

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA



Corso di Laurea Magistrale in Informatica

**Securing MAVLINK protocol: a Post
Quantum cryptography-based approach**

ANNO ACCADEMICO 2022/2023

RELATORE

Prof. Arcangelo Castiglione

Università degli Studi di Salerno

CANDIDATO

Hermann Senatore

Matricola: **0522501273**

We're flying high

We're watching the world pass us by

Never want to come down

Never want to put my feet back down on the ground

(Depeche Mode)

Indice

1 Introduzione	4
1.1 Scopi e struttura della tesi	4
1.2 UAV, APR o Droni: nomenclatura	5
1.3 Cenni storici	5
1.4 Sviluppi recenti	6
2 Piattaforme software per UAV e GCS	7
2.1 ArduPilot	8
2.1.1 SITL - Software in the Loop	10
2.1.2 ArduPilot: architettura software	11
2.1.3 Compilare ArduPilot	11
2.1.4 ArduPilot: integrazione con il protocollo MAVLink	13
2.2 QGroundControl	15
2.2.1 Panoramica sulle principali GCS disponibili	15
2.2.2 Architettura software di QGroundControl	16
2.2.3 Ottenere QGroundControl	17
3 Il protocollo MAVLink	20
3.1 Principi di funzionamento	20
3.2 Architettura di un pacchetto MAVLink	23
3.2.1 MAVLink 1.0: struttura di un pacchetto	23
3.2.2 MAVLink 2.0: struttura di un pacchetto	25
3.2.3 La firma digitale in MAVLink 2.0	27
3.2.4 Algoritmo di generazione della firma digitale in MAVLink 2.0	29
3.2.5 Scelta della versione del protocollo	30
4 Sicurezza del protocollo MAVLink	32
4.1 Possibili attacchi contro MAVLink	32
4.2 Possibili contromisure	33
4.2.1 MAVSec	34

INDICE

4.2.2	Securing Unmanned Aerial Vehicles by Encrypting MAVLink Protocol	35
4.2.3	Sicurezza in ambienti resource-constrained: una breve introduzione al concetto di lightweight cryptography	36
4.2.4	MAVLink e lightweight cryptography	38
5	Quantum Comuting e Post-Quantum Cryptography	40
5.1	Cos'è il Quantum Computing	40
5.2	Il problema: RSA, DH e l'algoritmo di Shor	42
5.3	La soluzione: Post-Quantum Cryptography	44
5.4	L'algoritmo Kyber	46
5.4.1	KEM - Key Encapsulation Mechanism	46
5.4.2	Reticoli	49
5.4.3	Il problema LWE	51

INDICE

Abstract

L’evoluzione tecnologica a cui si sta assistendo negli ultimi anni sta rivoluzionando pesantemente (tra le altre cose) il mondo dell’aviazione, merito anche (e soprattutto) dei cosiddetti **UAV** (unmanned aerial vehicle), che comunemente vengono definiti **droni**, impiegati sia in contesto "civile" che in contesto militare.

La potenziale delicatezza delle missioni che questi veicoli si trovano ad affrontare suggerisce dunque la necessità di definire dei requisiti di sicurezza che ne permettano un impiego più agevole. L’innovazione tecnologica porta però a nuove sfide anche nel campo della sicurezza.

In particolare, i recenti progressi nel campo del **quantum computing** aprono nuove sfide nel contesto della crittografia, rendendo quindi necessario sviluppare nuove tecniche resistenti ad attacchi veicolati mediante computer quantistici. In questo lavoro viene presentato un **proof of concept** di un’architettura basata principalmente sul protocollo MAVLink che permetta una comunicazione sicura tra un drone e la sua **Ground Control Station** e, ad un livello più alto, la definizione della chiave di cifratura utilizzata mediante il **Key Encapsulation Mechanism** Kyber, selezionato dal **NIST** come lo standard per quanto riguarda gli algoritmi di incapsulamento **quantum resistant**.

1 Introduzione

1.1 Scopi e struttura della tesi

Il presente lavoro si concentra, come tra l’altro già accennato in precedenza nell’abstract, sull’analisi del proof of concept di un’architettura basata sul protocollo MAVLink che permetta lo scambio di chiavi e la comunicazione cifrata tra un UAV e la sua centrale di comando a terra utilizzando il KEM quantum-resistant Kyber e l’algoritmo AES in modo tale da proteggere lo scambio di messaggi tra i due *endpoint* della comunicazione.

In particolare:

- Questo capitolo funge da introduzione al lavoro svolto durante l’attività di Tesi, ne illustra la struttura e ne chiarisce le motivazioni ed il contesto in cui è calato;
- il **Capitolo 2** fornisce una panoramica sulle principali piattaforme software considerate nel presente lavoro;
- il **Capitolo 3** fornisce una panoramica sul protocollo **MAVLink**;
- il **Capitolo 4** discute alcune problematiche di sicurezza che affliggono il protocollo MAVLink e discute alcuni approcci a questo problema presenti in letteratura;
- il **Capitolo 5** fornisce un’introduzione all’impatto dell’avvento dei **Computer Quantistici** nel campo della crittografia e al concetto di **Post-Quantum Cryptography** (PQC). Si procederà quindi ad una panoramica dell’algoritmo **Kyber**;
- il **Capitolo 6** illustra in dettaglio le **modifiche** effettuate alle piattaforme software considerate nel presente lavoro ed al protocollo MAVLink stesso;
- il **Capitolo 7** fornisce le **conclusioni** al presente lavoro e illustra alcuni possibili **sviluppi futuri**.

1.2 UAV, APR o Droni: nomenclatura

Nell'ultimo decennio, l'importante evoluzione tecnologica nel contesto dell'aviazione ha permesso la progettazione e la concreta realizzazione di veicoli in grado di volare e compiere missioni anche **senza la presenza di un pilota umano** sempre più versatili, efficaci e precisi. Questa tipologia di veicoli viene definita, a seconda del contesto linguistico in cui ci si trova, **UAV** (*unmanned aerial vehicle*), **APR** (*aeromobile a pilotaggio remoto*) o, più comunemente, **Drone**. Tutti questi acronimi sono, quindi, equivalenti tra di loro.

La presenza di un essere umano continua tuttavia ad essere fondamentale in quanto il pilotaggio di questi dispositivi viene effettuato mentre una struttura di controllo "a terra", che prende il nome di **Ground Control Station** (da qui in poi *GCS*). Tipicamente, una GCS può essere rappresentata da un qualsiasi apparato in grado di comunicare in qualche modo con l'UAV, quindi anche un comune Personal Computer su cui viene posto in esecuzione un software specifico.

1.3 Cenni storici

Il concetto di aeromobile senza pilota in sé non è sorprendentemente prerogativa degli ultimi anni e delle conseguenze che l'avvento delle tecnologie informatiche si porta dietro. Risale infatti agli Anni '40 del XIX secolo il primo (rudimentale!) impiego di "dispositivi" volanti senza pilota in campo militare.

Per fronteggiare i moti rivoluzionari, peraltro diffusi anche in tutta Europa nel 1848, nella città di Venezia (che avevano portato alla creazione della cosiddetta Repubblica di San Marco), l'esercito austriaco lanciò dei **palloni** a cui era stato fissato dell'**esplosivo** dalla nave "Vulcano".

Questo primo esperimento portò a risultati "misti": alcuni di questi dispositivi riuscirono effettivamente a colpire la città, altri furono invece deviati dal vento.

Per tutto il XIX secolo, lo sviluppo di questo tipo di dispositivi rimase prerogativa militare, con gli americani e gli inglesi che si occuparono dello

sviluppo dei primi aeromobili comandati tramite radiofrequenza. Questi ultimi, nel 1917, testarono con successo un velivolo chiamato **Aerial Target**, mentre gli americani misero a punto il cosiddetto **Kettering Bug** [1].

Nessuno di questi due prototipi venne tuttavia impiegato nella **prima guerra mondiale**.

Come già accennato in precedenza, con il passare del tempo e con il sempre più spinto progresso tecnologico i droni e, più in generale i veicoli a guida remota, hanno trovato un fertile campo di applicazione anche nell'ambito **civile**. Ad esempio, nel 2022 è stato proposto uno studio in cui è stato utilizzato un insieme di veicoli senza pilota per monitorare il processo di realizzazione di un **ponte** [2]. Si è assistito, riassumendo, ad una "democratizzazione" dell'uso dei droni negli ambiti più disparati.

1.4 Sviluppi recenti

Proprio questa diffusione più capillare dell'uso di suddetti veicoli a guida remota ha portato ad una più grande attenzione nello sviluppo di tecnologie software specifiche a questo ambito.

A partire dagli anni 10 del XXI secolo sono stati concepiti diversi progetti software **open source** che forniscono piattaforme integrate compatibili con diversi tipi di apparati sia lato UAV che lato GCS.

Parimenti, è stato necessario definire uno *standard* per quanto riguarda l'ambito dei protocolli di comunicazione tra il drone e la sua *GCS*, che è rappresentato sicuramente dal protocollo **MAVLink** e dalle sue successive evoluzioni.

Una panoramica più approfondita sul protocollo MAVLink verrà in ogni caso affrontata nelle prossime sezioni in cui verranno illustrate le piattaforme software utilizzate nell'ambito del presente lavoro.

2 Piattaforme software per UAV e GCS

Il presente capitolo fornisce una panoramica delle principali tecnologie utilizzate in ambito **civile** ed **amatoriale** nel contesto dei velivoli a pilotaggio remoto. In particolare viene riservata particolare attenzione ai seguenti due progetti **open source**:

- **ArduPilot**: una piattaforma integrata scritta in **C++** utilizzabile su diverse tipologie di veicoli non necessariamente adatti al volo;
- **QGroundControl**: una **Ground Control Station** open source scritta in **C++**, che utilizza il framework **Qt** e che permette una più stretta interazione con la piattaforma ArduPilot.



Il logo di ArduPilot



Il logo di QGroundControl

Viene quindi svolta una panoramica sul protocollo di comunicazione **MAVLink** che permette l'interazione tra queste due componenti.



Il logo del protocollo MAVLink

2.1 ArduPilot

ArduPilot, come accennato, è una suite software universale che permette il controllo di diversi veicoli **non necessariamente volanti** [3] la cui nascita risale al 2007. L'idea di una suite software quale ArduPilot è stata formalizzata presso la piattaforma **DIYDrones.com**, community dedicata agli UAV fondata da **Chris Anderson** e che si definisce quindi "The Birthplace of ArduPilot"[4].

Nel 2009 venne prodotta la prima **board** che utilizzava questa piattaforma e nel novembre dello stesso anno il codice sorgente del progetto venne reso pubblico (ed è attualmente disponibile su GitHub all'url <https://github.com/ArduPilot/ardupilot>)[5].

