UNIVERSITÀ POLITECNICA DELLE MARCHE

Dipartimento di Ingegneria dell'Informazione

TESI DI LAUREA MAGISTRALE

Sviluppo, implementazione e analisi di protocolli quantum-safe per le comunicazioni satellitari

Development and Analysis of Quantum-Safe Protocols for Satellite Communications

---

Candidato:
Davide De Zuane

Relatore:
Prof. Marco Baldi

Co-relatore:
Dr. Paolo Santini

I just wondered how things were put together.

Claude E. Shannon

UNIVERSITÀ POLITECNICA DELLE MARCHE

Sommario

Dipartimento di Ingegneria dell'Informazione

Laurea Magistrale Ingegneria Informatica e dell'Automazione

Sviluppo, implementazione e analisi di protocolli quantum-safe per le comunicazioni satellitari

by Davide DE ZUANE

In questa tesi, si esplora il contesto delle comunicazioni satellitari con l'obiettivo di applicare primitive post-quantum. Inizialmente, è stato condotto un lavoro di benchmarking preliminare per valutare la fattibilità di integrare tali primitive in questo settore e per analizzarne l'impatto sui protocolli di sicurezza attualmente utilizzati. Successivamente, è stata realizzata un'implementazione in grado di soddisfare i requisiti rigorosi di sicurezza di questi protocolli, specificamente per un ambiente a risorse limitate come quello spaziale. L'intento di questo lavoro è di fornire una base per future ricerche e sviluppi in questo campo.

Acknowledgements

La stesura di questa tesi rappresenta il risultato di un percorso lungo e impegnativo, che non sarebbe stato possibile senza il supporto e l'aiuto di diverse persone, alle quali vorrei esprimere la mia più sincera gratitudine.

Indice

Elenco delle figure

Elenco delle tabelle

List of Abbreviations

DH	**D**iffie **H**ellman
SA	**S**ecurity ellman
KE	**K**ey **A**ssociation
PQ	**P**ost **Q**uantum
RFC	**R**equest **F**or **C**omments
PQC	**P**ost **Q**uantum **C**riptography
GNU	**G**NU's **N**ot **U**nix
KEM	**K**ey **E**ncapsulation **M**echanism
SIG	**SIG**nature Mechanism
IKE	**I**nternet **K**ey **E**xchange
PRF	**P**seudo **R**andom **F**unction
MTU	**M**aximum **T**ransmission **U**nit
ISP	**I**nternet **S**ervice **P**rovider
NAT	**N**etwork **A**ddress **T**ranslation

Capitolo 1

Introduzione

Un protocollo di comunicazione consente a due utenti (che, come di consueto, verranno spesso indicati come Alice e Bob) di stabilire un canale sicuro, ovvero, nel quale autenticazione, integrità e confidenzialità siano garantite. In altre parole, grazie alle regole del protocollo, Alice e Bob sono in grado di accordarsi su un segreto condiviso, che andranno ad utilizzare per cifrare le loro comunicazioni. Normalmente, queste funzionalità vengono realizzate tramite primitive crittografiche di tre tipologie:

- *algoritmi di cifratura simmetrica*: utilizzati per garantire confidenzialità, trasformando un testo di senso compiuto (chiamato plaintext) in un testo che sembra incomprensibile (chiamato ciphertext). Queste operazioni vengono effettuate tramite algoritmi di cifratura e decifratura che prendono in input anche la *chiave segreta*, ovvero, una stringa che deve essere nota sia ad Alice, sia a Bob. Senza questa stringa, poter decifrare in modo efficiente non è possibile;

- *algoritmi di scambio di chiave*: algoritmi che permettono ad Alice e Bob di accordarsi su di un segreto condiviso, tramite comunicazioni che avvengono solo tramite canale pubblico;

- *algoritmi di firma digitale*: servono per garantire autenticazione, ad esempio, sono usati per generare certificati di chiave pubblica.

Algoritmi di scambio di chiave e di firma digitale richiedono, oltre alla chiave segreta, anche la cosiddetta *chiave pubblica* e che, tipicamente, è più grande della chiave segreta.
Esistono numerose applicazioni di questi concetti nella vita di tutti i giorni. Quando ci si collega ad un rete Wireless senza alcuna crittografia, un attaccante che si trova nella stessa rete potrebbe facilmente intercettare il traffico di rete. Protocolli come WPA (Wi-Fi Protected Access) sono stati sviluppati per affrontare questa problematica, offrendo misure di sicurezza che proteggono

le comunicazioni e garantiscono la riservatezza dei dati trasmessi sulla rete. Un altro esempio potrebbe essere quello in cui si immagina di configurare una macchina remota denominata Bob, mentre si assume il ruolo di Alice. Senza l'uso della crittografia, un attaccante potrebbe facilmente intercettare il traffico di rete, ottenendo accesso a tutte le informazioni trasmesse, comprese le credenziali di accesso e i comandi eseguiti. Ciò potrebbe portare alla compromissione dell'integrità del sistema. Per questo motivo per collegarsi a macchine remote viene utilizzato Secure Shell (SSH). Come abbiamo visto nei precedenti esempi, i messaggi scambiati tra due parti avvengono tramite canale pubblico, ovvero, un mezzo ne quale è possibile che sia presente un attaccante in grado di leggere e alterare i messaggi. L'utilizzo della crittografia, nei casi considerati (WPA e SSH), consente di contrastare in maniera efficiente l'azione degli attaccanti. In particolare, gli algoritmi a chiave asimmetrica (scambi di chiave e firme digitali) sono fondamentali dato che senza di essi, l'unico modo per instaurare una comunicazione sicura sarebbe quello di distribuire le chiavi manualmente.

