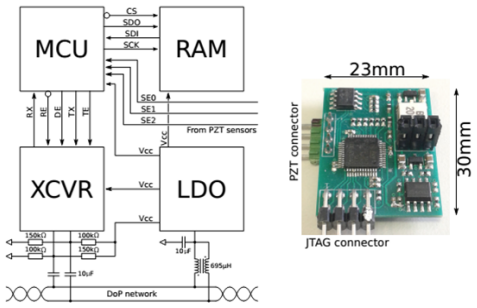

Figura 3.5: Nodo Sensore

3.2 Protocolli di trasmissione

I pacchetti usati per la trasmissione dei dati rilevati dai sensori sono di tipo 8-N-1, ovvero pacchetti da 10 bit, composti da start, dato e stop. Per codificare

e decodificare i dati viene utilizzata, come accennato nel capitolo 1, la Codifica Manchester, che prevede la sostituzione di ogni 0 con la sequenza binaria 01 e di ogni 1 con la sequenza binaria 10.

Questo tipo di codifica é stata resa necessaria per motivi hardware, infatti la durata della carica dei condensatori utilizzati per l'accoppiamento dell'interfaccia dati al bus di comunicazione non permette di mantenere il segnale a livello alto (basso) per una durata sufficiente a trasmettere una sequenza piú lunga di 3 bit a 1 (0). La codifica Manchester, anche se utilizzerà più spazio (ogni byte decodificato è rappresentato con 2 byte codificati) elimina questo problema perché crea un alternanza continua di 0 e 1.

Il pacchetto classico per la comunicazione è così composto:

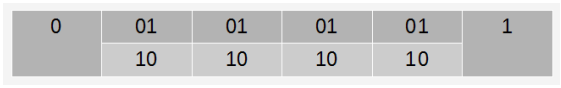

Figura 3.6: Pacchetto Classico

dove 0 indica linizio della sequenza, allinterno ci sono le informazioni, e 1 ne indica la fine.
Nella progettazione di questa comunicazione si é pensato di permettere lutilizzo di sequenze speciali, che hanno il compito di comunicare un comando. Infatti fissando i primi due bit e gli ultimi due a 01, e lavorando sulle restanti 3 coppie di bit si ha la possibilitá di creare word che non rispettano la codifica Manchester tradizionale, e sono quindi distinguibili dai pacchetti classici e interpretabili.

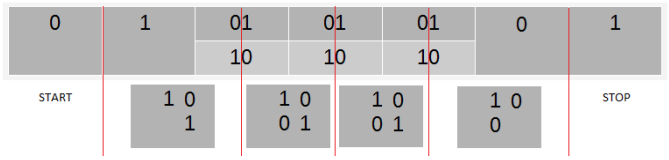

Figura 3.7: Sequenze Speciali

Le possibili configurazioni di bit nelle sequenze speciali sono 8, ma una di queste è anche una parola valida per la codifica manchester. I possibili comandi speciali implementati sono: CMDSTART, CMDSTOP, CMDACK, DATASTART, DATASTOP, DATAACK. Se necessario sarà dunque possibile inserire un ulteriore comando speciale.

Questi vengono utilizzati nelle comunicazioni per indicare il tipo di scambio di dati che deve avvenire.

Nello specifico le modalità di comunicazione sono 4:

- Long Query (LQRY), dove il Master chiede qualcosa e attende dei dati in risposta (ad esempio, *quali* dati sono stati raccolti).
- Short Query (SQRY), dove il Master chiede solamente un dato di tipo quantitativo (ad esempio, *quanti* dati sono stati raccolti).
- Long Command (LCMD), dove il Master invia una sequenza di comandi .
- Short Command (SCMD), dove il Master comunica qualcosa di breve, per esempio quanti dati sta per mandare.

Ad ogni pacchetto viene sempre risposto con un Acknolowdgment (ACK), per essere certi dellavvenuta consegna e non rischiare di perdere quindi i pacchetti.

3.3 Algoritmi di Crittazione

La crittografia (dall'unione di due parole greche: Κρυπτός (krypts) che significa "nascosto", e γραψια (grapha) che significa "scrittura") è la branca della crittologia che tratta delle "scritture nascoste", ovvero dei metodi per rendere un messaggio "offuscato" in modo da non essere comprensibile/intelligibile a persone non autorizzate a leggerlo.

Un tale messaggio si chiama comunemente crittogramma e i metodi usati sono detti tecniche di cifratura.(Cit. https://it.wikipedia.org/wiki/Crittografia)

Gli algoritmi di crittazione a chiavi simmetriche sono i tradizionali sistemi di crittazione dove due o più interlocutori concordano preventivamente un algoritmo e una chiave di crittazione, sicché poi possono continuare il loro dialogo scambiandosi messaggi dal contenuto incomprensibile agli eventuali intercettatori (Fig 3.8). L'unica chiave di cui hanno bisogno è l'informazione necessaria sia per crittare, sia per decrittare il messaggio. Se gli interlocutori sono più di due e utilizzano lo stesso algoritmo, allora ogni coppia dovrà concordare una chiave diversa.

Figura 3.8: Crittografia a chiave simmetrica

Per realizzare la crittografia di queste comunicazioni ho analizzato diversi algoritmi, tenendo sempre conto della necessità di rendere la comunicazione veloce e del fatto che le risorse hardware fossero molto limitate.

3.3.1 Blowfish

Il primo algoritmo che ho considerato è stato il Blowfish;
Blowfish ha una dimensione blocco a 64bit e una lunghezza di chiave variabile fra i 32 e i 448 bit. Ha una struttura a rete di Feistel a 16 cicli, e usa S-Box grandi e dipendenti dalla chiave.

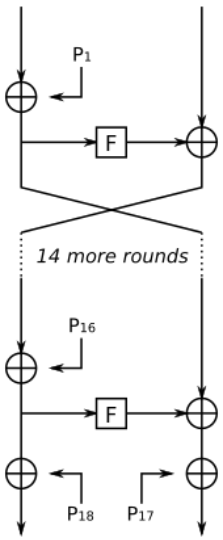

Figura 3.9: Diagramma di Blowfish

Il diagramma in Fig. 3.9 mostra come opera blowfish. Ogni linea rappresenta 32 bit. L'algoritmo gestisce due tipi di liste di sottochiavi: una lista di 18 elementi, definita P-array, e quattro liste da 256 elementi. Ciascuna S-Box ha in ingresso 8 bit di informazioni, e produce in uscita 32 bit. A ogni ripetizione del ciclo si usa un elemento diverso del P-array, e dopo l'ultimo ciclo a ogni metà del blocco di dati viene moltiplicata in XOR con uno dei due elementi inutilizzati del P-array.

Il diagramma della Fig. 3.9 rappresenta la funzione F di blowfish. La funzione divide 32 bit in ingresso in quattro byte, utilizzati a loro volta come ingressi delle S-Box. I risultati sono sommati in modulo 232 e messi in XOR, per ottenere il risultato finale di 32 bit cifrati.

Dato che blowfish è una rete di Feistel, può essere invertito semplicemente mettendo in XOR gli elementi 17 e 18 del P-array, e quindi utilizzando gli elementi del P-array in ordine inverso.

La lista delle chiavi di blowfish viene generata caricando i valori iniziali del P-array e delle S-Box con rappresentazioni esadecimali delle cifre di pi greco, che apparentemente non hanno periodicità o schemi ripetitivi. La chiave segreta viene quindi messa in XOR con ciascun elemento del P-array (ripetendo la chiave, quando necessario). Un blocco da 64bit di valore zero viene criptato con l'algoritmo stesso, e il risultato cifrato sostituisce gli elementi P-1 e P-2. Il risultato cifrato viene quindi codificato nuovamente con le nuove sottochiavi, e va a sostituire gli elementi P-3 e P-4. Questo procedimento continua fino a rimpiazzare tutti gli elementi del P-array e delle S-Box. In tutto, l'algoritmo di cifratura di blowfish viene eseguito 521 volte in modo da generare tutte le sottochiavi, ed elaborando circa 4kB di dati.

Nonostante questo algoritmo fosse molto sicuro purtroppo non andava bene, infatti ogni nuova chiave richiede un tempo di elaborazione equivalente alla codifica di 4 kB di testo, che è molto se confrontato con altri sistemi di cifratura a blocchi. Inoltre lo spazio di memorizzazione richiesto per i 18 P-array e le 4 S-box andava in contrasto con le esigenze di ridurre al minimo l'utilizzo di memoria per carenza di supporti hardware adeguati.