Correntemente, il progetto permette l'utilizzo su diverse tipologie di dispositivi, mediante delle versioni del firmware leggermente diverse tra di loro. In particolare, ArduPilot supporta dispositivi del tipo:

- **Copter**: probabilmente quello più diffuso tra tutti e che tipicamente è composto da una board a cui sono associate diverse **eliche**. Il progetto di riferimento si chiama **ArduCopter**[6];
- **Plane**: un tipo di dispositivo ad **ala fissa**, tipicamente un **aereo radiocomandato**. Il progetto di riferimento si chiama **ArduPlane**[7];
- **Rover**: un tipo di dispositivo che non può spiccare il volo ma dotato di ruote e che di conseguenza viene utilizzato per missioni "a terra". Il progetto di riferimento si chiama semplicemente **Rover**[8];
- **Sub**: un tipo di dispositivo subacqueo, appartenente alla categoria degli **Autonomous Underwater Vehicles (AUV)**. Il progetto di riferimento si chiama **ArduSub**[9];



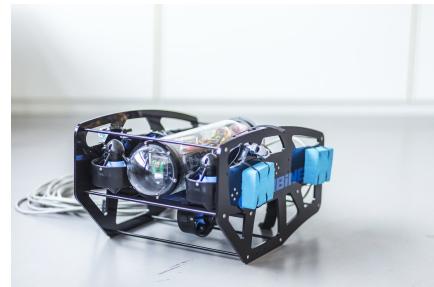
(a) Copter



(b) Plane



(c) Rover



(d) Sub

Il progetto fornisce anche del codice specifico per apparati "complementari" ai dispositivi qui sopra descritti. In particolare:

- **AntennaTracker**: che permette di governare apparati che si occupano di ricevere il segnale proveniente dai dispositivi controllati da remoto[10];
- **Blimp**: un particolare tipo di "pallone" aerostatico dotato di motore[11].



AntennaTracker



Blimp

2.1.1 SITL - Software in the Loop

Il progetto ArduPilot mette inoltre a disposizione anche un **emulatore** che permette il testing di tutte queste varianti della piattaforma **senza possedere l'hardware adatto** [12] e quindi procedere al **flashing** del firmware.

Tale emulatore prende il nome di **SITL** (*Software In The Loop*) e risulta quindi molto utile quando è necessario svolgere attività di testing preliminari per valutare il comportamento del firmware in use case differenti.

Tra l'altro, proprio l'utilizzo di SITL ha permesso la realizzazione di questo lavoro in totale autonomia da hardware specifico.

Maggiori informazioni su SITL e sul suo utilizzo saranno proposte in una successiva sottosezione dove verranno presentati ulteriori dettagli sull'architettura software del progetto ArduPilot.

2.1.2 ArduPilot: architettura software

Come menzionato in precedenza, il progetto ArduPilot utilizza il linguaggio di programmazione **C++**, che è uno di quelli più utilizzati per la programmazione di sistemi embedded in generale poiché **gira direttamente sull'hardware** senza bisogno di macchine virtuali di sorta (al contrario di Java o di linguaggi interpretati), permettendo di ottenere prestazioni migliori in un contesto in cui proprio le prestazioni rappresentano un requisito fondamentale.

Dal punto di vista della configurazione del progetto, la scelta degli sviluppatori di ArduPilot è ricaduta sul framework **waf**. Tale framework, scritto in Python, offre un ambiente di build modulare e per quanto possibile agnostico rispetto ai linguaggi di programmazione utilizzati nel progetto col quale lo si vuole usare.

In breve, la documentazione di waf [13] suggerisce che:

- Per utilizzare il framework è necessaria unicamente un'installazione di **Python**;
- Waf non definisce un nuovo "linguaggio" (come avviene nel meccanismo dei **Makefile**) ma si compone di moduli scritti in Python, che permette quindi una maggiore riusabilità delle componenti;
- I "target" sono definiti come oggetti python dichiarati in maniera separata dai comandi (definiti invece come funzioni in un file chiamato **wscript**).

2.1.3 Compilare ArduPilot

Il progetto GitHub della piattaforma ArduPilot contiene il codice sorgente specifico di tutte le tipologie di veicolo, organizzato in directory separate[14]. In particolare:

- ArduCopter/ contiene il codice sorgente specifico per i veicoli di tipo **Copter**;
- ArduPlane/ contiene il codice sorgente specifico per i veicoli di tipo **Plane**;
- ArduSub/ contiene il codice sorgente specifico per i veicoli di tipo **Sub**;
- Rover/ contiene il codice sorgente specifico per i veicoli di tipo **Rover**;
- AntennaTracker/ contiene il codice sorgente specifico per i dispositivi che comandano **antenne di invio/ricezione**;
- Blimp/ contiene il codice sorgente specifico per i veicoli di tipo **Blimp**, come descritti in precedenza;

Inoltre, mediante il comando `configure` ed il parametro `--board=` è possibile personalizzare ulteriormente la compilazione del firmware adattandolo ad un *tipo di board* specifico o istruendo il sistema a prepararsi ad essere utilizzato mediante **SITL**[15]. Quest'ultimo scenario è quello utilizzato nel presente lavoro e si ottiene eseguendo il comando:

```
./waf configure --board=sitl
```

e, per generare il firmware specifico per un determinato tipo di veicolo (in questo caso quello di elezione è **Copter**), è necessario eseguire il seguente comando:

```
./waf copter
```

Per avviare l'emulatore **SITL** configurato in precedenza è necessario recarsi nella directory del veicolo per cui si è scelto di compilare il firmware (specificato nel comando precedente) (in questo caso, **ArduCopter/**)

```
cd ArduCopter/
```

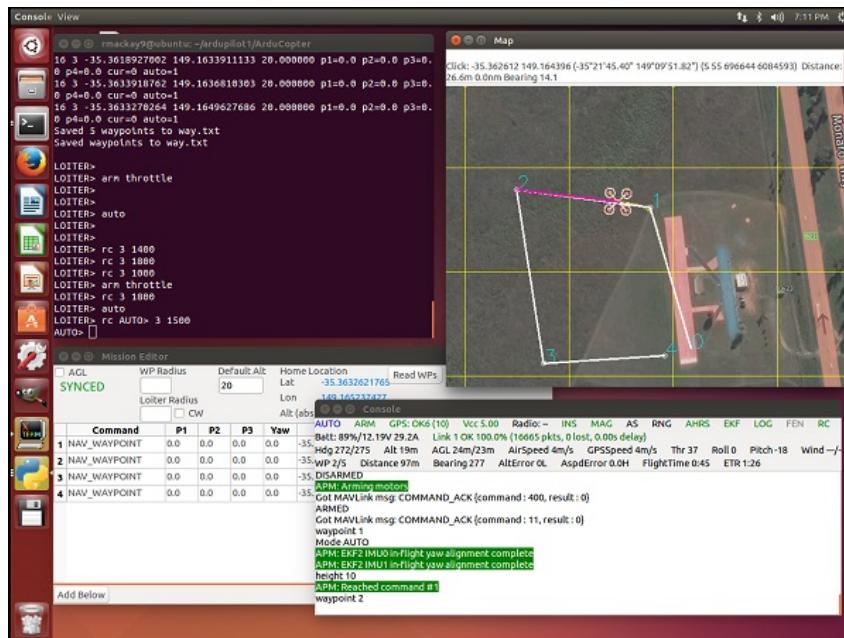
e finalmente eseguire l'emulatore mediante il comando

sim_vehicle.py

Nota: il file appena descritto è presente nel percorso `/Tools/autotest` ma non è necessario specificare il suo path assoluto perché durante l'installazione dei prerequisiti di ArduPilot, quest'ultimo viene aggiunto in maniera system-wide alla variabile d'ambiente PATH.

A questo punto, il veicolo "virtuale" è pronto ad accettare connessioni da una GCS. In particolare, viene aperta la porta **5760** sull'interfaccia di **loopback** locale.

È anche possibile modificare il numero di porta utilizzato e l'interfaccia su cui mettersi in ascolto.



SITL in esecuzione

2.1.4 ArduPilot: integrazione con il protocollo MAVLink

Il progetto ArduPilot usa il protocollo MAVLink per comunicare con le Ground Control Stations. Il codice sorgente che implementa le primitive e le funzioni di supporto per la comunicazione non è però già disponibile all'atto della clonazione del progetto da GitHub. Al contrario, è stato configurato il tool `mavgen` come *submodule* all'interno della repository. Tale tool, richiamato

dal sistema di building `waf` all'atto della compilazione del codice per un determinato veicolo, permette di generare *dinamicamente* tutto il codice di basso livello necessario per le operazioni di comunicazione verso l'esterno[16]. Più nello specifico, viene generata una libreria *headers-only* in linguaggio **C**, che tuttavia risulta perfettamente interoperabile con il resto del software scritto invece, come già menzionato, in C++. Maggiori dettagli su questo tool saranno forniti durante la panoramica che verrà svolta riguardo il protocollo MAVLink.

2.2 QGroundControl

2.2.1 Panoramica sulle principali GCS disponibili

Come menzionato in precedenza, un UAV per essere totalmente operativo ha bisogno di una componente "**a terra**" che lo governi: questa componente prende appunto il nome di **Ground Control Station** (GCS) e si occupa di gestire la connessione con l'UAV mediante il protocollo più adatto e di controlarlo mediante messaggi tipicamente aderenti alle specifiche **MAVLink** (maggiori informazioni sono fornite nella sottosezione dedicata). È quindi pacifico affermare che il software in questione risulti essere nella maggior parte dei casi **molto complesso** dal punto di vista dei requisiti a cui deve rispondere e da quello dell'architettura software.

Attualmente, sono disponibili pubblicamente diverse Ground Control Station più o meno sofisticate. Tra queste si annoverano:

- **MAVProxy** [17]: una (non necessariamente) command-line-based GCS che possiede tutti gli strumenti necessari a comandare un UAV risultando al contempo utilizzabile su sistemi con a disposizione relativamente poche risorse. Di default, è quella che si avvia al momento dell'esecuzione di **SITL** a meno che non venga passato il parametro `--no-mavproxy` allo script `sim_vehicle.py` (come accennato in precedenza);
- **Mission Planner** [18], sviluppato da **Michael Orbone** e che permette di gestire diversi aspetti di una missione di un UAV: dalla configurazione dei waypoints (in termini di coordinate GPS) ai comandi da far eseguire all'UAV durante lo svolgimento della missione;
- **APM Planner 2**: sostanzialmente **un'evoluzione** dello strumento Mission Planner, creato dal team di sviluppo di **ArduPilot** e che permette di gestire sia dispositivi basati su ArduPilot stesso che basati sulla piattaforma **PX4**;
- **QGroundControl** [19]: probabilmente la GCS più funzionale tra quelle appena descritte. È stata sviluppata dal team che si occupa del proto-

collo **MAVLink** e che permette di interfacciarsi con qualsiasi dispositivo che supporti i messaggi MAVLink. Tale piattaforma è stata utilizzata per realizzare parte del lavoro di tesi qui presentato.

2.2.2 Architettura software di QGroundControl

Come accennato, QGroundControl è un software molto complesso e che fornisce una moltitudine di funzionalità, che vanno dalla pianificazione delle missioni al monitoraggio dei messaggi MAVLink tramite un **inspector ad-hoc**, per citarne alcune.

Il progetto è stato sviluppato dall'organizzazione che si occupa di MAVLink e, più in generale, del **MAV-SDK**: la **DroneCode Foundation**, che ha come missione quella di "creare uno standard nell'industria dei droni mediante progetti open-source"[20].

È scritto interamente in **C++**, eccetto per lo stack di comunicazione MAVLink in uso, mediante l'ausilio del framework **Qt** ed il suo codice sorgente è disponibile su GitHub [21].