Crittografia post-quantum

Come è ben noto, gli attuali schemi di crittografia a chiave pubblica sono minacciati dall'algoritmo di Shor [1], un algoritmo quantistico in grado di risolvere, in tempo polinomiale, problemi come la fattorizzazione ed il calcolo del logaritmo discreto. Questa vulnerabilità implica che gli schemi attualmente in uso sono vulnerabili ad attacchi condotti da computer quantistici, ciò ha spinto la comunità a sviluppare alternative che possano resistere a tali attacchi.
Sono stati esplorati nuovi problemi matematici alternativi su cui sviluppare gli algoritmi; tra i principali su cui si fondano le soluzioni proposte fino ad oggi, si possono citare i problemi legati ai **reticoli**, quelli di **isogenia** e i problemi di **codifica**. Questi si sono dimostrati resistenti ad attacchi quantistici, poichè attualmente non ci sono algoritmi noti in grado di risolvere questi problemi in tempi ragionevoli utilizzando computer quantistici. A partire da tali problemi, sono stati sviluppati algoritmi come *Crystal-Kyber*, *Dilithium* e *Falcon*, che non solo forniscono soluzioni pratiche e sicure per la crittografia moderna, ma rappresentano anche un passo fondamentale verso la creazione di sistemi resilienti nell'era quantistica. Pertanto, la crittografia post-quantum si riferisce a un insieme di protocolli e algoritmi crittografici progettati per essere resistenti agli attacchi condotti da computer quantistici, garantendo così la sicurezza

delle comunicazioni e dei dati anche nell'era in cui la tecnologia quantistica sarà ampiamente diffusa.

Comunicazioni satellitari

Come anche recenti notizie di cronaca mostrano (ad esempio, Starlink è ormai sulla bocca di tutti, anche dei non esperti del settore), c'è un interesse sempre maggiore per lo spazio e per le comunicazioni satellitari. Infatti lo spazio è visto come un terreno inesplorato e, pertanto, pieno di potenzialità. Questa tendenza è confermata anche dall'Agenza Spaziale Europea (ESA). Infatti, negli ultimi anni l'agenzia ha finanziato numerosi studi legati all'utilizzo dei satelliti come tramite per l'esecuzione di protocolli Internet. Esempi di questi studi, ai quali l'Università Politecnica delle Marche ha partecipato, sono:

- CRYPTOSAT: lo scopo del progetto è quello di studiare l'utilizzo di comunicazioni satellitari per implementare protocolli Internet. Il protocollo più adatto per gli scenari considerati è risultato essere IKEv2 [2].
- QUALITA: lo scopo del progetto è quello di studiare come IKEv2 può essere reso sicuro nei confronti di attaccanti equipaggiati con computer quantistici.
- SATELIKE: lo scopo del progetto è progettare e testare una nuova versione di IKEv2, con ridotti costi di comunicazione e computazionali. Possibilmente, il protocollo finale deve possedere sicurezza contro attaccanti con computer quantistici.

Criticità Il contesto delle comunicazioni satellitari, specialmente nel caso di schemi con sicurezza quantum, pone nelle sfide non banali. Infatti, il canale satellitare è dotato di una qualità inferiore rispetto ad altri tipi di canali di comunicazione. Questo fenomeno porta a una maggiore probabilità di corruzione dei pacchetti che transitano sul canale. Tale caratteristica si rivela poco compatibile, in linea teorica, con gli algoritmi di crittografia post-quantum. Del resto, questi ultimi, per garantire una maggiore sicurezza, tendono ad aumentare le dimensioni delle chiavi pubbliche. Tale incremento dimensionale implica un maggior carico di dati da trasmettere, aumentando così la probabilità di errori durante la trasmissione.
La limitatezza dell'hardware presente nei satelliti non facilita la situazione. Gli algoritmi di crittografia post-quantum, per aumentare la loro complessità e sicurezza, si fondano su problemi matematici che richiedono operazioni

computazionalmente onerose. Questo comporta un elevato carico sui processori, i quali potrebbero non essere in grado di gestire tali operazioni in modo efficiente. Le operazioni di codifica e decodifica, anch'esse complesse, possono portare a ritardi significativi nella trasmissione dei dati e a un aumento del consumo energetico, che è un fattore critico nei sistemi satellitari.
Queste problematiche possono ostacolare il corretto funzionamento dei protocolli, poiché, in un contesto come quello delle comunicazioni satellitari, potrebbero essere necessari numerosi tentativi per completare con successo le operazioni di crittografia. Dunque le limitazioni appena descritte sono le sfide da affrontare per poter applicare soluzioni di crittografia post-quantum nel contesto delle comunicazioni satellitari.

Contributo Apportato

In questo lavoro di tesi è stata studiata la fattibilità di applicare IKEv2 nel contesto delle comunicazioni satellitari. Partendo dai risultati di CRYPTOSAT (progetto pressochè terminato nel momento in cui l'attività di tesi è stata svolta), si è andato a studiare il funzionamento di IKEv2 nel contesto satellitare, considerando però versioni *quantum-safe* del protocollo. Il protocollo può essere reso quantum-safe considerando diverse modalità; ognuna di esse porta all'utilizzo di diversi algoritmi (o combinazioni di essi) e di diverse procedure per derivare il materiale crittografico necessario per instaurare un canale sicuro tra Alice e Bob. Tipicamente, ogni modalità porta ad aumenti sia nelle dimensioni dei messaggi, sia nel carico computazionale. Pertanto, una delle prime domande relative al progetto QUALITA è la seguente: è possibile rendere quantum-safe IKEv2 e, al tempo stesso, garantire il suo utilizzo nel contesto delle comunicazioni satellitare? Il lavoro svolto in questa tesi offre una prima risposta, positiva, al quesito. L'esecuzione del protocollo è stata testata all'interno di un ambiente isolato, creato tramite tecniche di containerizzazione, come Docker. Per studiare la fattibilità del protocollo, sono state misurate proprietà chiave come dimensione dei messaggi scambiati e tempi necessari al completamento dagli scambi del protocollo. Questo è stato possibile grazie a strumenti di sniffing, come tcpdump, operando a livello kernel offrono una precisione maggiore sui tempi. L'intero processo è stato reso ripetibile e automatizzato grazie all'implementazione di uno script bash che gestisce sia il parsing dei log a livello kernel che la configurazione dell'ambiente di test. I risultati mostrano che, nonostante gli overhead già previsti, l'esecuzione del protocollo resta molto veloce: i tempi richiesti