3.3.2 Cifrario Vigenère

Un altro algoritmo che ho analizzato è stato il cifrario di Vigenère: il metodo si può considerare una generalizzazione del cifrario di Cesare; invece di spostare sempre dello stesso numero di posti la lettera da cifrare, questa viene spostata di un numero di posti variabile ma ripetuto, determinato in base ad una parola chiave, da concordarsi tra mittente e destinatario, e da scrivere ripetutamente sotto il messaggio, carattere per carattere; la chiave viene detta anche verme, per il motivo che, essendo in genere molto più corta del messaggio, deve essere ripetuta molte volte sotto questo.
Il vantaggio rispetto ai cifrari monoalfabetici (come il cifrario di Cesare o quelli per sostituzione delle lettere con simboli/altre lettere) è evidente: il testo è cifrato con n alfabeti cifranti. In questo modo, la stessa lettera viene cifrata (se ripetuta consecutivamente) n volte; ciò rende quindi più complessa la crittoanalisi del testo cifrato.
Per semplificare la cifratura, Vigenère propose l'uso di una "matrice" quadrata Fig. 3.10, composta da alfabeti ordinati e spostati. Se si vuole cifrare,ad esempio, la lettera "H" della parola basterà trovare la lettera "H" nella prima riga, individuando la colonna relativa. Basterà poi trovare la "K" della chiave nella prima colonna per trovare la riga, individuando, tramite l'incrocio, la lettera corretta da usare. La complessità di questo cifrario, inteso come numero di operazioni necessario per forzarlo con un approccio di tipo *brute-force* è $O(n^k)$, dove dove n rappresenta il numero di simboli dell'alfabeto e k il numero di simboli della chiave.

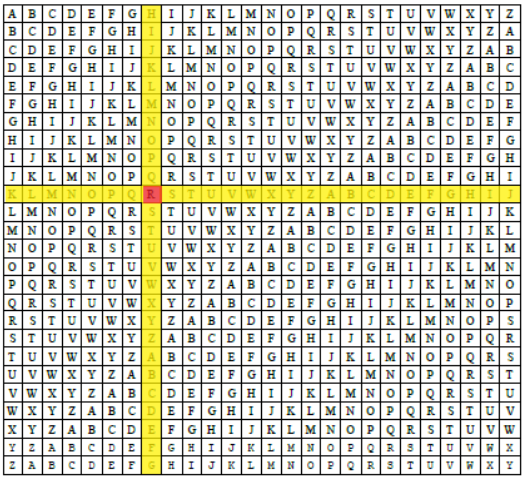

Figura 3.10: Matrice di Vigenere

Questo cifrario ha diversi aspetti positivi, infatti é leggero e facilmente implementabile, purtroppo peró é facilmente interpretabile, infatti se la lunghezza del messaggio é superiore di almeno 20 volte a quella del verme c'é una probabilitá di oltre il 95% di decifrare il contenuto utilizzando algoritmi noti facilmente reperibili online. Probabilitá troppo alta per le necessitá della trasmissione sviluppata.
Le caratteristiche positive di questo cifrario mi hanno peró spinto a cercare una soluzione accettabile che permetta di aumentare la sicurezza della trasmissione dei messaggi utilizzando questa tecnica.

3.4 Un idea per cifrare

Il cifrario di Vigenere utilizza le lettere, mentre ció che deve essere trasmesso all'interno di questa rete sono dei numeri, per l'esattezza i numeri sono solamente due, lo zero e l'uno, disposti in sequenze da otto (un byte). Come cifrario anziché avere una tabella da 23 x 23 (lettere) abbiamo quindi una tabella 2 x 2 (0 e 1) del tipo seguente.

0	1
1	0

Questa tabella può essere realizzata facilmente tramite la funzione "or" esclusivo, (XOR) dove la combinazione di due cifre identiche dà 0, e di due cifre diverse dà 1. Questo metodo di crittazione sembra più debole ma è possibile dimostrare che, fissata in *Kbit* la lunghezza in bit della chiave, e scegliendo per rappresentare sia il messaggio da codificare che la chiave di codifica un alfabeto composto da

$n = 2^{Nbit}$ simboli, la robustezza del cifrario di Vigenere non varia al variare di $Nbit$. Infatti possiamo calcolare k come $k = Kbit/Nbit$, dunque facendo i calcoli possiamo ottenere i risultati mostrati in figura 3.11.

Nbit	1	2	4	8	16	32
K	256	128	64	32	16	8
n	2	4	16	256	65536	4G
	2^{256}	4^{128}	16^{64}	256^{32}	65536^{16}	$4G^{8}$
	2^{256}	2^{256}	2^{4*64}	2^{8*32}	2^{16*16}	2^{32*8}

Figura 3.11: Calcolo della complessitá di decodifica per $Kbit = 256$

Calcolando quindi la complessità, seguendo la precedente formula per quanto riguarda questo tipo di cifrario ($O(n^k)$), vediamo come il risultato sia sempre il medesimo, cioé 2^{256}.
Parte del problema comunque non era risolto, infatti la chiave deve essere un numero assolutamente casuale, non riconducibile a nessun evento, e inoltre piú tempo rimaneva attiva la stessa chiave, piú probabile era che venisse scoperta. Abbiamo perció pensato di generare una nuova chiave ogni ora e di trasmetterla codificata tramite la vecchia, inoltre tramite il registro a scorrimento a retroazione lineare (linear feedback shift register, LFSR) viene generata una chiave assolutamente casuale e non dipendente dalle precedenti.

3.4.1 Linear feedback shift register

Il registro a scorrimento a retroazione lineare (da ora in poi LFSR), é un tipo di registro a traslazione, i cui dati in ingresso sono prodotti da una funzione lineare dello stato interno (Fig. 3.12). Il valore iniziale di un LFSR si chiama seme, e poiché l'operazione del registro é deterministica, la sequenza di valori prodotta dal registro é completamente determinata dal suo stato corrente o precedente.
Allo stesso modo, poiché il registro ha un numero finito di stati possibili, prima o poi i valori in uscita si ripetono; ciò nonostante, un LFSR con una funzione di retroazione ben scelta può produrre una sequenza di bit che appare casuale ed ha un periodo molto lungo.

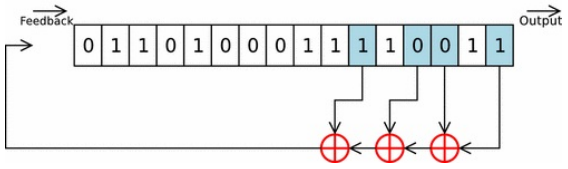

Figura 3.12: Funzionamento registro a scorrimento laterale

Applicazioni degli LFSR includono la generazione di numeri pseudo-casuali, sequenze pseudo-rumore (approssimazione del rumore bianco) e contatori digitali. Le modalitá in cui questo tipo di registro é stato utilizzato verranno meglio approfondite nel capitolo 4 sull'Implementazione.

Capitolo 4

Dall'Idea alla pratica, l'Implementazione

4.1 Implementazione

4.1.1 Codifica Manchester

Come specificato in precedenza, é necessario codificare i dati, e successivamente decodificarli, tramite una specifica codifica che si chiama Codifica Manchester. La particolaritá di questo tipo di codifica é il fatto che converte l' 1 in sequenza 01 e lo 0 in sequenza 10 (fig. 4.1). In questo modo é impossibile avere sequenze di piú di 2 bit con lo stesso valore.

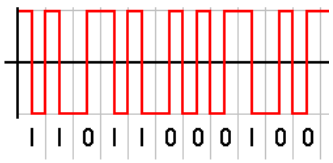

Figura 4.1: Codifica Manchester

Esistono molti algoritmi che implementano la Codifica Manchester. Quello che é stato scelto, prende come input la lunghezza del messaggio da codificare, utilizza poi due vettori, uno sorgente con il messaggio ancora decodificato, e uno destinazione in cui andrá messo il messsaggio una volta codificato.
Tramite un ciclo while che scorre tutta la lunghezza viene calcolato il valore in decimale corrispettivo e viene inserito nel bit giusto (Fig 4.2).

```
while(cnt<len){
        dst->c0 = 2 - (*src     & 1);
        dst->c1 = 2 - (*src>>1 & 1);
        dst->c2 = 2 - (*src>>2 & 1);
        dst->c3 = 2 - (*src>>3 & 1);
        dst->c4 = 2 - (*src>>4 & 1);
        dst->c5 = 2 - (*src>>5 & 1);
        dst->c6 = 2 - (*src>>6 & 1);
        dst->c7 = 2 - (*src>>7 & 1);
        dst++;
        src++;
    cnt++;
}
```

Figura 4.2: Codice Codifica Manchester

La decodifica é speculare alla codifica, infatti viene fatta l'operazione inversa su buffer codificato, e viene inserita mano a mano in quello decodificato.
La scelta dell'algoritmo é stata influenzata dalla necessitá di usare meno spazio in memoria possibile: infatti nonostante esistano metodi piú facilmente manutenibili e piú veloci, nessun altro metodo aveva un cosí basso utilizzo di memoria e di accessi alla memoria stessa.
Questo aspetto é fondamentale per il sistema utilizzato che ha risorse hardware molto modeste (48k di memoria Ram e 256K di memoria Flash), e quindi necessita di codice leggero.