Al contrario delle altre GCS, QGroundControl **supporta anche la compilazione per la piattaforma Android**[22], il che di conseguenza permette l'utilizzo del software su dispositivi portatili e rendendo il progetto completamente **cross-platform**.

In ogni caso, il *core* della piattaforma è presente all'interno della directory **src/** nella root della repo GitHub. Qui sono presenti tutti i moduli che permettono, tra le altre cose, la gestione dei veicoli connessi alla GCS (classe **Vehicle** coadiuvata da un'istanza di **LinkManager**) e dei messaggi di cui è stato svolto il *parsing* dalla libreria MAVLink (classe **MAVLinkProtocol**: presente nella sottodirectory **comm/**).

La comunicazione con l'UAV connesso avviene mediante **datagrammi UDP** gestiti da un **LinkManager** che contengono messaggi MAVLink costruiti utilizzando le funzioni della libreria C di questo progetto.

Al contrario di quanto avviene nel contesto di ArduPilot, con QGroundControl suddetta libreria **non viene generata** al momento della compilazione

ma viene fatto uso dell'**implementazione di riferimento in C** [23] messa a disposizione pubblicamente da DroneCode su GitHub ed inclusa nella repo di QGroundControl utilizzando (ancora una volta) il meccanismo dei *submodules*.

Il "dialetto" utilizzato di default da QGroundControl per la libreria MAVLink è (a partire dal 24 agosto 2023)[24] `all`, ma tale aspetto può essere variato in uno dei **file di configurazione del progetto**: `QGCExternal-Libs.pri`.

Per quanto riguarda questo lavoro, invece, viene ancora utilizzato il "dialetto" `ArduPilotMega` poiché tale cambiamento non era stato proposto ed approvato.¹

2.2.3 Ottenere QGroundControl

La natura open source e cross platform di tale progetto permette a chi è interessato di ottenerne una copia sostanzialmente in due modi:

1. Utilizzando una **release** periodica precompilata;
2. Compilando (ed eventualmente modificando!) il codice sorgente, approccio utilizzato tra l'altro nel presente lavoro.

Per quanto riguarda la (1.), pacchetti precompilati sono disponibili nella sezione "release" di GitHub, dove sono presenti di solito:

- Un file `.exe` che permette l'installazione su Microsoft Windows;
- Un'immagine disco `.dmg` montabile ed installabile su macOS;
- Un'*AppImage* compatibile la maggior parte delle distro GNU/Linux;
- Un file apk installabile su Android.

¹Cambiamenti così importanti sono particolarmente frequenti in progetti open source del genere, ndr

Per quanto riguarda la (2.), sono naturalmente necessari particolari accorgimenti, oltre ad una piattaforma che permetta di compilare agevolmente un progetto di grandi dimensioni.²

In particolare:

1. È necessario installare il framework **Qt**, mediante l'online installer oppure tramite tool di terze parti [25]. La versione di Qt necessaria attualmente è, secondo quanto indicato [26], la 5.15.2;
2. Tramite il gestore pacchetti `apt` è necessario installare i seguenti pacchetti:
 - `speech-dispatcher`;
 - `libudev-dev`;
 - `libsdl2-dev`;
 - `patchelf`;
 - `build-essential`;
 - `curl` (che probabilmente è già installato di default).

Successivamente è necessario preparare l'ambiente di build mediante i seguenti comandi nella *root* della repository:

```
mkdir build/ && cd build/  
qmake ../
```

L'invocazione di `qmake` permette a Qt di inizializzare i **Makefile** ed di caricare le **variabili d'ambiente** contenute nei **file di configurazione** (come ad esempio il dialetto di MAVLink da utilizzare, già accennato in precedenza).

È ora finalmente possibile invocare `make` per iniziare la procedura di compilazione vera e propria:

```
make -j$(nproc --all)
```

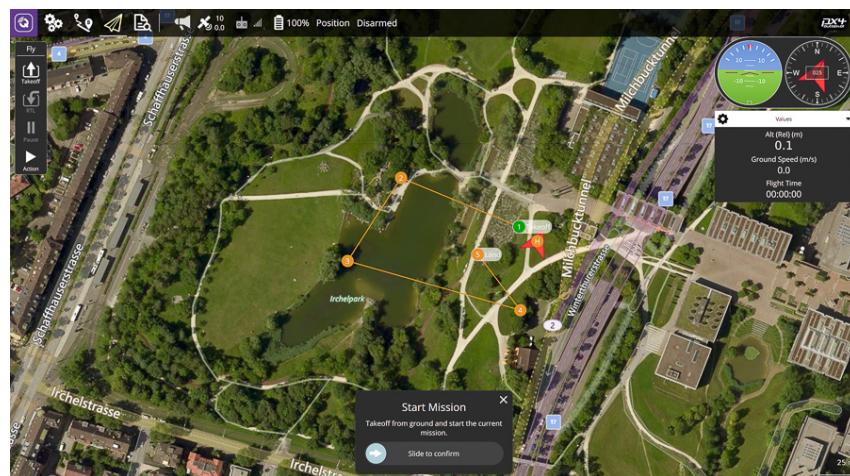
²Gli step riportati qui di seguito si riferiscono al sistema operativo **Ubuntu 20.04 LTS**, lo stesso utilizzato per il lavoro presentato.

2 PIATTAFORME SOFTWARE PER UAV E GCS

Il comando `nproc --all` permette di determinare il numero di core logici che la CPU della macchina su cui si sta svolgendo la compilazione ha a disposizione. Ciò permette una compilazione veloce quanto più possibile.

Una volta terminata la compilazione del software, l'eseguibile prodotto sarà posizionato all'interno della cartella `build/staging` e si chiama semplicemente `QGroundControl`. Sudetto file ha già assegnati i permessi di esecuzione ed è possibile eseguirlo mediante il seguente comando:

staging/QGroundControl



L'interfaccia di QGroundControl una volta aperto

Component #:	Value	Description
BAT_A_PER_V	15.39103031	Battery current per volt (A/V)
*Default Group		
BAT_CAPACITY	-1 mA	Battery capacity
Battery Calibration	0.00080566	Scaling from ADC counts to volt on the ADC input (battery current)
Camera trigger	0.00080566	Scaling from ADC counts to volt on the ADC input (battery voltage)
Circuit Breaker	7 %	Critical threshold
Commander	5 %	Emergency threshold
Data Link Loss	15 %	Low threshold
EKF2	3S Battery	Number of cells
FW Attitude Control	-1.000 Ohms	Explicitly defines the per cell internal resistance
FW L1 Control	Power Module	Battery monitoring source
FW Launch detection	4.05 V	Full cell voltage (SC load)
FW TECS	10.17793941	Battery voltage divider (V divider)
Follow target	BAT_V_EMPTY	Empty cell voltage (SC load)
GPS Failure Navigation	3.40 V	Voltage drop per cell on full throttle
	BAT_V_LOAD_DROP	Offset in volt as seen by the ADC input of the current sensor
	BAT_V_OFFS_CURR	
Geofence		
Land Detector		

Le impostazioni di QGroundControl

3 Il protocollo MAVLink

Come accennato in fase di introduzione a questo lavoro, il **protocollo MAVLink** è il vero e proprio "collante" tra le due parti della comunicazione, ovvero l'UAV e la GCS a terra. In questa sezione viene presentata la filosofia, le specifiche e l'architettura del protocollo MAVLink e se ne illustra l'evoluzione con il passare del tempo.

3.1 Principi di funzionamento

Il protocollo MAVLink (nome breve per **Micro Aerial Vehicle Link**) è stato proposto per la prima volta nel 2010 dallo sviluppatore **Lorenz Meier** [27] e si propone come un protocollo basato su **scambio di messaggi** che ha come caratteristica principale quello di essere **lightweight**: alla luce del contesto in cui si sta operando, quest'ultima caratteristica risulta essere particolarmente desiderabile.

Il principio di funzionamento del protocollo MAVLink è tipico di altri protocolli di tipo **publish-subscribe**, in quanto le due parti della comunicazione (l'UAV e la GCS) pubblicano **messaggi MAVLink** su un **topic** a cui sono entrambi iscritti.



Logo del progetto

La particolarità di questo protocollo risiede nella sua **duttilità** poiché (al contrario di quanto potrebbe suggerire il suo nome) può essere utilizzato per dispositivi diversi da quelli comunemente definiti come "droni". D'altronde, il progetto ArduPilot lo utilizza anche nei progetti che riguardano Rover e Sub.

Come menzionato in precedenza, l'implementazione di riferimento di questo protocollo è disponibile su **GitHub** e si configura come una libreria **header-only** scritta nel linguaggio di programmazione C e che può essere liberamente integrata nei progetti che intendono farne uso, secondo la licenza **LGPL**. [23]

3 IL PROTOCOLLO MAVLINK

Altra particolarità di questo protocollo è la sua **estensibilità**: è presente un **pool** di messaggi standard che permettono una corretta e completa a 360° comunicazione tra UAV e GCS ma è possibile aggiungerne liberamente degli altri per introdurre un comportamento personalizzato, ottenere informazioni non standard e **scambiare dati** di conseguenza.

Proprio questo ultimo aspetto è stato determinante per il lavoro presentato.

Data la sua estensibilità sono stati stabiliti, col passare del tempo, diverse varianti (o **dialetti**) di MAVLink, alcuni di questi "supportati ufficialmente".

Ad esempio:

- **minimal**: il "minimo indispensabile", ovvero quei messaggi che vanno implementati su ogni veicolo pena il non funzionamento del protocollo;
- **common**: il sottoinsieme dei messaggi più **comuni** che *andrebbero* messi a disposizione su tutti i veicoli. Naturalmente, include **minimal**;
- **ardupilotmega**: i messaggi MAVLink usati dal progetto **ArduPilot**, include **common**;
- **ASLUAV**: i messaggi MAVLink utilizzati per veicoli ad ala fissa denominati **ASLUAV**;
- **uAvionix**: i messaggi MAVLink utilizzati dal progetto **uAvionix**;
- **paparazzi**: i messaggi MAVLink utilizzati dall'autopilot **paparazzi**;
- **all**: una collezione di tutti i messaggi definiti in precedenza.

Le definizioni di questi messaggi sono contenute in dei file XML presenti nella repo GitHub del progetto. In ciascuno di questi è presente la direttiva `<include></include>` che permette la definizione di **gerarchie di messaggi**.
[28]

La caratteristica che rende lightweight tale protocollo è il modo in cui i messaggi MAVLink viaggiano sul link di comunicazione: essi vengono infatti **serializzati** in forma **binaria** [29], operazione che ne facilita di molto il

3 IL PROTOCOLLO MAVLINK

parsing, con un *gain* prestazionale non indifferente rispetto al parsing da altri formati **text-based** come JSON o XML, usati tipicamente in contesti in cui le risorse a disposizione non sono limitate, al contrario di quanto avviene nel caso degli UAV.

Con il passare del tempo sono state proposte **due revisioni maggiori** del protocollo MAVLink:

- La versione **1.0**, standardizzata nel 2013 [30];
- La versione **2.0**, le cui migliorie sono descritte nella sezione successiva, rilasciata nella prima parte del 2017 [31].

Fino al 2013 era inoltre ampiamente utilizzata la versione **0.9** del protocollo, deprecata con l'adozione da parte dei vari progetti della versione 1.0.