per una corretta esecuzione restano nettamente inferiori rispetto ai tempi di latenza dovuti alla propagazione via satellite. Impiegando le primitive post-quantum adeguate sia durante la fase di KEM (Key Exchange Mechanism) che in quella di AUTH (Autenticazione) del protocollo, è possibile ottenere tempi di esecuzione e dimensioni dei pacchetti tali da permettere di proteggere efficacemente le comunicazioni satellitari dagli attacchi quantistici. Ad esempio, l'utilizzo di *Kyber3* per lo scambio `INIT` richiede solamente $0.77ms$ e un tototale di $3KB$. Mentre l'utilizzo di *Falcon512* per lo scambio di `AUTH` richiede $1ms$ per essere completato e uno scambio di dati pari a $5.21KB$. Il tempo per completare l'handshake IKE ($1.77ms$) è estremamente basso rispetto ai tempi di latenza tipici delle comunicazioni satellitari, sia per satelliti GEO ($700ms$), che LEO ($50ms$). Pertanto, l'overhead introdotto dalle primitive post-quantum è praticamente irrilevante rispetto al tempo di latenza della trasmissione, che dunque rimangono il fattore dominante nelle performance globali della comunicazione. Partendo da questi risultati, si è andato poi a studiare la possibilità di rendere IKEv2 *lightweight*[3]. Come già detto, il contesto delle comunicazioni satellitari è caratterizzato da notevoli limitazioni hardware; ovviamente, situazioni come questa capitano in numerosi altri contesti, ad esempio, nel caso di reti IoT. Il lavoro svolto ha portato ad un'implementazione custom di IKEv2, scritta in linguaggio C. Per le funzioni crittografiche, è stata utilizzata la libreria OpenSSL, ampiamente utilizzata e certificata per quel che riguarda protezione da attacchi side-channel. Grazie a questa implementazione, è stato possibile iniziare a semplificare il pattern dei messaggi richiesti in IKEv2. Il lavoro svolto si è finora concentrato sui primi scambi (IKE_INIT ed IKE_AUTH), che sono i più critici visto che utilizzano primitive a chiave pubblica (dunque, hanno dimensioni dei messaggi che sono tipicamente più grandi). Sono state apportate semplificazioni in modo da ridurre overhead legati a funzionalità non necessarie per lo scenario di riferimento. In particolare, si sono esclusi o ridotti i seguenti elementi:

- *Protocolli di autenticazione* non richiesti, come EAP, che pur risultando utili in contesti specifici, introducono complessità e carico computazionale non giustificati.
- *Meccanismi di rekeying* delle SA, utilizzati per mantenere attive le associazioni di sicurezza. Questi richiedono un notevole utilizzo di risorse computazionali.
- *Funzionalità di NAT traversal* (NAT-T): progettato per gestire le situazioni

in cui i pacchetti devono attraversare dispositivi che modificano gli indirizzi IP. Tuttavia, in ambienti controllati o in scenari di comunicazione satellitare, questa funzione non è necessaria.

- *Modalità di negoziazione multiple*: la versione completa di IKE supporta diverse modalità di negoziazione per garantire flessibilità e interoperabilità tra dispositivi, come la modalità Aggressive e la modalità Main. La versione minimale si limita alla modalità essenziale.

- *Cipher suites ridotte*: per semplificare la negoziazione, il numero di algoritmi di crittografia proposti è stato limitato. Questo non solo riduce l'overhead computazionale, ma incide anche sulla dimensione dello scambio SA init, dato che ogni algoritmo aggiuntivo comporta un incremento di 8 byte.

- *Ruolo limitato a initiator*: la versione minimale implementa esclusivamente il ruolo di initiator, senza la possibilità di fungere da responder, semplificando ulteriormente la gestione delle connessioni.

Questa implementazione minimale è progettata per funzionare con un server che offre le funzionalità complete del protocollo IKE. Le connessioni di sicurezza stabilite vengono mantenute attive solo per la durata della comunicazione e poi chiuse, evitando il mantenimento prolungato dello stato della connessione, che potrebbe risultare oneroso in termini di risorse. Confrontando l'occupazione in memoria tra srongSwan e l'implementazione si capisce il perchè è stata proposta una versione minimale. Infatti mantenere in esecuzione **charon**, il componente principale di strongSwan, richiede:

- VIRT (Virtual Memory Size): $1,194,308KB$ (circa $1.19GB$). Questa è la memoria virtuale totale che il processo ha richiesto, inclusa quella che potrebbe non essere effettivamente in uso. È una misura teorica di quanta memoria potrebbe utilizzare il processo.

- RES (Resident Set Size): $10,092KB$ (circa 10 MB). Questa è la quantità di memoria fisica (RAM) effettivamente utilizzata dal processo al momento. Questi sono i dati attivamente residenti in RAM.

- SHR (Shared Memory Size): $7,788KB$ (circa $7.8MB$). Questa è la quantità di memoria residente condivisa con altri processi, come librerie di sistema o altre risorse.

Oltre a questa memoria c'è da considerare anche quella per stabilire la connessione, che richiede utleriori $10MB$. L'implementazione proposta richiede

solamente $9028KB$ di memoria, un valore che può essere paragonato alla memoria residente (RES) di Charon. Tuttavia, è importante notare che, mentre Charon è sempre attivo, consumando continuamente risorse di sistema, l'implementazione proposta viene eseguita solo quando necessario. In questo modo, si evita anche il problema di una memoria virtuale eccessivamente grande, che può compromettere le performance complessive del sistema. Oltre all'occupazione in memoria si hanno miglioramenti anche sulla dimensione dei pacchetti. Infatti sullo scambio `INIT` se confrontiamo una versione di StrongSwan già ottimizzata e l'implementazione proposta a parità di configurazione abbiamo una riduzione del 20%, si può affermare che questa riduzione rappresenta un valore garantito. Infatti, nel caso si utilizzino configurazioni più generali, in cui vengono effettuate numerose proposte e inviati flag non essenziali, il risparmio in termini di dimensioni del pacchetto potrebbe aumentare considerevolmente. Per quanto riguarda i tempi necessari al completamento dello scambio, gli stessi risultano confrontabili dato che si utilizza la stessa implementazione delle primitive utilizzata da Strongswan.