4.1.2 Inizializzazione

Il main principalmente lancia due procedure. La InitApp(), e il MainCycle(), all'interno di un ciclo while(1) perché deve essere sempre in esecuzione.
La InitApp() [metodo all'interno di `user.c`] si occupa di inizializzare lapplicazione, lancia quindi a sua volta tutti i metodi di inizializzazione: (Fig. 4.3)

```
void InitApp(void){
        InitConfig();
        InitDoP();
        InitEusart(2*_MAX_DOP_SAMPLES,2*_MAX_DOP_SAMPLES);
        InitDAQ();
        InitDSP();
        InitTIMER();
}
```

Figura 4.3: Inizializzazione sistema

La InitConfig(), che si trova in `Command.c`, si occupa di controllare se la configurazione salvata é quella di default oppure quella personalizzata dallutente. Poi copia byte per byte la configurazione salvata in quella corrente. Se non é inizializzata genera quella di default, e infine aggiorna le soglie.

La InitDoP(), che si trova in `dop.c`, inizializza semplicemente i buffer e0 ed e1 (buffer per la codifica) e il buffer per la Decodifica.

La InitEusart(), che si trova in `eusart.c`, inizializza l'hardware per la trasmissione e ricezione seriale dei dati e i buffer circolari di trasmissione e ricezione.

La InitDAQ(), che si trova in `daq.c`, si occupa di preparare gli operazionali e i convertitori ADC e DAC per lacquisizione dei dati, settando i loro parametri e facendo la calibrazione.

La InitDSP(), che si trova in `dsp.c`, inizializza il sistema di elaborazione dei dati.

La InitTIMER(), che si trova in `timer.c`, si occupa di resettare a zero (tramite la ResetTime()) il sottosistema che viene utilizzato per sapere da quanto il sistema é in funzione.

4.1.3 Comunicazione

Finita linizializzazione di tutto il sistema si entra nel Loop del MainCycle(). Il MainCycle é la funzione principale che orchestra tutte le altre e si occupa di mantenere il sistema funzionante. Essa si trova in `user.c`, e principalmente tramite un *do while* rimane in ascolto dei comandi che possono arrivare. Quando MainCycle riceve un comando (quindi *cmdType != NOCMD*) entra nel corpo del ciclo e chiama la DOP_ReceiveCmd() (fig. 4.4).

```
do{
   DOP_ReceiveCmd();
   cmdType = DecodeCommand(&execFunc, &txrxFunc, &postFunc);
}while(cmdType == NOCMD);
```

Figura 4.4: Ricezione Comando

La DOP_ReceiveCmd(), si trova in `dop.c`, questa funzione abilita lhalfduplex alla ricezione, poi si mette in attesa della sequenza di start (*while != CMDSTART*); quando arriva la sequenza di start, esce dal ciclo while di attesa, ed entra in un secondo ciclo while, che continua a leggere il comando e lo salva nei buffer e0 ed e1 finché la sequenza non é quella di stop (*CMDSTOP*).
Tramite un contatore controlla sempre che la lunghezza ricevuta non superi mai la dimensione del buffer; nel caso questo succeda il ciclo while viene interrotto e, se ancora non é arrivato il comando di stop, il contatore viene messo a -1. In questo modo é pronto a scrivere di nuovo, ma senza ancora aver cancellato i dati precedenti.
Infine disabilita alla ricezione lhalfduplex e fa partire la decodifica Manchester sul comando ricevuto. La decodifica Manchester, come abbiamo giá visto, si occupa di scrivere il comando in chiaro nel DecodeBuff, e ritorna il contatore di caratteri, oltre a riempire il buffer col comando decodificato. In questo modo é possibile sapere la lunghezza del comando ricevuto.

Terminata la sequenza di ricezione del comando, il controllo viene ridato al MainCycle(), che procede con la lettura del comando appena ricevuto, invocando la DecodeCommand() (fig. 4.4).

La DecodeCommand() si trova in `command.c` e si occupa di dare un'interpretazione al comando che é stato ricevuto e decodificato.
Per prima cosa controlla che non ci siano comandi sbagliati, quindi che la lunghezza non sia inferiore a 2 byte (in questo caso ritorna un *NOCMD*).
Stabilito che il comando non é sbagliato, si assicura che sia un comando destinato a quel nodo specifico (se lindirizzo del buffer decodificato é diverso dallindirizzo del nodo stesso) e controlla che non sia un comando di BROADCAST. Nel caso tutti questi controlli vadano a buon fine, viene distinta la tipologia di comando in base al numero di byte.
Mano a mano che il comando viene decodificato, viene salvato allindirizzo di exexFunc() il puntatore alla funzione che esegue il comando, all'indirizzo di postFunc() la funzione che esegue il codice successivo alla conclusione del comando, all'indirizzo di TxRxfunc() il puntatore alla funzione che gestisce la trasmissione e la

ricezione dei dati; infine ritorna al chiamante il tipo di comando (fig. 4.5).

```
if (len==2){
        if ((cmdcode_t)DecodedBuff[1] == CMDGO){
                *postFunc = AcquireData;
                return ((*DecodedBuff == BROADCAST) ? BCMD : SCMD);
    }
    if ((cmdcode_t)DecodedBuff[1] == CMDSYNC){
                *postFunc = AcquireData;
                return ((*DecodedBuff == BROADCAST) ? BCMD : SCMD);
        }

.....
```

Figura 4.5: Esempio Interpretazione Comando

Tornando quindi al ciclo principale, il MainCycle(), é stato ricevuto il comando, é stato decodificato e interpretato. Abbiamo quindi la certezza di quale comando si tratta, e le sue specifiche funzioni sono giá note.
A questo punto termina il ciclo di attesa del comando e viene eseguita la execFunc(), che ha giá nel proprio indirizzo di memoria quale procedura avviare (per esempio acquisizione, reset, invio, settaggio di indirizzi...).
Il mainCycle(), prima di terminare, esegue uno switch che controlla il tipo di comando ricevuto.

I possibili tipi sono:

- Broadcast Command (BCMD)
- Long Command (LCMD)
- Short Command (SCMD)
- Short Query (SQRY)
- Long Query (LQRY)
- Nessun Comando (NOCMD)

In base al tipo di comando deve inviare una risposta coerente al mittente del comando stesso.
Solo nel caso di Broadcast non deve esserci nessuna risposta: infatti il broadcast

viene utilizzato per consegnare lo stesso pacchetto a tutti i nodi connessi alla rete (metodo che viene utilizzato, ad esempio, per la sincronizzazione degli orologi interni ai nodi).
Nel caso dello SCMD va solamente rispedito un Acknowledge (ACK) che comunica lavvenuta ricezione.
Nel caso del LCMD, va comunque inviato un Ack di avvenuta ricezione, peró poi viene avviata la DOP_ReceiveData() (all interno di `dop.c`) che abilita alla ricezione lhalfduplex e inizia a leggere i dati in ricezione fino a quando non arriva la sequenza di stop.
Terminata la ricezione il controllo viene restituito al MainCycle(), che invia un ACK di trasmissione dati.
Nel caso della SQRY, viene inviata la risposta richiesta (tramite la DOP_TransmitCmd() in `dop.c`) e poi si resta in attesa dellACK di avvenuta consegna del Comando.
La DOP_TransmitCmd() scrive nel buffer i dati che vengono poi letti finché non arriva la sequenza di stop.
Nel caso della LQRY, come nel caso precedente viene inviata la risposta, dopodiché si resta in attesa in attesa dell' ACK di avvenuta consegna del comando. Come nella LCMD tramite un ciclo while() vengono trasmessi i dati (tramite la DOP_TransmitData()) e ogni volta si attende l'ACK di avvenuta consegna. Terminata la trasmissione tramite un break il controllo viene restituito al MainCicle().