3.2 Architettura di un pacchetto MAVLink

Un pacchetto MAVLink, al netto delle differenze tra le due revisioni, è composto sostanzialmente da due parti:

- Un **header** che contiene delle informazioni di controllo;
- Un **payload** che contiene i dati veri e propri scambiati nel pacchetto.

3.2.1 MAVLink 1.0: struttura di un pacchetto

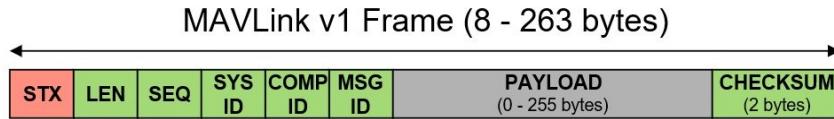
Un **frame** MAVLink 1.0 è strutturato come segue [32]:

- Start of Text (**STX**, chiamato anche **MAGIC**): un byte che rappresenta l'inizio di un frame MAVLink. Nella versione 1.0 del protocollo questo byte assume un valore fisso: 0xFE;
- Payload Length (**LEN**): un byte che rappresenta la **lunghezza in byte** del payload. Poiché questo campo è formato da un solo byte, la lunghezza del payload **non può superare i 255 bytes**;
- Packet Sequence Number (**SEQ**): un byte che rappresenta il numero di sequenza del pacchetto nel contesto di una comunicazione. Serve principalmente per la **detection** di **perdita** dei pacchetti e degli errori in generale;
- System ID (**SYSID**): un byte che rappresenta l'ID del **mittente** del messaggio. Non è possibile indicare 0 (l'indirizzo di broadcast) come mittente poiché viene interpretato come non valido dal parser;
- Component ID (**COMPID**): un byte che rappresenta la **componente** del veicolo che ha generato il messaggio (una videocamera o un qualche accessorio);
- Message ID (**MSGID**): un byte che rappresenta il **Message ID** del contenuto del payload. Serve alla libreria per identificare il messaggio e selezionare il parser più adatto per interpretare il contenuto e deserializzarlo;

3 IL PROTOCOLLO MAVLINK

- Payload (PAYLOAD): un vettore di byte di **al più** 255 bytes (si veda il ragionamento fatto in precedenza) che contiene i dati veri e propri scambiati tra l'UAV e la GCS;
- CRC (+ CRC Extra) (CHECKSUM): due bytes:
 - Il primo rappresenta il checksum del messaggio (nel suo calcolo non viene considerato STX);
 - Il secondo (denominato CRC_EXTRA) rappresenta il CRC della **struttura della definizione XML del messaggio**. Ciò permette alle due parti della comunicazione di assicurarsi di star considerando messaggi definiti allo stesso modo.

Di seguito è presente un riepilogo della struttura di un pacchetto MAVLink nella sua revisione 1.0:



La struttura di un messaggio MAVLink 1.0

Per quanto riguarda la dimensione dei pacchetti MAVLink 1.0:

- La dimensione **minima** di un pacchetto MAVLink 1.0 è di 8 bytes (quando **non** è presente un payload, tipicamente quando si ha a che fare con dei pacchetti di ACK);
- La dimensione **massima** di un pacchetto MAVLink 1.0 è di 263 bytes (8 bytes di header + 255 bytes di payload, quando quest'ultimo è completamente "riempito").

3.2.2 MAVLink 2.0: struttura di un pacchetto

Con il rilascio della versione 2.0 delle specifiche del protocollo, la struttura di un pacchetto è variata leggermente.

In particolare:

- Sono stati aggiunti due nuovi membri dell'header: **CMPFLAGS** e **INCFLAGS**;
- È stato aggiunto il supporto alla **firma digitale del pacchetto**;

Un frame MAVLink 2.0 è quindi strutturato come segue [33]:

- Start of Text (**STX**, chiamato anche **MAGIC**): un byte che rappresenta l'inizio di un frame MAVLink. Nella versione 2.0 del protocollo questo byte assume un valore fisso: 0xFD;
- Payload Length (**LEN**): un byte che rappresenta la **lunghezza in byte** del payload. Poiché questo campo è formato da un solo byte, la lunghezza del payload **non può superare i 255 bytes**;
- Incompatibility Flags (**INCFLAGS**): un byte che rappresenta quelle features del protocollo che **devono essere tassativamente supportate** da chi riceve il pacchetto poiché **ne alterano la struttura**. In caso contrario, le specifiche del protocollo indicano che il pacchetto deve essere **scartato**. L'unico valore attualmente supportato è 0x01, che rappresenta la possibilità di **firmare il messaggio**;
- Compatibility Flags (**CMPFLAGS**): un byte che indica la presenza di features "aggiuntive" che, anche se non supportate dal ricevente **non pregiudicano** la deserializzazione del pacchetto e che quindi lo scarto del pacchetto non è per forza necessario. Un esempio è una flag che denota un pacchetto "ad alta priorità".
- Packet Sequence Number (**SEQ**): un byte che rappresenta il numero di sequenza del pacchetto nel contesto di una comunicazione. Serve prin-

cipalmente per la **detection** di **perdita** dei pacchetti e degli errori in generale;

- System ID (**SYSID**): un byte che rappresenta l'ID del **mittente** del messaggio. Non è possibile indicare 0 (l'indirizzo di broadcast) come mittente poiché viene interpretato come non valido dal parser;
- Component ID (**COMPID**): un byte che rappresenta la **componente** del veicolo che ha generato il messaggio (una videocamera o un qualche accessorio);
- Message ID (**MSGID**): **tre bytes** che rappresentano il **Message ID** del contenuto del payload. Serve alla libreria per identificare il messaggio e selezionare il parser più adatto per interpretare il contenuto e deserializzarlo. Con l'aumento della dimensione di questo campo è possibile supportare un numero più grande di messaggi e di rendere la libreria ancora più flessibile rispetto a prima;
- Payload (**PAYLOAD**): un vettore di byte di **al più** 255 bytes (si veda il ragionamento fatto in precedenza) che contiene i dati veri e propri scambiati tra l'UAV e la GCS;
- CRC (+ CRC Extra) (**CHECKSUM**): due bytes:
 - Il primo rappresenta il checksum del messaggio (nel suo calcolo non viene considerato **STX**);
 - Il secondo (denominato **CRC_EXTRA**) rappresenta il CRC della **struttura della definizione XML del messaggio**. Ciò permette alle due parti della comunicazione di assicurarsi di star considerando messaggi definiti allo stesso modo.

Oltre all'header ed al payload, il protocollo MAVLink 2.0 introduce anche la possibilità di aggiungere (*append*) una **firma digitale** ad un frame. Tale campo è lungo esattamente **13 bytes**.

3 IL PROTOCOLLO MAVLINK

Di seguito è presente un riepilogo della struttura di un pacchetto MAVLink nella sua revisione 2.0:



La struttura di un messaggio MAVLink 2.0

Alla luce di tutte le informazioni qui sopra riportate, riguardo la dimensione di un frame MAVLink 2.0 è possibile menzionare che:

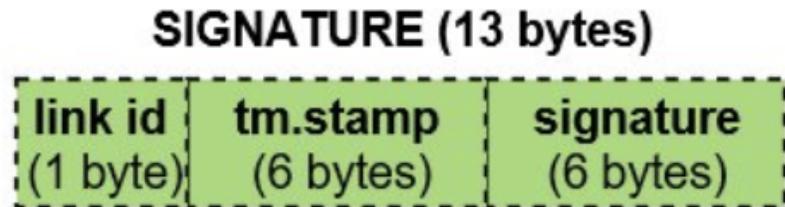
- La dimensione **minima** di un pacchetto MAVLink 2.0 (senza payload e senza firma) è di **12 bytes**;
- La dimensione **massima** di un pacchetto MAVLink 2.0 (con payload "riempito" e con apposta una firma digitale) è di **280 bytes** (**12** bytes di header + **255** bytes di payload + **13** bytes di firma).

3.2.3 La firma digitale in MAVLink 2.0

Come menzionato, a partire dalla versione 2.0 del protocollo MAVLink è stato aggiunto il supporto alla firma digitale dei frame. La struttura dell'header della firma del pacchetto risulta essere grande 13 bytes, così strutturati [34]:

- **Link ID (LINK_ID)**: un byte che rappresenta il **link** su cui è stato inviato un determinato pacchetto;
- **Timestamp (TM_STAMP)**: **sei bytes** che rappresentano il numero di unità di 10 **microsecondi** passati dal **1 Gennaio 2015**. Questo contatore deve aumentare in maniera *monotona* ad ogni messaggio inviato. Se ci si trova in un contesto in cui vengono inviati più di 100.000 messaggi al secondo, questo timestamp può **superare** l'ora corrente;
- **Signature (SIGNATURE)**: **sei bytes** (48 bit) che contengono la firma digitale vera e propria.

Di seguito è presente una rappresentazione grafica dell'header aggiuntivo della firma digitale.



La struttura dell'header di firma

Un pacchetto MAVLink firmato, come accennato in precedenza, è caratterizzato dal campo `INCFLAGS` impostato a `0x01`.

3.2.4 Algoritmo di generazione della firma digitale in MAVLink 2.0

I **sei bytes** che compongono la firma digitale vera e propria sono ottenuti concatenando [35]:

- Una **chiave segreta** condivisa tra le due parti della comunicazione (**SECRET_KEY**);
- I bytes che compongono l'**header** del pacchetto (**HEADER**);
- I bytes che compongono il **payload** del pacchetto (**PAYLOAD**);
- I bytes che compongono il **CRC** dell'intero pacchetto (**CRC**);
- I bytes che compongono il **Link ID** dell'header della firma (**LINKID**);
- I bytes che compongono il **timestamp** dell'header della firma (**TM.STAMP**).

ed estraendo i primi **48 bit** dell'hash di questa sequenza ottenuto utilizzando l'algoritmo **SHA256**.

Ricapitolando:

```
SIGN = SHA256_48(SECRET_KEY + HEADER + PAYLOAD + CRC + LINKID +
                   TM.STAMP) ;
```

dove **SHA256_48** rappresenta il troncamento dell'hash prodotto ai primi **48 bit** e il simbolo **+** rappresenta la concatenazione tra bytes.