Nel complesso, l'implementazione proposta rappresenta una base su cui costruire e sviluppare ulteriormente le funzionalità necessarie per affrontare le sfide delle comunicazioni satellitari, così come nel campo della sicurezza delle reti in contesti a bassa disponibilità di risorse.

Organizzazione della Tesi

Il proseguo della tesi sarà strutturato nel seguente modo:

- *Capitolo 2*: vengono fornite le basi matematiche della sicurezza e il modo in cui queste sono applicate alle comunicazioni digitali. In particolare, si analizza il caso di IPsec e del protocollo ausiliario utilizzato per negoziare i relativi parametri di sicurezza.
- *Capitolo 3*: viene descritto il contesto satellitare e le problematiche legate all'applicazione di algoritmi post-quantum in tali scenari. Si introduce il benchamrking realizzato per aiutare in questa decisione.
- *Capitolo 4*: affronta la progettazione e le valutazioni relative all'implementazione proposta, illustrando i miglioramenti conseguiti rispetto alle soluzioni precedenti.

- *Capitolo 5*: vengono formalizzati i risultati ottenuti e viene condotta un'analisi dettagliata sugli sviluppi dell'implementazione proposta.
- *Capitolo 6*: vengono presentate le conclusioni del lavoro svolto, riassumendo i principali risultati ottenuti dal benchmarking e dall'implementazione e l'analisi delle sue prestazioni.

Capitolo 2

Fondamenti di Comunicazioni Sicure

In questo capitolo esamineremo le problematiche relative alla sicurezza nelle comunicazioni. Inizieremo con un'analisi degli strumenti matematici fondamentali che sono alla base della protezione dei dati, esplorando le tecniche crittografiche e i loro principi teorici. Successivamente, ci concentreremo su come queste tecniche vengono effettivamente applicate per garantire la sicurezza nelle comunicazioni sulle reti di computer.

2.1 Teoria

Dalla crittografia classica, come il cifrario di Cesare, fino alle tecniche più sofisticate del ventesimo secolo, come i sistemi di cifratura a chiave pubblica, la storia della crittografia è caratterizzata da una continua evoluzione e innovazione. Nonostante questo, però, le basi matematiche e le fondamenta della crittografia sono rimaste le stesse nel corso degli anni. Nata con il solo scopo di celare informazione, oggi la crittografia si è evoluta fino a garantire proprietà diventate ormai imprescindibili per la vita quotidiana (come, appunto, la possibilità di accordarsi su di un segreto, utilizzando unicamente comunicazioni pubbliche). In questo capitolo si andranno a definire le basi necessarie per comprendere il funzionamento dei principali algoritmi crittografici.

2.1.1 Hash Function

Una funzione **hash crittografiche** è una funzione matematica che prende in input un messaggio di lunghezza arbitraria e restituisce un output di lunghezza fissa, noto come digest, ovvero

$$H : \{0,1\}^* \to \{0,1\}^n, \tag{2.1}$$

dove:

- $\{0,1\}^*$ indica l'insieme di tutte le stringhe binarie di lunghezza arbitraria;
- $\{0,1\}^n$ indica l'insieme delle stringhe binarie di lunghezza fissa n.

Le funzioni di hash crittografiche sono strumenti fondamentali nel campo della sicurezza informatica, progettate per garantire l'integrità e l'autenticità dei dati. Per questo motivo, a questo tipo di funzioni sono richieste le seguenti proprietà:

- **Resistenza alle collisioni**: devono essere progettate in modo tale che sia computazionalmente impraticabile invertire il processo, ovvero, dato il digest è dificile risalire al messaggio che lo ha prodotto.
- **Proprietà di diffusione**: una leggere variazione dell'input deve produrre un hash completamente diverso, questa è fondamentale ed imprevedibile per garantire che gli attaccanti non possano prevedere o manipolare il valore di hash a seguito di modifiche all'input. Le funzioni hash vengono spesso modellate tramite il cosiddetto Random Oracle Model (ROM): ad ogni nuovo input, l'oracolo risponde in maniera casuale. L'oracolo è però una funzione deterministica, pertanto, a parità di input risponde sempre allo stesso modo.

Di fatto, si può pensare ad una funzione hash come ad una funzione pseudorandom (quindi, imprevedibile) difficile da invertire.

2.1.2 Schemi crittografici

Uno schema di cifratura è un insieme di algoritmi e funzioni che definisce come trasformare un messaggio in chiaro (plaintext) in un messaggio cifrato (ciphertext) e viceversa, al fine di garantire la confidenzialità e la sicurezza delle comunicazioni. Formalmente può essere rappresentato come una quintupla:

$$(\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$$

composta dai seguenti elementi:

- $\mathcal{P}$: Insieme dei messaggi in chiaro (plaintext).
- $\mathcal{C}$: Insieme dei messaggi cifrati (ciphertext).

- $\mathcal{K}$: Insieme delle chiavi utilizzate per la cifratura e decifratura, *key space*. Questo insieme è a sua volta ottenuto come il prodotto tra due insiemi $\mathcal{K}_E$ e $\mathcal{K}_D$, rispettivamente, lo spazio delle chiavi di cifratura e lo spazio delle chiavi di decifratura. Come si vedrà nel seguito, i due insiemi possono anche coincidere.
- $E : \mathcal{K}_E \times \mathcal{P} \to \mathcal{C}$: Funzione di cifratura.
- $D : \mathcal{K}_D \times \mathcal{C} \to \mathcal{P}$: Funzione di decifratura.