4.2 Crittazione

In precedenza é giá stato illustrato il problema della sicurezza e della necessitá di criptare i dati trasmessi, la soluzione adottata in questo sviluppo utilizza tre classi. Ognuna di queste classi si occupa di uno specifico compito necessario a cifrare, e successivamente decifrare le informazioni che devono essere trasmesse.
La `key_generator.c`si occupa di generare una chiave random diversa ogni volta, la `crypto.c`invece, prende come parametro la chiave generata da `key_generator.c`e i dati da trasmettere in chiaro, dopodiché fa lo XOR tra i due e restituisce il `chipertext`. Il messaggio é cosí elaborato é pronto per essere trasmesso, una volta arrivato a destinazione tramite la `decrypto.c`, che conosce la chiave di cifratura, viene fatta l'operazione inversa per risalire al `plaintext`.
Per aumentare la sicurezza, come avevo giá accennato nel capitolo 3, la chiave viene generata nuova ogni ora, e viene trasmessa da `crypto.c`a `decrypto.c`, cifrata tramite la chiave precedente ed é riconoscibile da un pacchetto ordinario grazie a un comando speciale.
`Decrypto.c`salva dunque la nuova chiave che utilizzerá per decifrare i pacchetti in transito futuri.

Questo meccanismo permette di avere sempre una chiave fresca e diminuire quindi le probabilitá di successo per gli attacchi brute force.

Andando ad analizzare piú nello specifico il funzionamento di queste classi: La `key_generator.c`é composta da 5 funzioni. Quella principale, che da il nome anche alla classe, si appoggia ad una funzione ausiliaria, la initl_fsrs(void), questa inizializza l'algoritmo lfsr impostando i valori di seed, cioé i semi sopra i quali verrá generata la chiave. Il primo seed é 0xABCDE, mentre il secondo viene generato casualmente tramite la hex():

```
int hex(void){
        int valore;
        int n=10,i;
        bool arr[65535]={0};
        time_t t;
        srand((unsigned)time(&t));
        for(i=0;i<n;++i){
        int r=rand()%65535;
        if(!arr[r]){
                if(n=5){
                                        valore=r;
                        }
                }
        else
                i--;
        arr[r]=1;
        }
        return valore;
}
```

Figura 4.6: Generazione numero esadecimale random

Successivamente la hex() verrá modificata, per la scelta della chiave dall'elenco di quelle generate al momento usa la funzione base di C srand(), che peró eseguita ogni ora nello stesso momento rischia di ripetere alcune scelte, la modifica prevede di usare come numero casuale non quello prodotto da srad() ma il valore di una tensione di riferimento interna al nodo in quel preciso momento, valore molto piú variabile e incerto se letto con la giusta sensibilitá perché alterato da fenomeni

totalmente random come il rumore flicker ed il rumore termico, tipici dei dispositivi a semiconduttore.

```
int get_random(void){
        shift_lfsr(&lfsr32 ,POLY_MASK_32);
    return(shift_lfsr(&lfsr32 ,POLY_MASK_32)
                ^shift_lfsr(&lfsr31 ,POLY_MASK_31))&0xffff;
}
```

Figura 4.7: Esecuzione lsfr

Una volta inizializzata la lfsr, il controllo ritorna alla keygenerator(), che chiama la getrandom(). Questa funzione si occupa di eseguire le istruzioni dell'algoritmo lsfr(Fig 3.12). Infatti esegue lo shift_lfsr() con i seed generati in precedenza:

```
int shift_lfsr(uint32 *lfsr, uint32 polynomial_mask){
        int feedback;
        feedback=*lfsr &1;
        *lfsr >>1;
        if (feedback == 1)
                *lfsr ^= polynomial_mask;
        return *lfsr;
}
```

Figura 4.8: Algoritmo lsfr

Una volta che la chiave é stata generata viene passata a `crypto.c`. Crypto prende come parametri tre vettori, il plaintext (uint8 PlainText[LENGTH]), la chiave e un vettore vuoto chiamato chiper in cui andrá a mettere il testo cifrato. Principalmente per eseguire la crittografia scorre per tutta la loro lunghezza i tre vettori, che hanno uguale lunghezza definita da LENGHT (definita come multiplo di quantità a 32 bit) e ad ogni giro mette in chipertext lo XOR corrispondente di quella posizione tra il Plaintext e la Key.

Il messaggio da trasmettere che rappresenta in questo momento l'informazione relativa ad un urto, é quindi stato criptato, codificato tramite la codifica Manchester ed é quindi pronto per essere mandato al centro di elaborazione che si occuperá di decodificarlo di interpretarlo e di utilizzare l'informazione trasmessa.

Il ricevitore deve quindi sapere come decodificare il messaggio, ma come ho giá specificato precedentemente, la chiave nuova viene trasmessa ogni ora criptata

tramite la chiave vecchia.
In ogni momento il ricevitore sará quindi in grado di decifrare ogni messaggio,dopo avere fatto la decodifica Manchester, facendo l'operazione inversa a quella fatta per criptare. Ovvero un operazione di XOR tra il Chipertext (messaggio tramsmesso) e la Key (ricevuta in precedenza). Il risultato di questa operazione sará quindi l'informazione pulita pronta per essere utilizzata.

Capitolo 5

Testing e Risultati

5.1 Testing e analisi del sistema

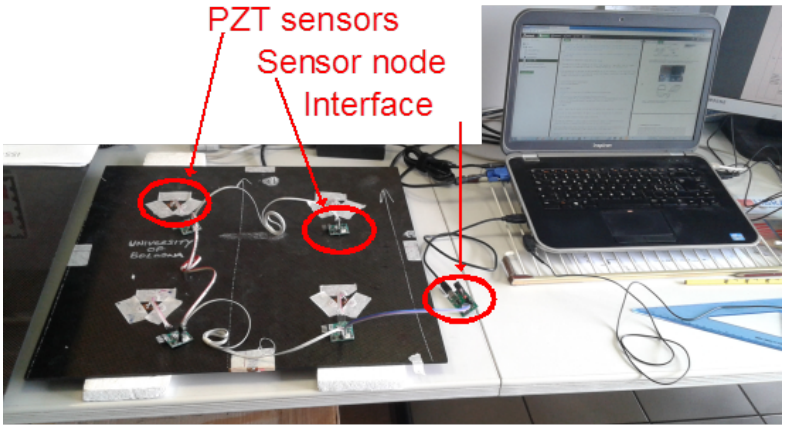

Figura 5.1: Collegamenti sensori in fase di Testing

Per la fase di testing, visto che non era possibile equipaggiare un aereo con la rete di sensori a causa delle certificazioni necessarie è stato necessario avvalersi di strutture di supporto. Come si può vedere in figura 5.1, sono state utilizzate mattonelle in carbonio di dimensione 50 × 50cm, derivate dal progetto Europeo SARISTU [10], costruite con gli stessi standard utilizzati per la produzione di materiale ad uso aeronautico su cui sono stati posizionati i sensori, i relativi nodi sensori, il cablaggio connesso in Data Over Power, il gateway che connette i nodi sensori al PC, e il tutto é stato collegato ad un PC per rilevare i dati trasmessi. Sollecitando con diversi tipi di urto la superficie controllata è possibile vedere, tramite un apposito software su PC, i dati trasmessi e l'effettivo funzionamento di

tutto il sistema.

Figura 5.2: ST-Link Debugger and flasher programmer

Per avere peró delle simulazioni di diverso tipo, per poterle modificare e personalizzare in modo da riuscire ad entrare nello specifico di ogni possibile caso di rilevazione, ci siamo avvalsi di strumenti di simulazione.
Il tipo di collegamento che abbiamo realizzato per questa fase di testing prevede l'utilizzo del ST-LINK (Fig. 5.2), un debugger di circuito e programmatore per le famiglie di microcontrollori STM8 e STM32 utilizzato per comunicare con qualsiasi microcontrollore STM8 o STM32 localizzato sulla scheda di applicazione. Presenta un isolamento digitale tra il PC e la scheda di applicazione target che sopporta inoltre tensioni fino a 2500 VRMS.
Tramite questo debugger é stato collegato il nodo sensore al pc, a sua volta il nodo sensore é connesso con il gateway che anch'esso viene collegato al pc tramite porta USB come visibile in Fig. 5.3.