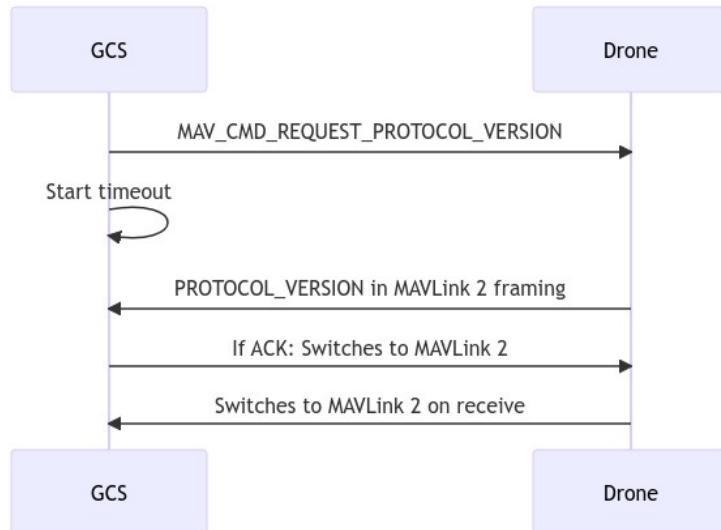
3.2.5 Scelta della versione del protocollo

Poiché l'uso del protocollo MAVLink 1.0 è ancora ampiamente diffuso, è stato necessario implementare un meccanismo di "definizione" della versione del protocollo da utilizzare per una comunicazione: una sorta di **handshaking** [36] preliminare la cui struttura viene riportata di seguito:

- La **Ground Control Station** invia il messaggio `MAV_CMD_REQUEST_PROTOCOL_VERSION` al drone, richiedendo quindi la versione del protocollo in uso dal veicolo ed iniziando contemporaneamente un *countdown di timeout*;
- L'UAV a questo punto può rispondere in due modi:
 - Se è presente il supporto per il protocollo MAVLink 2.0, risponderà con il messaggio di `ACK_PROTOCOL_VERSION` formattato come frame MAVLink 2.0;
 - Se invece non vi è supporto per MAVLink 2.0, risponderà con un **NACK** e, in caso di assenza di procedure apposite, **sarterà il messaggio precedente** e non risponderà affatto.
- – Se la GCS riceve il messaggio `PROTOCOL_VERSION` allora verrà effettuato lo *switch* alla versione 2.0 del protocollo.
 - Se viene ricevuto un NACK oppure si giunge al timeout, viene utilizzata la versione 1.0 del protocollo poiché evidentemente l'UAV non è riuscito ad interpretare in maniera corretta il frame precedente.

3 IL PROTOCOLLO MAVLINK

Di seguito è presente una rappresentazione grafica dell'handshaking appena descritto.



L'handshake effettuato per stabilire la versione del protocollo da utilizzare

4 Sicurezza del protocollo MAVLink

4.1 Possibili attacchi contro MAVLink

Come qualsiasi protocollo che fa utilizzo di un mezzo trasmittivo, MAVLink non è di certo immune a determinati tipi di attacchi che vanno a colpire ognuno dei membri della cosiddetta triade **CIA**, ovvero:

- Confidentiality;
- Integrity;
- Availability

Inoltre, l'intrinseca natura di MAVLink come protocollo **lightweight** implica un maggior focus sulle performance e sulla portabilità del protocollo piuttosto che sulla **sicurezza** dello stesso.

In uno studio [37] del 2019, **Kwon et al.** hanno svolto un'analisi delle possibili vulnerabilità che affliggono il protocollo MAVLink. Tra queste si annoverano:

- **Man-in-the-middle**, che permette ad un attaccante di posizionarsi dal punto di vista logico "al centro" di una comunicazione tra un drone e la sua GCS. Questo è un attacco che viola la confidenzialità e (specialmente) l'integrità della comunicazione;
- **Eavesdropping**, che permette ad un attaccante di "origliare" (*eavesdrop*) la conversazione tra un drone e la sua GCS. Questo attacco viola la confidenzialità della comunicazione;
- **DoS - Denial of Service**, che permette ad un attaccante di operare in maniera tale da rendere uno o anche entrambi gli apparati coinvolti della comunicazione non in grado di continuare. Questo è un attacco all'availability. Ciò può avvenire tramite flooding (attacchi "rumorosi") o mediante lo sfruttamento di determinate vulnerabilità o errori di configurazione degli apparati stessi (attacchi "non rumorosi").

Gli autori dello studio menzionano due tipologie di attacchi portati a termine con successo, testati mediante **SITL** ed ArduPilot **Mission Control**:

- Attacco basato su **ICMP flooding**, che permette di saturare i due endpoint della comunicazione, modificando pesantemente la **varianza** dei tempi di *inter-ricezione* dei pacchetti, rendendo quindi la comunicazione **instabile**;
- Attacco basato su **packet injection**, che permette ad un attaccante di effettuare l'**hijacking** della comunicazione tra drone e GCS ed è basata su una vulnerabilità del **Waypoint Protocol**. In breve, quando un UAV riceve il comando **MISSION_COUNT(N)** da parte della GCS cancella tutte le informazioni relative alla missione che stava eseguendo e resta in attesa di indicazioni sulla nuova missione da eseguire. L'iniezione di un pacchetto di questo tipo permette ad un attaccante di "iniettare" una nuova missione da eseguire magari creata ad-hoc.

Inoltre, il fatto che il protocollo MAVLink non preveda alcun tipo di cifratura dei pacchetti in transito [38] implica una intrinseca insicurezza dello stesso poiché tutte le informazioni sono scambiate in chiaro, rendendo quindi possibile gli attacchi di cui sopra.

4.2 Possibili contromisure

La versione 2.0 di MAVLink, che introduce il sistema della firma digitale dei pacchetti mette **solo parzialmente** una pezza alle problematiche descritte poiché previene il **tampering** delle comunicazioni ma non risolve le problematiche relative alla **confidenzialità**. In ogni caso, il fatto che suddetta firma sia grande solamente **48 bit** apre la porta a tutta un'altra serie di attacchi basati su **bruteforcing**.

Data la sempre maggiore attenzione riservata a questo tipo di dispositivi in particolare nell'ultimo decennio, il problema della messa in sicurezza dei protocolli come il MAVLink ha acquisito sicuramente un'importanza maggiore. Il problema, inoltre, viene reso ancora più difficile dal fatto che si renda

necessario fare un **tradeoff** tra il grado di sicurezza del protocollo e l'alto **throughput** che lo stesso deve offrire pur trovandosi in contesti caratterizzati da risorse hardware tutt'altro che illimitate.

Negli ultimi anni sono stati, in effetti, compiuti alcuni passi in avanti in questo senso grazie a diversi lavori di ricerca.

4.2.1 MAVSec

In uno studio [39] pubblicato nel Maggio del 2019, **Allouch et al.**, oltre ad un'ulteriore overview delle potenziali vulnerabilità insite nel protocollo MAVLink hanno proposto una soluzione che potesse risolvere la problematica della **confidenzialità** della comunicazione introducendo una sorta di *framework* denominato **MAVSec**, che prevede di modificare i progetti **ArduPilot**, **QGroundControl** e la libreria **MAVLink** (nella sua implementazione di riferimento in C) al fine di supportare diversi algoritmi di cifratura **simmetrica** da applicare al **payload** dei messaggi MAVLink prima della loro immissione nel mezzo trasmittivo.

Gli algoritmi considerati nello studio comprendono:

- AES in modalità **CBC**;
- AES in modalità **CTR**;
- RC4 (usato tra l'altro dal protocollo di sicurezza WEP);
- ChaCha20.

Gli autori dello studio hanno confrontato le performance di questi algoritmi sostanzialmente in tre macro-aree:

- **Consumo di memoria** da parte dell'UAV;
- **Numero di frame** inviati al secondo;
- **Consumo di CPU** da parte dell'UAV.

Dai risultati della ricerca emerge che l'algoritmo migliore in **tutti** questi campi risulti essere ChaCha20.

Tale algoritmo è stato proposto nel 2008 in un paper [40] pubblicato da **Daniel J. Bernstein** e si configura come un **cifrario a blocchi** variante di Salsa20/20 che permette di migliorarne la resistenza ad attacchi basati su **crittoanalisi** riducendo in alcuni casi il tempo di esecuzione di ogni round. Il 20 all'interno del nome è dovuto al numero di round che vengono effettuati durante l'esecuzione dell'algoritmo. In generale, i membri della famiglia ChaCha consistono:

- ChaCha8, derivato dall'algoritmo Salsa20/8 e che prevede **8 rounds**;
- ChaCha12, derivato dall'algoritmo Salsa20/12 e che prevede **12 rounds**;
- Il già citato ChaCha20.

4.2.2 Securing Unmanned Aerial Vehicles by Encrypting MAVLink Protocol

In un successivo studio [41] pubblicato alcuni anni più tardi, nel 2022, **Sabuwala et al.** proposero uno studio simile a quello menzionato in precedenza, prendendo però in considerazione algoritmi diversi. In particolare:

- ChaCha20, come nel caso dello studio precedente;
- **Encryption by Navid**, una tecnica di cifratura basata (in parte) su un **Cifrario di Cesare**;
- DMAV, uno schema di cifratura basata su una codifica denominata "DNA dinamica" proposto in uno studio del 2022.

I risultati di questo studio confermano, in un certo senso, quelli del precedente, determinando come l'algoritmo ChaCha20 sia il migliore dei tre confrontati negli ambiti di **pacchetti inviati al secondo, consumo di memoria nell'UAV, percentuale di CPU** in uso nell'UAV.

Si noti come in entrambi i casi sia stato utilizzato il progetto ArduPilot ed in particolare il già citato software **SITL** per condurre i test.

4.2.3 Sicurezza in ambienti resource-constrained: una breve introduzione al concetto di lightweight cryptography

Più o meno nello stesso periodo in cui si il panorama delle piattaforme hardware e software per UAV si è arricchito e l'utilizzo di questi dispositivi è e andata "democratizzandosi", si è assistito, in generale, alla sempre più capillare di dispositivi "intelligenti" da utilizzare nei contesti più disparati. Nasce quindi il concetto di **Internet of Things** (comunemente definito con il suo acronimo **IoT**), che condivide un discreto numero di aspetti sia positivi che negativi con il contesto dell'UAV.

Certamente, si ritrova in entrambi i casi la necessità dell'utilizzo di protocolli leggeri ed efficienti ma anche (e soprattutto) i limiti delle piattaforme hardware che permettono il funzionamento dei dispositivi (quindi tipicamente microcontrollori dotati di funzionalità di rete).

Vien da sé, quindi, che il discorso sulla generale insicurezza dei protocolli affrontato più volte in questo lavoro per gli UAV valga anche per i dispositivi IoT.

Nondimeno, la capillare diffusione di tali dispositivi in ambienti domestici ed aziendali pone un'ulteriore problematica di **privacy** oltre che di **safety** e di **security** come invece avveniva nel caso degli UAV.

Per far fronte a questo tipo di criticità e al (necessario) tradeoff tra performance e requisiti di sicurezza, ecco che col passare del tempo sorge una nuova "branca" di ricerca nell'ambito della crittografia dedicata allo sviluppo di algoritmi (e di relative piattaforme *ad-hoc* per la loro implementazione in hardware) pensati per funzionare in ambienti **resource-constrained**: la **lightweight cryptography**.

Come analizzato in una revisione della letteratura [42] compiuta nel 2018 da **Sadkhan e Salman**, diversi studi sono stati proposti nel campo della crittografia leggera e che hanno portato alla definizione di nuovi schemi di cifratura **simmetrica** ed **asimmetrica**.

Nell'Agosto del 2018, inoltre, il **NIST**³ ha annunciato l'inizio del processo

³National Institute of Standards and Technology, un importante ente statale americano

di standardizzazione [43] di algoritmi di crittografia leggera. Dopo **due rounds** di selezione, nel 2021 sono stati annunciati i 10 **finalisti** [44] di questo processo, ovvero:

- **ASCON**;
- **Elephant**;
- **GIFT-COFB**;
- **Grain128-AEAD**;
- **ISAP**;
- **Photon-Beetle**;
- **Romulus**;
- **Sparkle**;
- **TinyJambu**;
- **Xoodyak**.