Deve esistere una relazione inversa tra le operazioni di cifratura e decifratura, ovvero,

$$\forall K_E \in \mathcal{K}_E, \exists K_D \in \mathcal{K}_D : \quad D(K_D, E(K_E, m)) = m, \quad \forall m \in \mathcal{P}. \tag{2.2}$$

La precedente relazione può essere rilassata, richiedendo che essa sia verificata con probabilità sufficientemente alta, quando sia il messaggio m che la chiave K_E sono scelte in maniera uniforme dai corrispondenti insiemi. Per evitare di complicare la trattazione, non si andrà ad esplorare questa definizione generale. Lo schema in *Fig. 2.1*, mostra il funzionamento generale di uno schema crittografico. Tuttavia andando a caratterizzare le chiavi utilizzate nelle operazioni di cifratura e decifratura possiamo dare una prima classificazione:

- se $K_E = K_D$ allora si parla di un schema di crittografia *simmetrico*.
- se $K_E \neq K_D$ allora si parla di schema di crittografia *asimmetrico*.

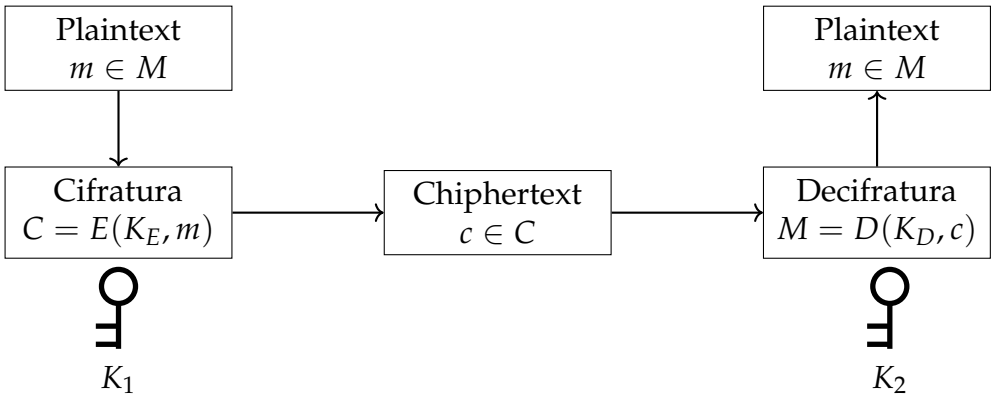

FIGURA 2.1: Descizione generale del funzionamento di uno schema crittografico

Simmetrici

Come descritto precedentemente, in uno schema simmetrico si utilizza la stessa chiave sia per le operazioni di cifratura che di decifratura. Ciò implica che le due parti coinvolte nella comunicazione debbano possedere la medesima chiave segreta, nota anche come chiave pre-condivisa (PSK - Pre-Shared Key). Questa caratteristica fondamentale rende gli schemi di crittografia simmetrica particolarmente veloci ed efficienti, esempi ne sono AES (Advanced Encryption Standard) e DES (Data Encryption Standard).
Le loro caratteristiche li rendono ideali per:

- *Cifratura* di Dati: proteggere file e database memorizzati su disco, garantendo che le informazioni sensibili rimangano riservate anche in caso di accesso non autorizzato, sia proteggere i dati mentre vengono trasmessi su reti.
- *HMAC* (Hash-based Message Authentication Code): combinati con funzioni di hash, gli algoritmi simmetrici possono generare codici, che forniscono autenticità e integrità ai messaggi. Fondamentale per garantire che i dati non vengano manomessi durante la trasmissione.

Asimmetrici

In uno schema di cifratura asimmetrica, detto anche a **chiave pubblica**, lo spazio delle chiavi $\mathcal{K}$ è costituito da una coppia di chiavi $(k_{\text{pub}}, k_{\text{priv}})$, dove:

- La chiave pubblica k_{pub} viene condivisa liberamente e utilizzata da chiunque per cifrare messaggi destinati al proprietario della chiave.
- La chiave privata k_{priv} è mantenuta segreta dal proprietario e viene utilizzata per decifrare i messaggi cifrati con la corrispondente chiave pubblica.

Quindi le due funzioni si riscrivono come:

$$E : \mathcal{K}_{\text{pub}} \times \mathcal{P} \to \mathcal{C} \tag{2.3}$$

$$D : \mathcal{K}_{\text{priv}} \times \mathcal{C} \to \mathcal{P} \tag{2.4}$$

Le due chiavi sono matematicamente legate, ma è computazionalmente difficile ottenere la chiave privata a partire da quella pubblica. Il funzionamento si basa sul concetto di **trapdoor** che rende possibile una funzione (come la cifratura o la decifratura) semplice per chi conosce un segreto (la chiave privata)

ma estremamente difficile per chi non lo conosce.

Uno dei principali utilizzi della crittografia asimmetrica è il **Key-Exchange**, il quale consente di scambiarsi un'informazione segreta su un canale pubblico.

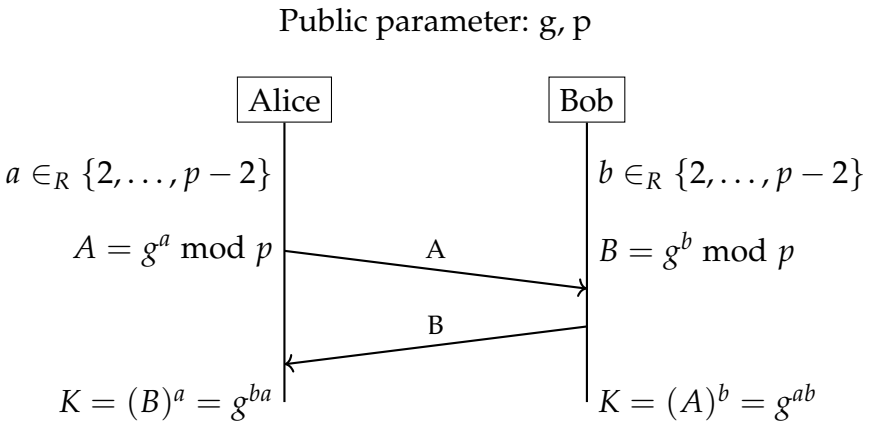

FIGURA 2.2: Scambio di Chiave Difie-Hellman basato sul problema del logaritmo discreto in gruppi moltiplicativi