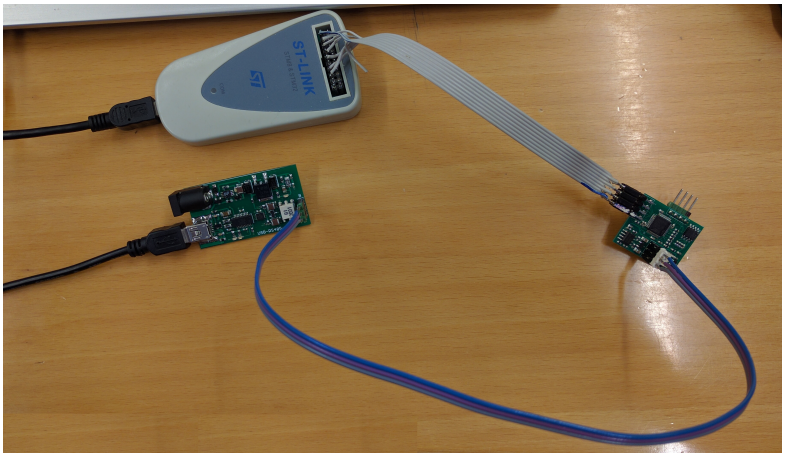

Figura 5.3: Collegamenti sensori e debugger in fase di Testing

Grazie a questo collegamento simultaneo al PC é possibile avviare i due software utilizzati per la programmazione dei sensori e testare l'invio e la ricezione dei dati come in un ambiente reale.
Il primo software che viene utilizzato per la programmazione del nodo sensore stesso é Atollic TrueSTUDIO for ARM, un Plugin del famoso IDE Eclipse che permette la programmazione in C/C++ dei chip utilizzati per la realizzazione dei sensori.
L'altro software, che utilizziamo solamente in fase di testing per simulare situazioni reali e per interrogare i nodi (tramite il gateway) ed elaborarne i risultati é il noto programma per simulazione di analisi matematica MATLAB^{TM}.
MATLAB^{TM} (abbreviazione di Matrix Laboratory) è un ambiente per il calcolo numerico e l'analisi statistica scritto in C e Java, che comprende anche l'omonimo linguaggio di programmazione creato dalla MathWorks. MATLAB^{TM} consente di manipolare matrici, visualizzare funzioni e dati, implementare algoritmi, creare interfacce utente, e interfacciarsi con altri programmi.
Utilizzando MATLAB^{TM} infatti é stato possibile scrivere diversi script che quando eseguiti simulano ed eseguono operazioni direttamente sul nodo sensore.
Alcuni esempi sono:

- cmdReset_v2.m
- getImpacts_v2.m
- selfTest_v2.m
- setAddr_v2.m
- setTrgMode_v2.m

La cmdReset serve per resettare i parametri del nodo sensore nello stato di Default, la getImpacts invece necessita dei sensori collegati al nodo, e richiede il valore che sta leggendo in quel momento per rilevare possibili impatti. La selfTest, quando i sensori sono collegati al nodo sensore, serve per calcolare la capacitá dei sensori stessi: é un particolare tipo di test per capire se i sensori sono collegati correttamente o meno, infatti il valore capacitivo cambia se i sensori non sono collegati al proprio nodo in maniera corretta.
La setAddr serve per cambiare l'indirizzo con cui il nodo é identificato all'interno della rete. Quindi viene utilizzata in fase di costruzione e di inizializzazione della rete per dare ad ogni nodo sensore la propria numerazione all'interno del sistema. Infine la setTrgMode viene usata per attivare la modalitá free run, ovvero una modalitá in cui il nodo sensore risponde subito inviando i dati rilevati senza aspettare il trigger.
Ognuno di questi script viene lanciato quando il sensore é attivo, e manda dati e riceve comandi tramite pacchetti come illustrato nei capitoli precedenti.

Ognuna di queste esecuzioni completa il ciclo di:

- Ricezione del comando
- Decodifica Manchester
- Decrittazione del comando
- Recupero delle informazioni richieste dal comando
- Crittazione dei dati da inviare
- Codifica Manchester del chipertext
- Invio dei dati criptati
- Ricezione dei dati
- Decodifica Manchester
- Decrittazione del chipertex
- Interpretazione dei dati

I risultati pervenuti da entrambe le parti (master che invia comandi e slave che risponde al comando) sono attendibili e rappresentano quello che ci aspettavamo, come ad esempio il testVref_v2, al termine della sua esecuzione riporta la misura di tensione di riferimento per tutti i dispositivi all'interno del chip.

Della correttezza di questo dato siamo certi in quanto é il costruttore stesso a stabilire quali sono i valori di tensione di riferimento dei propri chip (all'interno di una certa tolleranza), i due valori, cioé quello rilevato e trasmesso e quello stabilito dal costruttore coincidono. La comunicazione ha quindi un esito positivo.

5.2 Testing e analisi della crittazione

Una volta testato tutto il sistema, e verificato il suo effettivo funzionamento l'attenzione si é spostata sull'analisi della crittazione e sulla sua affidabilitá.

Per fare questo tipo di test é stato utilizzato lo script selfTest, che al suo interno chiama la readDat (Fig. 5.4), una funzione che si occupa di leggere i campioni prodotti dal sensore a seguito di una Long Query. In seguito all'esecuzione dello script il nodo sensore produce una variazione della tensione di polarizzazione dei trasduttori piezoelettrici e registra un numero prefissato di campioni (2000); la successione temporale di questi campioni viene comunicata al gateway come una forma di debug hardware in quanto la forma d'onda acquisita è strettamente correlata alla capacità attuale del trasduttore connesso al nodo. La comunicazione procede secondo la sequenza già illustrata di impacchettamento dei dati, crittazione, codifica Manchester ed invio sul canale Data-Over-Power.

```
function out = readDat(s,timeout)
    if ((nargin<2) || isempty(timeout)), timeout = -1; end
    % Load command mnemonics
    commands;
    key=[];
    load lastkey;
    % Read the buffer looking for the data start symbol
    while(1)
        tmp = readChar(s,timeout);
        if (tmp < 0), out = -1; return, end;
        if (tmp == DATSTRT), break, end;
    end
    % Initialize data buffer
    dat = zeros(1,MAX_DOP_SAMPLES*2);
    idx = 0;
    % Read the data contents
    tmp = readChar(s);
    while(tmp ~= DATSTOP)
        dat(idx+1) = tmp;
        dat(idx+2) = readChar(s);
        tmp = readChar(s);
        idx = idx+2;
    end
    % Trim the data buffer
    out = decode(dat(1:idx));
    out = crypto(out,key);
end
```

Figura 5.4: readDat

Abbiamo salvato i valori in transito per questa comunicazione su di un file esterno in modo da poterli analizzare al meglio. Questo file contiene i valori di input (PlainText), la chiave e i valori di output (ChiperText).
Il chipertext si presenta come una sequenza di 0 ed 1 che agli occhi umani appare casuale: per verificare peró che nemmeno una macchina possa riuscire ad interpretare questa sequenza abbiamo provato a processarla tramite diversi tipi di algoritmi di decifratura e su di essa sono stati provati i due tipi di attacchi piú diffusi, ovvero brute force e attacco di tipo statistico.

5.2.1 Attacchi Brute Force

Gli attacchi brute force sono falliti, e la loro inadeguatezza nella decodifica di queste stringhe si puó facilmente dimostrare con alcuni passaggi matematici.

Supponendo di avere a disposizione il migliore supercomputer esistente sul pianeta (preso per riferimento il supercomputer annunciato dall'India in attivazione nel 2018) con una velocitá di calcolo di 132 ExaFLOPS, ovvero 132x10^{18} operazioni al secondo, si puó calcolare che con una chiave da 256 bit utilizzando 2 simboli si hanno 2^{256} possibili combinazioni, ovvero $1.15 \cdot 10^{77}$.

Il tempo necessario per testare tutte le possibili combinazion si ottiene dal rapporto tra il numero di combinazioni e la potenza di calcolo della macchina che fa l'analisi, ovvero

$$\frac{1.15 \cdot 10^{77}}{1.32 \cdot 10^{20}} \simeq 8.77 \cdot 10^{56}$$

Per comprendere meglio la durata temporale di questo processo riporto l'etá dell'universo stimata in secondi:

$$13.82 \cdot 10^{9} \cdot 365.25 \cdot 24 \cdot 60 \cdot 60 \simeq 4.36 \cdot 10^{17}$$

ovvero circa 436 milioni di miliardi di secondi. Il numero calcolato per la riuscita di un attacco brute force é di diversi ordini di grandezza superiore a questo.