Nel 2023, infine, l'algoritmo **ASCON** è stato dichiarato vincitore [45].

4.2.4 MAVLink e lightweight cryptography

In un lavoro di tesi [46] proposto dallo studente **Angelo Passaro**, è stato proposto un approccio basato su crittografia leggera che riprendesse il framework e le tecniche presentate nel lavoro **MAVSec** menzionato in precedenza ed il cui codice è disponibile su GitHub [47].

Tale lavoro si differenzia dai precedenti poiché va a sfruttare diverse caratteristiche del protocollo MAVLink e delle piattaforme software descritte nelle precedenti sezioni del lavoro qui presentato.

Sono stati considerati diversi algoritmi che appartengono al pool della crittografia leggera sia dal punto di vista della cifratura **simmetrica** che dal punto di vista degli algoritmi di **scambio delle chiavi** e sono stati condotti diversi test per stabilire quale fosse quello più conveniente rispetto ai criteri definiti in precedenza dagli altri studi.

In particolare, gli algoritmi di cifratura **simmetrica** integrati nel lavoro qui considerato consistono in:

- Trivium;
- Rabbit;
- ChaCha20 (utilizzato principalmente negli altri studi);
- Simon & Speck;

Per quanto riguarda invece il concetto di scambio delle chiavi, nel lavoro considerato in questa sezione vengono discussi i risultati di uno studio [48] presentato nel 2017 da **Alvarez et al.** che propone un confronto tra i seguenti *meccanismi* di scambio delle chiavi:

- RSA;
- DH: scambio delle chiavi **Diffie-Hellman**;
- ECDH: meccanismo di scambio delle chiavi ispirato a Diffie Hellman ma che si basa sul concetto di **curva ellittica**;

- Curve25519: altro meccanismo di scambio delle chiavi basato su **curve ellittiche** che mira a migliorare le prestazioni rispetto ad ECDH;
- FourQ: ulteriore meccanismo basato su curve ellittiche proposto da **Microsoft** nel 2015.

I risultati raggiunti nel paper menzionato in precedenza suggeriscono che gli algoritmi basati su **curve ellittiche** offrano le migliori prestazioni ed un minore consumo di risorse (memoria e percentuale di CPU utilizzata) ed in particolare, **FourQ** risulta prevalere sugli altri algoritmi del pool considerato.

Proprio per questa ragione, l'algoritmo che sta alla base del meccanismo di scambio delle chiavi implementato nel lavoro di Angelo Passaro consiste proprio in **FourQ**.

Dal punto di vista tecnologico, invece, sono state utilizzate le stesse piattaforme considerate negli altri studi menzionati in precedenza, ovvero:

- **ArduPilot** per l'utilizzo di SITL;
- **QGroundControl** per controllare il simulatore;
- **MAVLink** (nella sua implementazione di riferimento in C) per integrare i nuovi messaggi per lo scambio delle chiavi e gestire cifratura e decifratura.

Il lavoro proposto in questo elaborato ricalca il framework stabilito da **MAVSec** prima e dal lavoro di **Angelo Passaro** poi dal punto di vista delle modifiche introdotte nello stack tecnologico, adattandolo poi al contesto della **Post-Quantum Cryptography**.

Una descrizione tecnica approfondita dell'implementazione effettuata verrà svolta nel **Capitolo 6**.

5 Quantum Computing e Post-Quantum Cryptography

In questo capitolo verrà svolta una breve introduzione al concetto di **Quantum Computing** e si discuteranno le implicazioni della sua diffusione, tra gli altri, nel campo della crittografia.

5.1 Cos'è il Quantum Computing

Sebbene il concetto di Quantum Computing stia acquisendo sempre più importanza negli ultimi decenni, l'idea che sta alla sua base risale ai **primi anni '80** del XX secolo, quando in uno studio [49] del fisico statunitense **Paul Benioff** (venuto a mancare di recente, nel marzo del 2022) venne proposto un primo modello di **macchina di Turing** operante sotto le leggi della **mecanica quantistica**. Da quel momento molti altri studi vennero basati su questa idea: tra questi annoveriamo quello proposto da **Richard Feynmann** a proposito della simulazione di fenomeni fisici usando computer quantistici.

La vera differenza che intercorre tra un computer classico ed una macchina quantistica consiste nel modo in cui sono rappresentate le informazioni: se nel caso "classico" a noi familiare si ragiona in termini di **bit** (ovvero la più piccola unità con cui possono essere codificate delle informazioni), nel caso quantistico si passa al concetto di **qubit**, termine coniato nel 1995 [50] dal fisico Benjamin Schumacher che rappresenta l'unità di informazione quantistica.

Se nel contesto della computazione "classica" gli **stati** possibili che un bit può assumere sono solamente due: lo stato "0" e lo stato "1", nel contesto quantistico il confine tra gli stati che un qubit può assumere è in un certo senso **più labile**, in accordo con il fenomeno quantistico della **sovraposizione tra stati**.

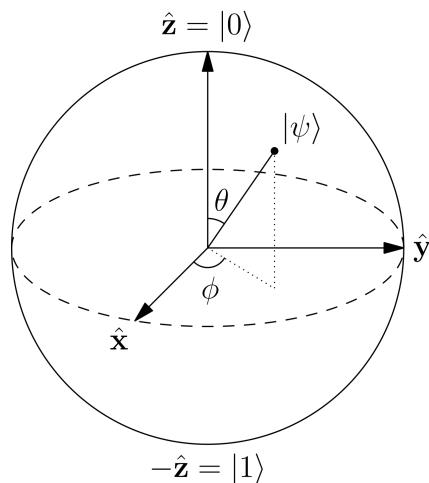
Un qubit può quindi trovarsi rispettivamente:

- Nello stato 0 (con il 100% di probabilità);
- Nello stato 1 (con il 100% di probabilità);

5 QUANTUM COMPUTING E POST-QUANTUM CRYPTOGRAPHY

- In una sovrapposizione di due stati: con il p di probabilità nello stato 0 e con probabilità $1 - p$ nello stato 1.

Al di là dei (comunque importanti e necessari) formalismi matematici, la conseguenza di usare un sistema basato su **qubit** al posto dei classici **bit** permette alle macchine che sfruttano questo sistema di ottenere performance nettamente maggiori a quelle dei computer tradizionali, tant'è che col passare del tempo questa tipologia di macchine sono state usate per provare a risolvere in tempo polinomiale problemi per cui non erano noti algoritmi "classici" efficienti.



La sfera di Bloch: una rappresentazione geometrica dei possibili stati in cui un **qubit** può trovarsi

5.2 Il problema: RSA, DH e l'algoritmo di Shor

Uno degli esempi lampanti del potere computazionale notevolmente maggiore dei computer quantistici rispetto a quelli classici consiste nella pubblicazione di uno studio [51] da parte di **Peter W. Shor** nel 1994 che propone un **algoritmo** (conosciuto appunto come **Algoritmo di Shor**) per la fattorizzazione di interi.

Dal punto di vista teorico, tale algoritmo si configura come una **riduzione** dal problema della fattorizzazione di interi ad un problema di ricerca dell'**ordine** di un gruppo (inteso come struttura algebrica), anch'esso considerato come problema *difficile* nel contesto della computazione classica.

Si noti comunque che il processo di riduzione in sé avviene in tempo polinomiale anche in un computer tradizionale.

Il *core* dell'algoritmo proposto da Shor si occupa quindi di risolvere efficientemente (leggasi, *tempo polinomiale*) il problema verso cui è stata effettuata la riduzione, seppur con un tasso di errore minore o uguale ad $\frac{1}{3}$ nel caso pessimo. La soluzione di tale problema, per com'è strutturato il processo della riduzione di sicurezza implica la soluzione anche del problema "originale".

L'analisi della complessità dell'algoritmo proposto da Shor suggerisce che il suo tempo di esecuzione, comprensivo di tutte le operazioni descritte in precedenza sia proporzionale a:

$$\mathcal{O}((\log N)^2 (\log \log N))$$

assumendo di utilizzare il metodo di **Harvey e Van Der Hoeven** [52] per la moltiplicazione efficiente di interi in tempo $\mathcal{O}(n \log n)$ e che quindi il problema della fattorizzazione di interi appartenga, dal punto di vista della teoria della computazione, alla classe **BQP**, ovvero **la classe dei problemi risolvibili in tempo polinomiale mediante computer quantistici**.

L'approccio proposto da Shor, secondo quanto menzionato nel sommario dello studio, è applicabile non solo al problema della fattorizzazione di interi ma anche al problema del **logaritmo discreto**.

5 QUANTUM COMPUTING E POST-QUANTUM CRYPTOGRAPHY

Le implicazioni dello studio di Shor sono enormi nel campo della crittografia poiché il problema della fattorizzazione di interi e del logaritmo discreto sono alla base degli schemi più utilizzati nel campo della crittografia asimmetrica (o a chiave pubblica): rispettivamente di **RSA** e del meccanismo di Key Exchange **Diffie Hellman**.

La risoluzione in tempo polinomiale mediante computer quantistici di tali problemi renderebbe del tutto insicuri tali schemi, ponendo un considerevole rischio correlato al funzionamento dei protocolli crittografici basati su problemi difficili, come appunto RSA e Diffie Hellman.

Tuttavia, vale la pena sottolineare come il tasso di errore di tale algoritmo (dovuto al concetto di sovrapposizione degli stati) sia ancora troppo alto, pregiudicando quindi l'esecuzione dell'algoritmo anche sulle piattaforme più potenti allo stato attuale.

5.3 La soluzione: Post-Quantum Cryptography

Il problema di situazioni del genere è che una volta note le modalità, ciò che manca è quindi la **potenza di calcolo**, che con l'avanzamento tecnologico sarà con ogni probabilità raggiunta col passare del tempo, ponendo quindi un rischio concreto per la sicurezza dei sistemi informatici (e non solo).

Proprio per regire a questo tipo specifico di minaccia, negli anni ha preso piede un nuovo concetto nel campo della crittografia, che permette di superare la minaccia posta dall'avvento del Quantum Computing: quello di **Post-Quantum Cryptography**.

Il grosso della ricerca in questo campo si è concentrato nel campo della crittografia a chiave pubblica e quindi nel contesto dello scambio di chiavi sicuro poiché è quello che fa affidamento su problemi difficili come quelli descritti in precedenza (si pensi appunto ad RSA ed a Diffie Hellman).

Gli schemi crittografici che sono stati messi appunti nel contesto di questo nuovo campo di studi vengono divisi in diverse categorie, a seconda dell'approccio che adottano [53]. Tra questi:

- Approccio basato su **Reticoli**: l'approccio su cui si basa lo schema Kyber e che fa riferimento ad una struttura algebrica detta **Reticolo** e che include i sistemi crittografici basati sul problema **LWE** (acronimo per **Learning With Errors**) o su loro varianti;
- Approccio basato su **crittografia multivariata**: tale approccio si basa sulla difficoltà della risoluzione di **sistemi di equazioni multivariate**;
- Approccio basato su **hash**, che si basa sull'assunzione della sicurezza delle **funzioni hash**. Esempi di schemi che sfruttano questo approccio consistono nello schema di firma digitale di **Merkle** o comunque altri schemi che usano il **Merkle Tree**;
- Approccio basato su **codici**: ad esempio lo schema di cifratura di **McEliece**, che si basa sui **Codici di Goppa**.