La procedura più conosciuta è quella proposta da *Diffie-Hellman* ed è mostrata in *Fig.* 2.2, al termine della quale entrambi gli utenti posseggono il medesimo valore K, che può quindi essere utilizzato come chiave segreta per le comunicazioni successive. La sicurezza dello schema dipende dal fatto che la funzione utilizzata per calcolare A e B deve essere *one-way*, ovvero, facile da calcolare ma difficile da invertire.
Le **firme digitali**, sono un meccanismo chiave per garantire l'autenticità e l'integrità dei messaggi. In questo caso si fornisce sia il messaggio che un digest del messaggio firmato, in questo modo chi lo riceve può utilizzare la chiave pubblica per verificare che l'hash firmato equivalga a quello calcolato. Questa pratica si utilizza per garantire integrità e autenticità.

Key Distribution

L'assunto che si è fatto in entrambi le tipologie di schema è che l'altra parte della comunicazione avesse ottenuto in qualche modo la chiave. Tuttavia la distribuzione delle chiavi è un problema importante per il crittosistema.

Le distribuzione delle chiavi per crittosistemi simmetrici deve avvenire tramite un canale segreto, per questo motivo si utilizzano le seguenti modalità:

- *Manuale*: vengono installate manualmente coppie di chiavi per ogni nodo che si vuol far comunicare, se si vogliono far comunicare n nodi

sono necessarie $n(n-1)/2$ chiavi. Questo approccio è robusto, essendo decentralizzato, ma risulta impraticabile per reti di grandi dimensioni come mostrato in *Fig. 2.3*.

- *Key Distribution Center (KDC)*: da un'approccio decentralizzato si passa ad uno centralizzato, in cui è presente un server che fa da intermediario fidato per la distribuzione delle chiavi.

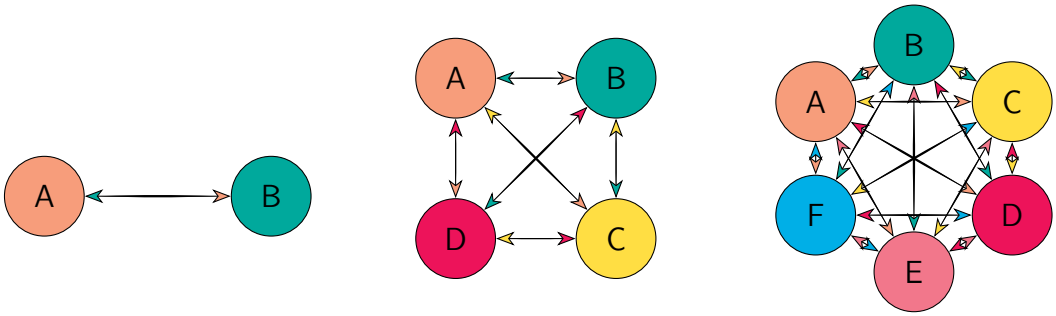

FIGURA 2.3: Esplosione combinatoria dovuta alla distribuzion manuale di chiavi in una rete con molti terminali

Per loro natura le chiavi pubbliche sono liberamente distribuite, non è dunque necessario un canale segreto. Il problema che si hanno con questa tipologia di chiavi è quello dell'autenticità, ovvero che la chiave provenga realmente dalla fonte dichiarata. Questa tipologia di schemi infatti per loro natura è vulnerabile ad attacchi MITM. Per questo nasce la *Public Key Infrastructure (PKI)*, che tramite l'utilizzo dei certificati X.509 consente la distribuzione sicura e scalabile di chiavi pubbliche.

2.1.3 Sicurezza

Il **security level** (SL) è una misura della sicurezza che una primitiva crittografica può raggiungere. Il security level corrisponde al numero minimo di operazioni elementari che devono essere eseguito per poter rompere una delle proprietà fondamentali di sicurezza di uno schema. In altre parole, il security level è il costo computazionale del più efficiente algoritmo di attacco noto. In pratica, il concetto di sicurezza di uno schema è legato al concetto di tempo: più uno schema è sicuro, più il tempo minimo necessario per attaccarlo è grande. Fino ad una decina di anni fa, un SL pari ad 80 era ritenuto accettabile. Oggi, invece, il valore minimo richiesto è aumentato sensibilmente e, di fatto, schemi con SL inferiore a 128 non sono raccomandati. Per le applicazioni critiche, si richiede che vi siano almeno 192 o 256 bit di sicurezza. Viene

espresso in bit: se uno schema ha n bit di sicurezza, significa che il miglior attacco noto impiega 2^n operazioni[1] per romperlo.

- Per i cifrari simmetrici il livello di sicurezza è pari alla dimensione del key-space. Infatti, il miglior attacco è quello a forza bruta: se la chiave è lunga n, allora vanno testate 2^n chiavi.
- La sicurezza dei cifrari asimmetrici si basa su problemi matematici. Tuttavia, gli attacchi contro gli attuali sistemi a chiave pubblica sono sempre più veloci della ricerca a forza bruta dello spazio delle chiavi.

Problemi matematici utilizzati in crittografia classica

La crittografia asimmetrica classica si basa su una serie di problemi matematici ritenuti difficili, ovvero, tali per cui un risolutore efficiente non è noto. Esempi di questi problemi sono:

- *Integer Factorization Problem (IFP)*: dato un numero intero, trovare la sua fattorizzazione in numeri primi. I migliori risolutori noti hanno un costo che è sub-esponenziale nella lunghezza binaria del numero. Il problema è alla base di RSA, uno degli algoritmi di firma digitale più diffuso.
- *Discrete Logarithm Problem (DLP)*: dato un numero primo q, un generatore g ed un numero $y \in \{0, \cdots, q-1\}$, trovare un numero $x \in \{0, \cdots, q-1\}$ tale che $y = g^x \mod q$. Il problema è alla base dello scambio di chiave Diffie-Hellman
- *DLP su Curve Ellittiche (EC-DLP)*: la formulazione matematica del problema è simile al DLP, ma utilizza operazioni su curve ellittiche. Il problema è alla base di ECDH ed ECDSA, rispettivamente, algoritmo di scambio di chiave e di firma digitale.