5.2.2 Attacchi ad Analisi Statistica

Questo tipo di attacco é piú mirato e piú pericoloso rispetto al brute force, infatti partendo dal pacchetto criptato, che puó essere sniffato dalla rete, si cerca di risalire alla chiave analizzandone simboli, ripetizioni di singoli o piú caratteri e incrociando la frequenza di questi spesso si riesce a risalite alla chiave di cifratura [2].

La forza del Cifrario Vigenère è che la stessa lettera può essere criptata in modi diversi. Ad esempio, se la parola chiave è KING, allora ogni lettera in chiaro puó essere crittografata in 4 modi, perché la parola chiave contiene 4 lettere. Ogni lettera della parola chiave definisce un diverso alfabeto cifrato nella Tabella di Vigenère.
La colonna 'E' della tabella (esempio in Fig. 5.5)

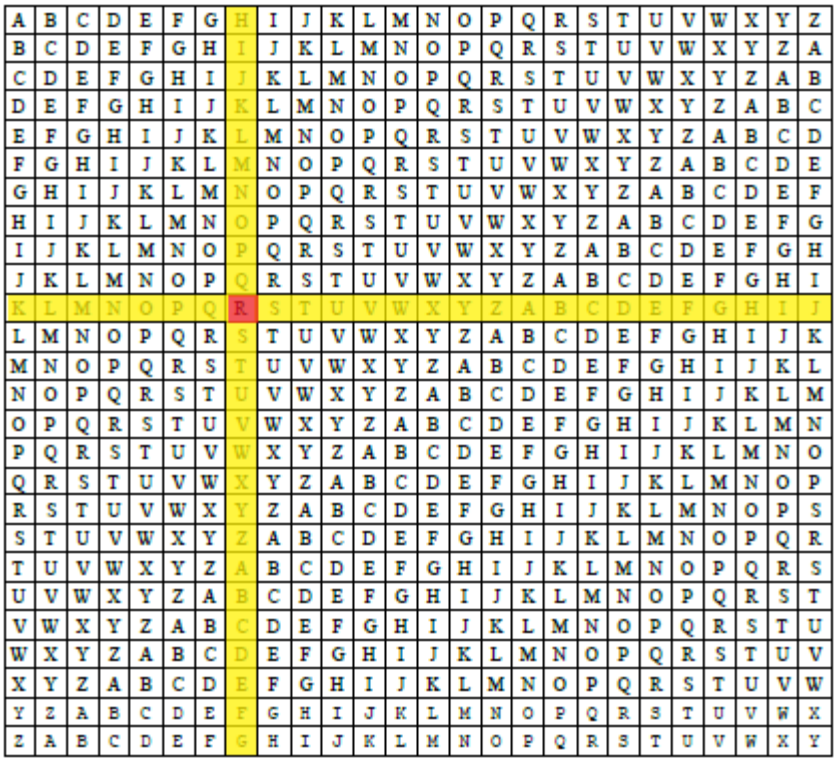

Figura 5.5: Tabella Vigenère

puó cifrare quella lettera con altre 23, a seconda di quale é la corrispondente lettera nella chiave. Allo stesso modo, intere parole saranno cifrate in modi diversi. La parola 'the', considerando l'alfabeto inglese, potrebbe essere cifrata come DPR, BUK, GNO e ZRM a seconda della sua posizione rispetto alla parola chiave.
Il punto importante da notare è che se ci sono solo quattro modi per cifrare la parola 'the', e il messaggio originale contiene diversi usi della parola 'the', allora è inevitabile che alcune delle quattro possibili cifrature di quella parola si ripeteranno nel testo cifrato.

Babbage ebbe l'intuizione che ripetizioni nel testo cifrato indicassero ripetizioni nel testo in chiaro, e che lo spazio tra queste ripetizioni indicasse di conseguenza la lunghezza della parola chiave.
Questo é il principio di analisi statistica che viene utilizzato per forzare il cifrario di Vigenére, nella fase di testing siamo peró riusciti a dimostrare che questa decodifica con il nostro tipo di algoritmo non funzionerá.
Infatti tendenzialmente nei cifrari la chiave é molto piú corta del testo che si vuole trasmettere, quindi a un certo punto la chiave verrá riutilizzata dall'inizio per diverse volte: nel nostro caso questa situazione non si verifica mai, infatti la

chiave utilizzata da 256bit non viene mai usata completamente piú di una volta per trasmettere una messaggio, quindi nel messaggio cifrato non saranno mai presenti due ripetizioni complete della stessa chiave. Inoltre essendo una sequenza di simboli che sono solamente 0 e 1 l'analisi descritta in precedenza troverá una quantitá enorme di ripetizioni uguali e di stringe ripetute: questo inganna il tipo di analisi che potrà solo con difficoltà scoprire che la chiave è 256bit, perché la probabilitá piú alta rilevata é che la chiave sia composta da solamente 2-3 lettere. Riporto qui sotto (Fig. 5.6) un esempio di uno dei numerosi test di decrittazione che sono stati eseguiti sul chipertext, l'alfabeto utilizzato é composto da due simboli, A che rappresenta il numero binario 0 e B che rappresenta il numero binario 1.

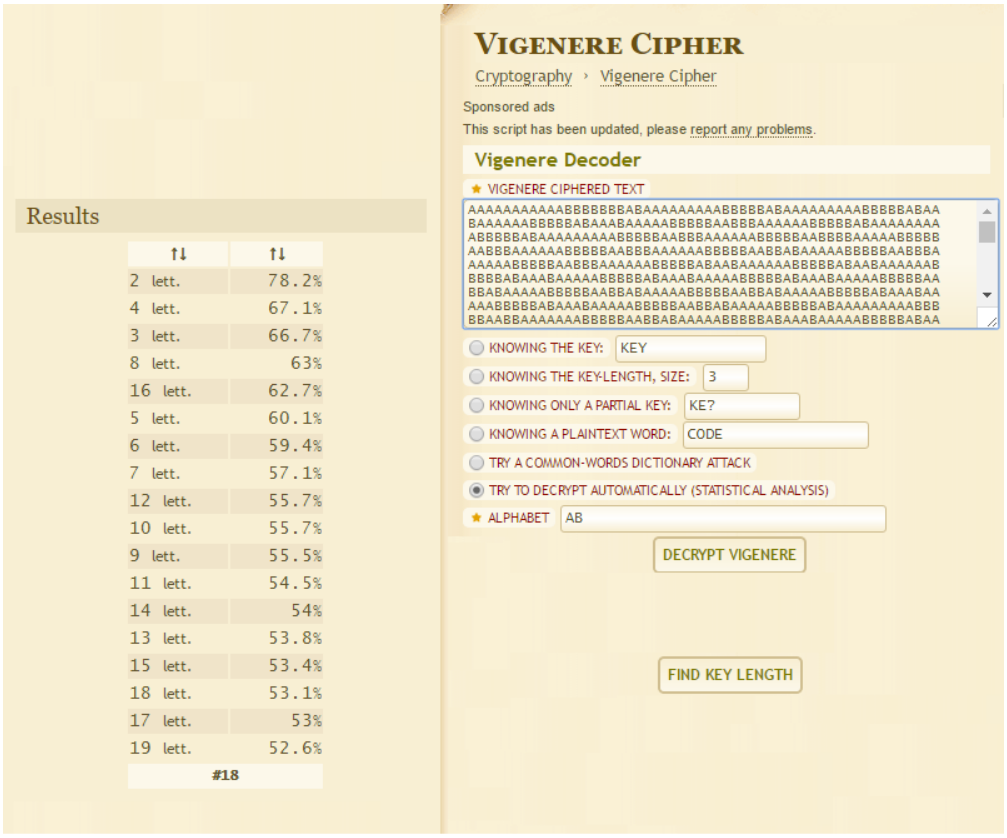

Figura 5.6: Tentativo di decrittazione

Come si puó osservare dalla figura, inserendo un messaggio trasmesso dai nostri sensori criptato, e chiedendo all'algoritmo di risalire alla lunghezza della chiave, questi risponde con una probabilitá del 78% che la chiave sia composta da 2 lettere: questo é dovuto al basso numero di simboli utilizzati con una chiave molto lunga,

infatti sappiamo bene che la lunghezza della chiave non é 2 ma bensií 256.