5 QUANTUM COMPUTING E POST-QUANTUM CRYPTOGRAPHY

Proprio l'attenzione riservata col passare del tempo a questo campo di ricerca, anche e soprattutto in seguito allo studio di Shor ed alla definizione del "suo" algoritmo, ha spinto il **NIST** a cercare di stabilire uno **standard** per algoritmi di cifratura a chiave pubblica resistenti ad attacchi con computer quantistici, un po' come successo (e menzionato in una precedente sezione del presente lavoro) con la crittografia leggera.

A dicembre 2016 [54] venne rilasciata una **Request for Nomination** sul sito web del NIST, prima milestone nel processo di standardizzazione, con deadline fissata al 30 novembre 2017.

Dopo quattro round di sottomissioni per tale processo, nel 2022 gli algoritmi selezionati dall'istituto sono i seguenti:

Per la crittografia a chiave pubblica:

- CRYSTALS-Kyber.

Riguardo invece gli schemi di firma digitale:

- CRYSTALS-DILITHIUM;
- FALCON;
- SPHINCS+

Si noti come nel caso della crittografia a chiave pubblica sia stato selezionato **un unico** algoritmo e che quindi è divenuto lo **standard** per questo tipo di schemi.

Anche nel contesto di questo lavoro, lo schema **Kyber** è stato quello scelto per eseguire operazioni di scambio chiavi.

Di seguito viene svolta una panoramica sulle basi e sulla struttura di tale schema crittografico.

5.4 L'algoritmo Kyber

Come affermato in precedenza, CRYSTALS-KYBER è l'unico schema di cifratura selezionato dal NIST nel processo di standardizzazione.

Fa parte della suite **CRYSTALS** (acronimo di **Cryptographic Suite for Algebraic Lattices**) [55] e si configura come "meccanismo di incapsulamento delle chiavi CCA-sicuro basato sulla difficoltà del problema LWE" [56].

Viene proposto alla comunità scientifica nel 2017 in uno studio [57] di **Joppe et al.** e successivamente valutato dal NIST per il processo di standardizzazione.

Prima di descrivere il funzionamento di tale schema è necessario fare una panoramica sulle sue basi matematiche e scientifiche, in particolare il concetto di KEM, il concetto di **Reticolo** ed il problema **LWE** (acronimo di **Learning with Errors**).

5.4.1 KEM - Key Encapsulation Mechanism

L'acronimo KEM è un abbreviazione per **Key Encapsulation Mechanism** ed è un particolare sistema crittografico *ibrido* ed in particolare permette ai due "interlocutori" di una conversazione di utilizzare **un algoritmo di cifratura asimmetrica** per lo scambio di una chiave di cifratura da utilizzare nel contesto di un cifrario **simmetrico**. Naturalmente, il concetto di KEM non è solamente prerogativa della crittografia post quantistica ma è stato studiato ed implementato anche nel contesto della crittografia classica. Un esempio lampante ne è il KEM **basato sullo schema RSA** o su **curve ellittiche**, ampiamente utilizzati al giorno d'oggi.

Formalmente, un generico schema di incapsulamento KEM K è una tupla di algoritmi **probabilistici di tempo polinomiale**:

$$K = (\text{Gen}(), \text{Encaps}(), \text{Decaps}())$$

dove:

5 QUANTUM COMUTING E POST-QUANTUM CRYPTOGRAPHY

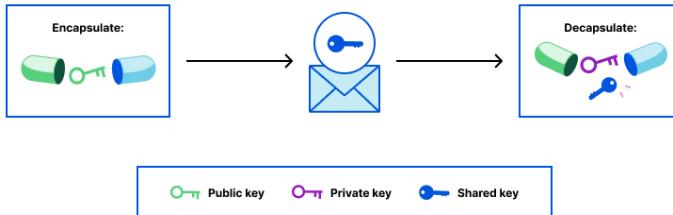
- **Gen(sec_par)** è un algoritmo che si occupa di generare (e restituire) una **coppia di chiavi** (**pk**, **sk**) di una lunghezza determinata da **parametro di sicurezza sec_par** dell'algoritmo. In particolare:
 - **pk** rappresenta la **chiave pubblica**;
 - **sk** rappresenta la **chiave segreta**.
- **Encaps(pk)** è l'algoritmo di **incapsulamento** che prende in input la chiave pubblica generata da **Gen** e restituisce:
 - **k**: una chiave segreta da usare con un cifrario simmetrico. Non deve essere inviata direttamente;
 - **s**: un **testo cifrato**, ovvero la "capsula" che viene inviata all'altro membro della conversazione e che contiene la chiave **k** appena descritta.
- **Decaps(s, sk)** è l'algoritmo di **decapsulamento** che prende in input:
 - **s**: la capsula generata da **Encaps**;
 - **sk**: la chiave segreta generata da **Gen**.E restituisce in output:
 - **k**: la chiave che era stata incapsulata da **Encaps** se non si sono verificati errori.

Questo tipo di costruzione crittografica si pone come **ibrido** tra uno schema di cifratura a chiave pubblica ed a chiave privata.

Si noti come, a differenza dei normali schemi di cifratura, non sia possibile scegliere un messaggio da incapsulare. Al contrario, **Encaps()** si occupa di scegliere la chiave simmetrica **k** e di incapsularla autonomamente.

5 QUANTUM COMUTING E POST-QUANTUM CRYPTOGRAPHY

Di seguito è presente un diagramma che riassume il funzionamento di un generico KEM K , appena descritto formalmente.



Funzionamento di un Key Encapsulation Mechanism⁴

L'architettura di questo tipo di schemi di cifratura rimane sostanzialmente la stessa anche nel caso degli algoritmi post-quantum e l'algoritmo **Kyber** non fa eccezione. Il concetto che cambia è invece l'**assunzione** che sta alla base del funzionamento dello schema: nel caso classico i principali KEM utilizzati si basano sul problema **RSA** (e quindi sulla fattorizzazione di interi) e su **curve ellittiche** come menzionato in precedenza mentre algoritmi come **Kyber** si basano sul problema **LWE**, che può essere ricondotto ad un altro problema definito su un **Reticolo**, entrambi oggetti di discussione nelle prossime sezioni.

⁴Fonente dell'immagine

5.4.2 Reticoli

Nelle sezioni precedenti sono stati presentati i vari approcci alla crittografia postquantistica. Uno di questi consisteva in quello basato su **Reticoli**.

Il nome "reticolo" (in inglese *lattice*) usato senza contestualizzazione risulta essere fuorviante poiché vi sono collegati due concetti distinti, ovvero:

1. **Reticolo** inteso come **struttura algebrica** o, equivalentemente, come **insieme parzialmente ordinato** tale che per ogni coppia di elementi x ed y è possibile individuare l'**estremo superiore** *Sup* e l'**estremo inferiore** *Inf*. Tale tipologia di insiemi può essere rappresentato mediante un **diagramma di Hasse**;
2. **Reticolo** inteso come particolare **sottogruppo** dello spazio vettoriale \mathbb{R}^n .

Il **secondo** concetto è quello utilizzato nel campo della crittografia.

Nel lavoro **Introduction to Post-Quantum Cryptography** [53] è presente una definizione formale di reticolo nel capitolo dedicato a tale concetto, curato da **Daniele Micciancio** ed **Oded Regev**, che viene qui proposta.

Intuitivamente:

Definizione 1. Un reticolo è un insieme di punti in uno spazio n -dimensionale con una struttura periodica.

Formalmente:

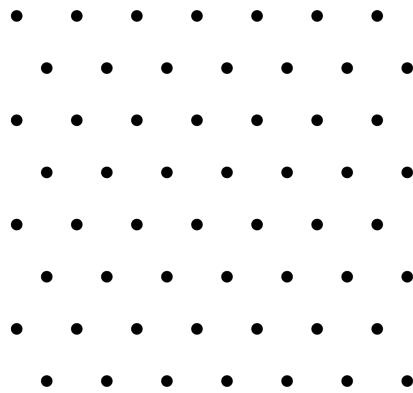
Definizione 2. Siano $b_1, b_2, \dots, b_n \in \mathbb{R}^n$ n vettori linearmente indipendenti (dunque una **base** di \mathbb{R}^n), l'insieme definito come:

$$\mathcal{L}(b_1, b_2, \dots, b_n) = \left\{ \sum_{i=1}^n x_i b_i : x_i \in \mathbb{Z} \right\}$$

prende il nome di **reticolo**. I vettori b_1, b_2, \dots, b_n prendono il nome di **base** del reticolo.

5 QUANTUM COMPUTING E POST-QUANTUM CRYPTOGRAPHY

L'impiego di questo particolare concetto nel campo della crittografia deriva principalmente dalla definizione di alcuni problemi che si suppone generalmente essere **difficili**, come il problema **SVP** (acronimo per **Shortest Vector Problem**, che consiste nel trovare il vettore più piccolo all'interno di un reticolo) e, mediante opportune trasformazioni, del problema **LWE**, che sta alla base dell'algoritmo **Kyber** e che verrà ora descritto.



La rappresentazione di un reticolo su un piano euclideo

5 QUANTUM COMPUTING E POST-QUANTUM CRYPTOGRAPHY

5.4.3 Il problema LWE

Strettamente legato al problema SVP appena menzionato è il problema LWE (acronimo per **Learning with Errors**) poiché mediante un processo di **riduzione** [58] è possibile affermare che se (una variante del) problema SVP (ed in particolare GapSVP) è difficile, allora anche il problema LWE è difficile.

L'idea che sta informalmente alla base di questo problema consiste nel rappresentare determinate informazioni segrete aggiungendo un **errore** ad esse.

Il problema LWE, nonostante si riduca ad un problema definito su reticolli, non è esso stesso un problema definito su reticolli ma un problema **algebrico**.

In particolare, esso sfrutta alcune proprietà dei **sistemi di equazioni lineari**.

Infatti, un generico sistema della forma

$$Ax = b$$

(espresso in forma matriciale per ragione di compattezza) può essere risolto facilmente mediante uno dei metodi "classici" come quello di **sostituzione** o quello di **eliminazione di Gauss-Jordan**.

Tuttavia, se ad un sistema di questa forma viene aggiunto del "rumore", ovvero un certo vettore e della forma:

$$e = \begin{bmatrix} e_1 \\ e_2 \\ \dots \\ e_n \end{bmatrix}$$

facendo quindi diventare il sistema della forma

$$Ax + \mathbf{e} = b$$

la situazione cambia drasticamente poiché **non è più possibile** utilizzare i metodi sopracitati per la risoluzione.

5 QUANTUM COMUTING E POST-QUANTUM CRYPTOGRAPHY

Il problema LWE è definito usando esattamente tale osservazione, anche se di solito per questo tipo di operazioni si tende a restringersi all'insieme degli interi $\text{mod } q$, con $q \in \mathbb{N}$.

Segue ora una definizione formale [59] del problema LWE.

Definizione 3. Sia $q \in \mathbb{N}$ e sia $\mathbb{Z}_q = \{0, 1, \dots, q - 1\}$. Dati m vettori a_1, a_2, \dots, a_m di k componenti ciascuno (formalmente, definiti sull'insieme \mathbb{Z}_q^k), si definisca $s = \{s_1, s_2, \dots, s_k\} \in \mathbb{Z}_q^k$ vettore **segreto** ed $e = \{e_1, e_2, \dots, e_m\} \in \mathbb{Z}_q^m$ vettore degli **errori**. [Work in Progress].