2.1.4 Crittografia post-quantum

Come è ben noto, tramite i computer quantistici possono essere implementati algoritmi che sono tremendamente più veloci degli algoritmi classici. Pertanto, attaccanti dotati di computer quantistici possono, potenzialmente, eseguire algoritmi di attacco che sono molto più veloci di quelli classici. In pratica, algoritmi quantistici potrebbero rompere schemi crittografici in tempi molto

[1] Tipicamente, queste grandezze vengono espresse utilizzando la notazione O grande (anche nota come notazione di Landau). Formalmente, uno schema ha n bit di sicurezza se tutti gli attacchi noti hanno un costo che cresce almeno come $2^{n \cdot \left(1+o(1)\right)}$.

inferiori a quelli richiesti da attaccanti dotati solamente di computer classici. Si nota come questa possibilità non va a modificare la definizione di security level: si vanno sempre a misurare le operazioni richieste per implementare un attacco, ma si cambia la macchina che è a disposizione dell'attaccante.
Come è ben noto, Shor nel 1994 ha pubblicato un celebre articolo in cui descrive degli algoritmi quantistici che risolvono, in tempo polinomiale, IFP, DLP ed ECDLP [1]. Questi algoritmi, ormai noti semplicemente come Algoritmi di Shor, devono essere lanciati su un computer quantistico sufficientemente potente (ad esempio, con un numero molto grande di qubit stabili).
Ad oggi, computer quantistici con queste caratteristiche non sono ancora disponibili. Infatti, vi sono problemi tecnologici non indifferenti: ad esempio, un alto numero di qubit porta ad un'elevata instabilità e, di fatto, rende i calcoli non affidabili. Va però detto che, negli ultimi anni, la situazione dei computer quantistici è migliorata notevolmente. I progressi nella tecnologia quantistica sono sempre più importanti e l'avanzamento dei computer quantistici è sotto gli occhi di tutti. Numerosi esperti ritengono che, in una decina di anni, saranno disponibili computer quantistici così potenti da poter attaccare RSA o DH. Al tempo stesso, l'algoritmo di Shor ha subito notevoli miglioramenti, per cui il numero di qubit richiesti è nel frattempo calato [4].
Il problema con la crittografia è evidente: gli algoritmi crittografici ad oggi più diffusi ed utilizzati non possono più essere considerati sicuri. Per questo motivo, serve una nuova generazione di tecniche crittografiche in grado di resistere (anche) ad attacchi implementati tramite computer quantistici. Questi algoritmi dovranno andare a rimpiazzare gli attuali algoritmi, perciò dovranno poter essere implementati tramite macchine classiche. L'area della crittografia che studia queste tecniche crittografiche prende il nome di *crittografia post-quantum*.
In pratica, algoritmi post-quantum sono basati su problemi matematici che non possono essere risolti in maniera efficiente da algoritmi quantistici.
Tipicamente questi problemi vengono classificati in base alla natura del problema considerato. Tra gli esempi più celebri, si possono considerare quelli basati su reticoli e quelli basati su codici, che possono essere descritti nel seguente modo: dato un sistema lineare sotto-determinato (più incognite piuttosto che equazioni), trovare una soluzione con caratteristiche specifiche (ad esempio, norma euclidea bassa oppure poche componenti non nulle). Problemi di questo tipo sono caratterizzati da un flavour combinatorio e, di fatto, i migliori algoritmi di attacco utilizzano sempre una ricerca a forza bruta come subroutine. Reticoli e codici sono oggetti combinatori spesso privi di

struttura algebrica, struttura che invece è presente in tutti i problemi utilizzati per la crittografia classica. Questa diversa natura fa sì che gli algoritmi di Shor non si possano applicare.

Standard NIST

Il NIST (National Institute of Standards and Technology) ha indetto una gara per la standardizzazione di schemi di crittografia post-quantum [5]. La gara, ormai in fase di chiusura, ha già prodotto quattro algoritmi vincitori: tre algoritmi di firma digitale (Dilithium, Falcon e SPHINCS+) ed un algoritmo di scambio di chiave (Kyber). Ogni versione degli algoritmi presenta tre livelli di sicurezza, chiamati *Security Category*. Per completezza, i livelli di sicurezza utilizzati sono riportati in *Tabella 2.1*.

Security Level	Descrizione
Livello 1	Sicurezza equivalente alla cifratura simmetrica con chiavi da 128 bit, come AES-128.
Livello 2	Sicurezza equivalente ad attacchi contro SHA-256, con complessità circa pari a 128 bit. Leggermente più sicuro del livello 1.
Livello 3	Sicurezza equivalente alla cifratura simmetrica con chiavi da 192 bit, come AES-192.
Livello 4	Sicurezza equivalente ad attacchi contro SHA-384. Leggermente più sicuro del Livello 3.
Livello 5	Sicurezza equivalente alla cifratura simmetrica con chiavi da 256 bit, come AES-256.

TABELLA 2.1: Security Levels definiti dal NIST

2.2 Applicazioni

Le reti sono un mezzo di comunicazione intrinsecamente insicuro, soprattutto quando operano in modalità broadcast. In questo contesto, la crittografia riveste un ruolo cruciale nel garantire la sicurezza dei dati scambiati tra entità remote. I crittosistemi, ossia le applicazioni crittografiche, integrano algoritmi di cifratura, autenticazione e gestione delle chiavi per fare in modo che vengano rispettati i requisiti di sicurezza per le informazioni trasmesse.

Il modello di riferimento per la comunicazione su Internet è il modello **TCP/IP**, il quale suddivide il processo di trasmissione dei dati in vari livelli, ciascuno con delle funzioni specifiche che non si sovrappongono con quelle degli altri livelli. Come mostrato in *Figura 2.4*, è possibile applicare la sicurezza ai vari livelli della pila e di lato sono riportati i protocolli che vengono utilizzati.