Tramite uno script in MatLab é stata eseguita un'ulteriore analisi di una stringa criptata: ne viene analizzata ogni occorrenza di simboli ripetuti e, come suggerisce la tecnica di decifratura di Vigenére, ad ogni ripetizione si contano quanti caratteri stanno nel mezzo: i divisori interi di quel numero sono le possibili lunghezze della chiave.
Al termine dell'analisi del testo cifrato il programma stampa un grafico (fig. 5.7) che rappresenta in numero di occorrenze di ogni divisore, per vedere se é riuscito ad avvicinarsi alla lunghezza reale della chiave.

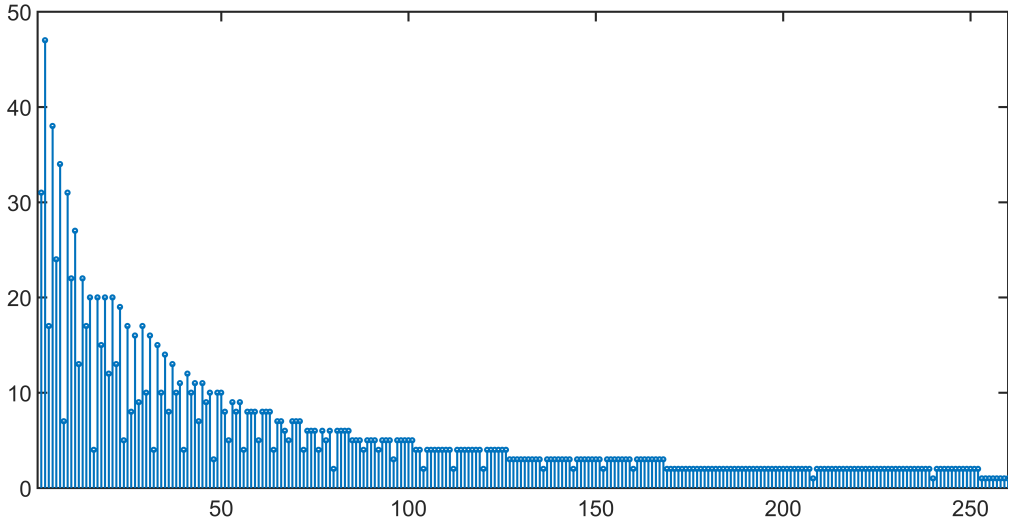

Figura 5.7: Analisi statistica delle possibili lunghezze della chiave

Come possiamo vedere dal grafico anche l'analisi di tipo statistico, dopo quella di tipo brute force, fallisce nel tentativo di decriptazione: infatti come lunghezza piú probabile della chiave risulta una sequenza di 3 simboli e la probabilitá che sia 256 é molto prossima allo zero.

5.3 Sviluppi futuri

Al momento attuale questa comunicazione, come é stato illustrato, risponde bene agli attacchi di tipo COA (ciphertext-only). Un solo tipo di comunicazione peró é soggetto a una piú elevata probabilitá di decrittazione, quella che da il comando di reset. Infatti essendo di un solo byte ha solamente 256 possibili combinazioni.
Per risolvere questo problema verrá concatenato in coda un altro pacchetto contente la chiave attuale (ovviamente cifrata) che aumenta quindi la dimensione della

comunicazione e quindi il numero di possibili combinazioni rendendola uguale alle altre, e inoltre puó essere usata come controllo.

La valutazione di un sistema di cifratura si fa peró non solo sui COA come abbiamo analizzato in questo capitolo, ma anche su altri tipi di attacchi. I più frequenti sono i known-plainText (KPA) e i chosen-plainText (CPA).

In un attacco di tipo KPA (fig. 5.8) l'aggressore viene a conoscenza di una coppia plainText-cipherText e cerca di risalire alla chiave per decodificare futuri messaggi di cui non hai il plainText.

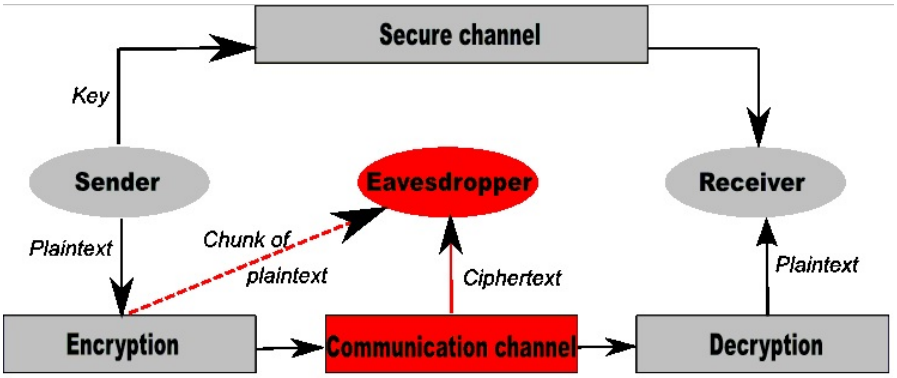

Figura 5.8: Funzionamento attacco KPA

In un attacco di tipo CPA (Fig. 5.9) l'aggressore é in grado di imporre il plain-Text (o una sua porzione) e osservare il corrispondente chiperText per risalire alla chiave.

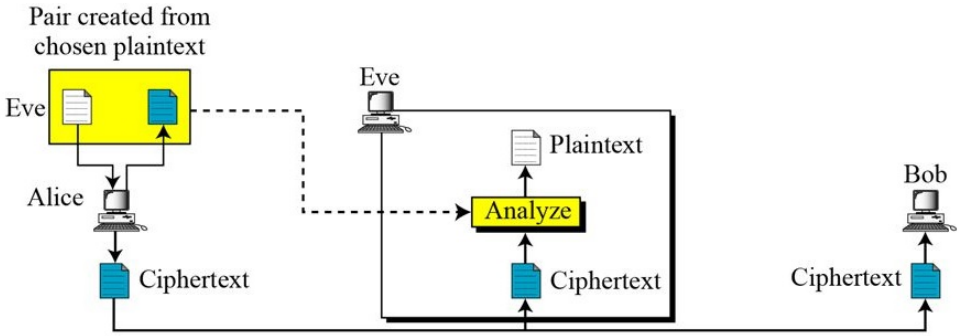

Figura 5.9: Funzionamento attacco CPA

I prossimi sviluppi di questo algoritmo di crittazione saranno rivolti a ridurre il rischio questo tipo di minacce, una possibile soluzione (ancora da testare e verificare) puó essere quella di cambiare la chiave ad ogni comunicazione, senza trasmetterla mai.
Questa procedura é attuabile leggendo la tensione di riferimento interna al nodo in quel preciso momento, come stiamo facendo giá ora per inizializzare l'lfsr; questo valore é soggetto ad un rumore e non é pertanto mai uguale a quello precedente nei valori delle cifre meno significative. Possono quindi essere usate queste cifre sempre diverse ed assolutamente casuali per reinizializzare l'lfsr (vedi capitolo 4) sul trasmettitore e sul ricevitore ad ogni comunicazione: in questo modo la chiave di crittografia sará sempre nuova e sempre diversa.
La tensione di riferimento rilevata dal nodo sensore potrebbe venire allegata al pacchetto trasmesso in coda ad ogni comunicazione, in modo che il ricevitore sappia esattamente dove si trovano i bit che contengono il valore della tensione di riferimento. Anch'essa dovrebbe essere ovviamente criptata, ed una volta decodificata verrà utilizzata per inizializzare l'lfsr del trasmettitore. Affinché questo tipo di comunicazione avvenga con successo, è assolutamente fondamentale che il trasmettitore ed il ricevitore siano perfettamente sincronizzati, altrimenti si rischierebbe di cifrare-decifrare uno stesso messaggio con due chiavi diverse.
Attuando questa soluzione il possibile aggressore della rete perderebbe ogni riferimento fra chiperText e plainText: infatti anche intercettando una comunicazione e decodificando la chiave attuale, quella chiave non sarebbe piú utilizzabile per la successiva comunicazione perché verrá cambiata dalla generazione di una nuova sequenza degli lfsr interni, inizializzati con lo stesso seed, sempre diverso e casuale.