Riferimenti bibliografici e risorse consultate

- [1] Imperial Wars Museum. *A brief history of drones*. <https://www.iwm.org.uk/history/a-brief-history-of-drones>. URL consultato il 5 settembre 2023.
- [2] Antonio Bono et al. “Path Planning and Control of a UAV Fleet in Bridge Management Systems”. In: *Remote Sensing* 14.8 (2022). ISSN: 2072-4292. DOI: 10.3390/rs14081858. URL: <https://www.mdpi.com/2072-4292/14/8/1858>.
- [3] *Sito web di ArduPilot*. <https://ardupilot.org>. URL consultato il 6 settembre 2023.
- [4] *Sito web di DIYDrones*. <https://diydrones.com>. URL consultato il 6 settembre 2023.
- [5] *Documentazione di ArduPilot - sezione "History"*. <https://ardupilot.org/dev/docs/common-history-of-ardupilot.html>. URL consultato il 6 settembre 2023.
- [6] *Documentazione di ArduCopter*. <https://ardupilot.org/copter/index.html>. URL consultato il 6 settembre 2023.
- [7] *Documentazione di ArduPlane*. <https://ardupilot.org/plane/index.html>. URL consultato il 6 settembre 2023.
- [8] *Documentazione di Rover*. <https://ardupilot.org/rover/index.html>. URL consultato il 6 settembre 2023.
- [9] *Documentazione di ArduSub*. <http://www.ardusub.com>. URL consultato il 6 settembre 2023.
- [10] *Documentazione di AntennaTracker*. <https://ardupilot.org/antennatracker/index.html>. URL consultato il 6 settembre 2023.
- [11] *Documentazione di Blimp*. <https://ardupilot.org/blimp/index.html>. URL consultato il 6 settembre 2023.

RIFERIMENTI BIBLIOGRAFICI E RISORSE CONSULTATE

- [12] *Introduzione a SITL.* <https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html>. URL consultato il 7 settembre 2023.
- [13] *Introduzione a WAF.* https://waf.io/book/#_the_waf_framework. URL consultato il 7 settembre 2023.
- [14] *Repository principale di ArduPilot.* <https://github.com/ArduPilot/ardupilot/tree/master>. URL consultato il 7 settembre 2023.
- [15] *Repository principale di ArduPilot - istruzioni per la compilazione.* <https://github.com/ArduPilot/ardupilot/blob/master/BUILD.md>. URL consultato il 7 settembre 2023.
- [16] *Informazioni su mavgen.* https://mavlink.io/en/getting_started/generate_libraries.html. URL consultato l'8 settembre 2023.
- [17] *Documentazione di MAVProxy - pagina iniziale.* <https://ardupilot.org/mavproxy/index.html>. URL consultato l'8 settembre 2023.
- [18] *Documentazione di Mission Planner - pagina iniziale.* <https://ardupilot.org/planner/index.html>. URL consultato l'8 settembre 2023.
- [19] *Sito web di QGroundControl.* <http://qgroundcontrol.com>. URL consultato l'8 settembre 2023.
- [20] *Home page della fondazione DroneCode.* <https://dronecode.org>. URL consultato l'8 settembre 2023.
- [21] *Codice sorgente di QGroundControl.* <https://github.com/mavlink/qgroundcontrol>. URL consultato l'8 settembre 2023.
- [22] *Codice sorgente di QGroundControl - App Android.* <https://github.com/mavlink/qgroundcontrol/tree/master/android>. URL consultato l'8 settembre 2023.
- [23] *Implementazione di riferimento in linguaggio C della libreria MAVLink.* https://github.com/mavlink/c_library_v2. URL consultato l'8 settembre 2023.

RIFERIMENTI BIBLIOGRAFICI E RISORSE CONSULTATE

- [24] *QGCExternalLibs.pri: switch to all mavlink dialect by default.* Link al commit relativo al cambiamento. URL consultato l'8 settembre 2023.
- [25] *aqtinstall.* <https://github.com/miurahr/aqtinstall>. URL consultato l'8 settembre 2023.
- [26] *QGroundContol: Getting Started with source & builds.* https://dev.qgroundcontrol.com/master/en/getting_started/index.html. URL consultato l'8 settembre 2023.
- [27] *Initial commit.* Link al primo commit del progetto MAVLink. URL consultato il 13 settembre 2023.
- [28] *Definizione dei messaggi MAVLink per le varie piattaforme.* https://github.com/mavlink/mavlink/tree/master/message_definitions/v1.0. URL consultato il 13 settembre 2023.
- [29] *Overview - MAVLink Developer Guide.* <https://mavlink.io/en/about/overview.html>. URL consultato il 13 settembre 2023.
- [30] *MAVLink Versions - MAVLink Developer Guide.* https://mavlink.io/en/guide/mavlink_version.html. URL consultato il 13 settembre 2023.
- [31] *MAVLink 2 - MAVLink Developer Guide.* https://mavlink.io/en/guide/mavlink_2.html. URL consultato il 13 settembre 2023.
- [32] *Struttura di un pacchetto MAVLink 1.0.* https://mavlink.io/en/guide/serialization.html#v1_packet_format. URL consultato il 13 settembre 2023.
- [33] *Struttura di un pacchetto MAVLink 2.0.* https://mavlink.io/en/guide/serialization.html#v2_packet_format. URL consultato il 13 settembre 2023.
- [34] *Struttura dell'header della firma in MAVLink 2.0.* https://mavlink.io/en/guide/message_signing.html#frame-format. URL consultato il 13 settembre 2023.

RIFERIMENTI BIBLIOGRAFICI E RISORSE CONSULTATE

- [35] *Algoritmo per la creazione della firma in MAVLink 2.0.* https://mavlink.io/en/guide/message_signing.html#signature. URL consultato il 13 settembre 2023.
- [36] *Handshake per la definizione del protocollo.* https://mavlink.io/en/guide/mavlink_version.html#version_handshaking. URL consultato il 13 settembre 2023.
- [37] Young-Min Kwon et al. “Empirical Analysis of MAVLink Protocol Vulnerability for Attacking Unmanned Aerial Vehicles”. In: *IEEE Access* 6 (2018), pp. 43203–43212. DOI: 10.1109/ACCESS.2018.2863237.
- [38] *MAVLink FAQ - How secure is MAVLink?* <https://mavlink.io/en/about/faq.html>. URL consultato il 14 settembre 2023.
- [39] Azza Allouch et al. “MAVSec: Securing the MAVLink Protocol for ArduPilot/PX4 Unmanned Aerial Systems”. In: *CoRR* abs/1905.00265 (2019). arXiv: 1905.00265. URL: <http://arxiv.org/abs/1905.00265>.
- [40] Daniel Bernstein. “ChaCha, a variant of Salsa20”. In: <https://cr.yp.to/papers.html#chacha> (Jan. 2008).
- [41] Noshin Sabuwala and Rohin D Daruwala. “Securing Unmanned Aerial Vehicles by Encrypting MAVLink Protocol”. In: *2022 IEEE Bombay Section Signature Conference (IBSSC)*. 2022, pp. 1–6. DOI: 10.1109/IBSSC56953.2022.10037546.
- [42] Sattar B. Sadkhan and Akbal O. Salman. “A survey on lightweight-cryptography status and future challenges”. In: *2018 International Conference on Advance of Sustainable Engineering and its Application (ICASEA)*. 2018, pp. 105–108. DOI: 10.1109/ICASEA.2018.8370965.
- [43] *Announcing Request for Nominations for Lightweight Cryptographic Algorithms.* <https://csrc.nist.gov/News/2018/requesting-nominations-for-lightweight-crypto-algs>. URL consultato il 15 settembre 2023.

- [44] *Lightweight Cryptography Standardization: Finalists Announced.* <https://csrc.nist.gov/News/2021/lightweight-crypto-finalists-announced>. URL consultato il 15 settembre 2023.
- [45] *Lightweight Cryptography Standardization Process: NIST Selects Ascon.* <https://csrc.nist.gov/news/2023/lightweight-cryptography-nist-selects-ascon>. URL consultato il 15 settembre 2023.
- [46] *A custom MAVLink with lightweight crypto.* <https://github.com/angelopassaro/SEC-UAV/tree/master>. URL consultato il 15 settembre 2023.
- [47] *Codice sorgente di MAVSec.* <https://github.com/aniskoubaa/mavsec>. URL consultato il 15 settembre 2023.
- [48] Rafael Álvarez, Juan Santonja, and Antonio Zamora. “Algorithms for Lightweight Key Exchange”. In: *Ubiquitous Computing and Ambient Intelligence*. Ed. by Carmelo R. García et al. Cham: Springer International Publishing, 2016, pp. 536–543. ISBN: 978-3-319-48799-1.
- [49] Paul Benioff. “The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines”. In: *Journal of Statistical Physics* 22.5 (May 1980), pp. 563–591. ISSN: 1572-9613. DOI: 10.1007/BF01011339. URL: <https://doi.org/10.1007/BF01011339>.
- [50] Benjamin Schumacher. “Quantum coding”. In: *Phys. Rev. A* 51 (4 Apr. 1995), pp. 2738–2747. DOI: 10.1103/PhysRevA.51.2738. URL: <https://link.aps.org/doi/10.1103/PhysRevA.51.2738>.
- [51] P.W. Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

- [52] David Harvey and Joris van Der Hoeven. “Integer multiplication in time $O(n \log n)$ ”. In: *Annals of Mathematics* (Mar. 2021). DOI: 10.4007/annals.2021.193.2.4. URL: <https://hal.science/hal-02070778>.
 - [53] Daniel J. Bernstein. “Introduction to post-quantum cryptography”. In: *Post-Quantum Cryptography*. Ed. by Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–14. ISBN: 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_1. URL: https://doi.org/10.1007/978-3-540-88702-7_1.
 - [54] *Announcing Request for Nominations for Public-Key Post-Quantum Cryptographic Algorithms*. <https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms>. URL consultato il 19 settembre 2023.
 - [55] *CRYSTALS - Cryptographic Suite for Algebraic Lattices*. <https://pq-crystals.org/index.shtml>. URL consultato il 19 settembre 2023.
 - [56] *Kyber*. <https://pq-crystals.org/kyber/index.shtml>. URL consultato il 19 settembre 2023.
 - [57] Joppe Bos et al. *CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM*. Cryptology ePrint Archive, Paper 2017/634. <https://eprint.iacr.org/2017/634>. 2017. DOI: 10.1109/EuroSP.2018.00032. URL: <https://eprint.iacr.org/2017/634>.
 - [58] Oded Regev. “On Lattices, Learning with Errors, Random Linear Codes, and Cryptography”. In: *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*. STOC ’05. Baltimore, MD, USA: Association for Computing Machinery, 2005, pp. 84–93. ISBN: 1581139608. DOI: 10.1145/1060590.1060603. URL: <https://doi.org/10.1145/1060590.1060603>.
 - [59] *La matematica dietro la PQC: Learning With Errors*. <https://www.telsy.com/it/la-matematica-dietro-la-pqc-learning-with-errors-lwe/>. URL consultato il 21 settembre 2023.
-