Capitolo 6

Conclusioni

In uno scenario globale dove il tempo é sempre piú prezioso, e dove ogni azione viene gestita in maniera smart e intelligente, questo lavoro cerca di allineare a questi standard anche un ambito come quello della rilevazione e della prevenzione dei danni, che al momento spesso si trova a rimanere bloccato in colli di bottiglia. Ne sono un esempio la necessitá di un ispezione ed un intervento umano che spesso richiede tempi lunghi, e valuta la qualitá dell'intervento solamente sulla base delle competenze dei tecnici specializzati, che nonostante tutto restano esseri umani e quindi capaci di dare interpretazioni diverse agli stessi dati.
Per fare fronte a tale necessitá nasce Intelligent Structures, uno studio di ricerca universitario che ha progettato, sviluppato ed implementato un sistema che si pone come obiettivo quello di rendere intelligenti le strutture su cui viene applicato.
Per strutture intelligenti si intendono elementi fisici che sono in grado di autodiagnosticarsi e di capire in autonomia quale è il loro stato di salute attuale, quali sono le zone interessate da eventuali guasti o danneggiamenti, e fornire indicazioni riguardo la posizione, l'entità del danneggiamento, il momento in cui questo è avvenuto e l'intesità dell'urto che ha provocato il problema.
Grazie ad una rete di sensori piezoelettrici studiati e sviluppati internamente ai laboratori di ricerca dell'Università di Bologna, qualunque struttura sarà in grado di comunicare in maniera efficiente il proprio stato attuale di salute.
Questo progetto fornisce inoltre la possibilità per il gestore della struttura monitorata di avere notifiche in real-time su eventi avvenuti in ogni istante grazie ad una piattaforma Android [7] sviluppata simultaneamente, che carica un modello 3D della struttura su cui sono applicati i sensori e posiziona dei marker rappresentanti i danneggiamenti. Interagendo con questi marker è possibile verificare lo stato di salute del sistema.
Un altro aspetto molto importante per lo sviluppo di questo progetto è stato ridurre al minimo l'assorbimento di energia da parte di tutto il sistema, infatti non avendo una batteria propria si appoggia ai generatori della struttura che lo ospita,

e non deve quindi andare ad influire sul suo ordinario funzionamento.
Per fare fronte a questo problema sono state utilizzate diverse strategie, dalla comunicazione in modalità Data Over Power alla scelta accurata dei componenti elettronici che compongono l'hardware della rete, tali da fornire le funzionalità richieste dal sistema anche a fronte di un basso consumo energetico.
In questo modo la rete è applicabile su qualunque struttura senza necessità di avere una propria batteria (che andrebbe periodicamente cambiata) e può funzionare in ogni momento senza influire in nessun modo sul funzionamento della struttura stessa.
Un'altra parte importante di questo progetto è quella che riguarda le comunicazioni, che a causa dell'elettronica minimale volta a rispettare il vincolo dei bassi consumi, devono essere molto leggere e veloci; per questo motivo ogni parte del codice che realizza lo scambio di dati tra sensori, gateway e dispositivo di controllo è stata pensata per essere efficiente e per utilizzare la minor quantità di memoria, sia fisica che volatile, possibile.
Per rendere sicura tutta la comunicazione, e ridurre quindi il rischio di attacchi volti ad alterare i contenuti dei dati ed a creare quindi falsi positivi/negativi per quello che riguarda lo stato di salute, sono stati implementati degli algoritmi di crittografia a chiave privata (vedi capitoli 3 e 4) che permettono una comunicazione più sicura basandosi sulla nota architettura lfsr e sfruttandola per generare ad ogni iterazione una chiave da 256 bit risultante mediamente sicura per questo tipo di comunicazione.
Come illustrato in precedenza questo algoritmo ha ancora delle falle, perché pur avendo un ottima efficienza sugli attacchi ChiperText-Only ha ancora un tasso di vulnerabilità sugli attacchi known-plaintext (KPA) e i chosen-plaintext (CPA), due minaccie con pericolosità crescente. Soluzioni per sopperire a questa carenza sono note in letteratura, una delle quali è stata illustrata negli sviluppi futuri previsti (vedi capitolo 5.3). L'obiettivo è quello di raggiungere un livello di sicurezza che rispetti il principio di Kerchoff, cioè un sistema è ritenuto sicuro quando resiste agli attacchi di qualcuno che conosce tutto dei dettagli implementativi del cifrario, ovvero come funziona l'algoritmo, tranne la chiave.
All'interno di questo elaborato mi sono maggiormente soffermato su un possibile utilizzo areonautico di questo sistema, perché è l'ambito da cui ha avuto origine l'idea, ma in realtà la sua modularità e scalabilità la rendono applicabile a qualsiasi struttura.
Basti pensare infatti a quanto può risultare utile se applicato in ambito automobilistico, su auto da corsa, soggette a continui stress e innumerevoli urti durante una competizione, oppure se inserito in oggetti di produzione industriale, che necessitano di eventuale assistenza solamente su autorizzazione della casa produttrice. Questo sistema potrebbe infatti prevenire le sempre piú numerose riparazioni "fai

da te" che possono mettere a rischio la salute di chi usa un dispositivo riparato in casa e quindi non piú conforme agli standard. Ancora, se applicato a bombole di gas, potrebbe contribuire a prevenire eventuali esplosioni o perdite. Infine, ma non per importanza, questo sistema può essere applicato a strutture edili ed architettoniche, come ponti, palazzi, edifici storici e contribuire a rendere più sicuri i posti in cui viviamo e lavoriamo.

Ringraziamenti

Voglio ringraziare il Professor Fabio Panzieri per il prezioso sostegno datomi nella conclusione dei miei studi, il Professor Nicola Testoni per la pazienza e l'entusiasmo che ci ha messo nell'accompagnarmi in questo percorso. Ringrazio la mia famiglia, Marco, Antonietta e Valentina per avermi dato l'opportunitá di seguire i miei sogni e sviluppare la mia passione. Ringrazio infine Costanza e tutti i miei amici, fondamentali compagni di viaggio, la Social, mia seconda casa, per la pazienza, il sostegno e l'aiuto che mi hanno dato in questi anni dedicati al mio percorso universitario.

Bibliografia

[1] 787 no-bleed systems: Saving fuel and enhancing operational efficiencies. http://www.boeing.com/commercial/aeromagazine/articles/2015_q1/archive.html. Accessed: 2016-12-01.

[2] The black chamber. http://www.simonsingh.net/The_Black_Chamber/crackingprinciple.html. Accessed: 2016-12-01.

[3] Networking & communications mica: The commercialization of microsensor motes. http://www.sensorsmag.com/networking-communications/mica-the-commercialization-microsensor-motes-1070. Accessed: 2016-12-01.

[4] Tinyos. https://github.com/tinyos/tinyos-main. Accessed: 2016-12-01.

[5] D. Gao, Y. Wang, Z. Wu, G. Rahim, and S. Bai. Design of a sensor network for structural health monitoring of a full-scale composite horizontal tail. *Smart Materials and Structures*, 23(5):055011, 2014.

[6] M. Ilyas and I. Mahgoub. *Smart Dust: Sensor network applications, architecture and design*. CRC press, 2016.

[7] F. Negretti. Interfaccia interattiva per la conversione e visualizzazione di strutture 3d. Technical report, Università di Bologna, Scuola di Scienze Naturali, Settembre 2016.

[8] W. Staszewski, S. Mahzan, and R. Traynor. Health monitoring of aerospace composite structures–active and passive approach. *Composites Science and Technology*, 69(11):1678–1685, 2009.

[9] B. Wang, J. Takatsubo, Y. Akimune, and H. Tsuda. Development of a remote impact damage identification system. *Structural Control and Health Monitoring*, 12(3-4):301–314, 2005.

[10] P. Wolcken and M. E. Papadopoulos. *Smart Intelligent Aircraft Structures (SARISTU): Proceedings of the Final Project Conference*. Springer International Publishing, 2015.

[11] C. M. Yeum, H. Sohn, H. J. Lim, and J. B. Ihn. Reference-free delamination detection using lamb waves. *Structural Control and Health Monitoring*, pages 1–10, 2013.

[12] L. Yu and V. Giurgiutiu. In situ 2-d piezoelectric wafer active sensors arrays for guided wave damage detection. *Ultrasonics*, 48(2):117–134, 2008.

[13] S. Yuan, P. Liu, and L. Qiu. A miniaturized composite impact monitor and its evaluation research. *Sensors and Actuators A: Physical*, 184:182 – 192, 2012.

[14] S. Zhongqing, Y. Lin, and L. Ye. Guided lamb waves for identification of damage in composite structures: a review. *J. Sound Vib.*, 295:753–780, 2006.