- SSH (Secure Shell): protegge l'accesso remoto e il trasferimento di file, fornendo autenticazione e cifratura.
- TLS (Transport Layer Security): permette di instaurare una connessione TCP sicura. Viene utilizzato principalemte nell'architettura C/S.
- IPsec (Internet Protocol Security): protegge i pacchetti IP scambiati tra due nodi, fornendo autenticazione, integrità e cifratura.

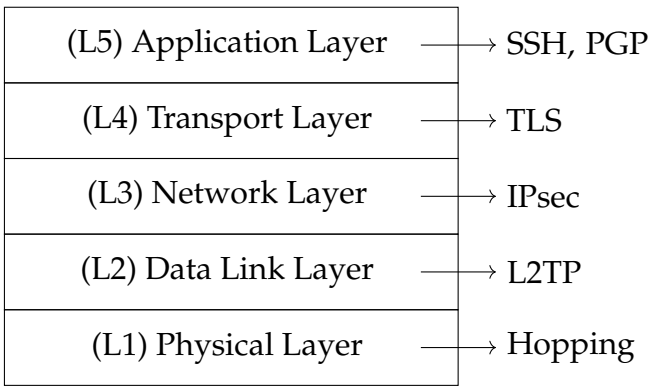

FIGURA 2.4: Rappresentazione dei livelli dello stack TCP/IP, con alcuni protocolli di sicurezza di ogni livello

Ogni protocollo avrà le proprie caratteristiche, tuttavia fare sicurezza a L3 dello stack TCP/IP offre un vantaggio significativo: poiché tutti gli strati superiori dipendono da esso per la trasmissione dei dati, non è necessario apportare modifiche ai singoli protocolli o applicazioni che si basano su di

esso. Questo consente di implementare soluzioni di sicurezza centralizzate e trasparenti, senza dover intervenire su ciascun servizio o applicazione a livello più alto.

2.3 IPsec

IPsec (Internet Protocol Security) è un insieme di protocolli standard, suddivisi tra core e ausiliari in *Figura 2.5*, utilizzati in modo tale da fornire meccanismi per l'autenticazione, la cifratura e l'integrità dei dati trasmessi tra due o più dispositivi, proteggendo così le comunicazioni IP da intercettazioni e manomissioni.

2.3.1 Architettura

Come definito dall'`RFC 1825`, l'architettura di IPsec è composta dai seguenti componenti:

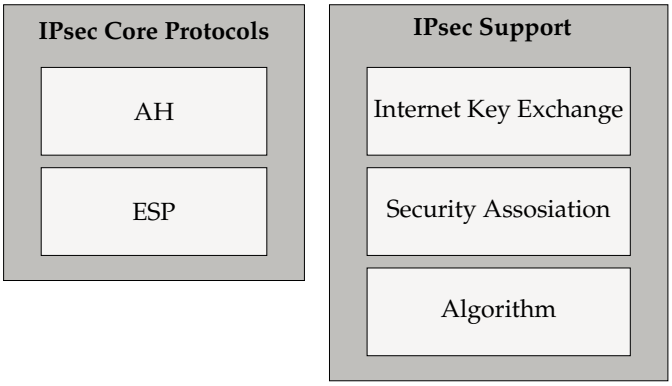

FIGURA 2.5: IPsec Protolocol Suite

- **AH (Authentication Header)**: si tratta di un protocollo di sicurezza che fornisce autenticazione e integrità dei dati, garantendo che i pacchetti non vengano modificati durante la trasmissione. Non offre cifratura, quindi i dati rimangono in chiaro.
- **ESP (Encapsulating Security Payload)**: protocolo che fornisce cifratura per garantire la riservatezza dei dati, oltre a integrità e autenticazione opzionale. ESP è il protocollo più utilizzato per garantire sia sicurezza che riservatezza.
- **SA (Security Association)**: un insieme di parametri che definisce come i dati devono essere protetti durante la comunicazione tra due entità

su una rete. Ogni SA contiene le informazioni necessarie per stabilire e mantenere una connessione sicura.

- **IKE (Internet Key Exchange)**: protocollo che consente di negoziare, autenticare e distribuire dinamicamente le chiavi crittografiche che vengono poi impiegate dai protocolli di sicurezza per proteggere le comunicazioni.
- **Algoritmi**: gli algoritmi crittografici e di hashing utilizzati per ottenere sicurezza.

2.3.2 Security Association

IP è un protocollo *stateless*, ovvero non mantiene informazioni o stato relativo alle connessioni o ai pacchetti che gestisce. Tuttavia affinchè IPsec possa garantire la sicurezza è necessario che mantenga il contesto di ogni connessione, le principali motivazioni sono:

- *Replay Protection*: per evitare attacchi di tipo replay, IPsec tiene traccia dei numeri di sequenza dei pacchetti, un'inforamzione di stato che va mantenuta per ogni connessioni e che IP non fa nativamente.
- *Connessioni Multiple*: in uno scenario di rete complesso, un singolo dispositivo potrebbe avere più connessioni sicure in corso simultaneamente, ognuna delle quali ha i propri parametri di sicurezza. IPsec deve tenere traccia di queste informazioni per sapere come trattare i pacchetti in entrata e uscita in base alla connessione a cui appartengono.
- *Protezione*: i protocolli di sicurezza AH e ESP richiedono di conoscere le chiavi crittografiche appropriate e gli algoritmi utilizzati per cifrare e decifrare i pacchetti.

IP diventa in grado di mantenere un'insieme di informazioni di stato grazie al concetto di *Security Association* (SA). Più precisamente si tratta di un'insieme di parametri che servono per associare a ciascun canale uno stato condiviso tra le entità coinvolte nello comunicazione, tra questi abbiamo:

- *Security Parameter Index* (SPI): un'identificatore della SA.
- *Destination Address*: serve all'host per determinare quale SA utilizzare.
- *Lifetime*: il tempo di vita della SA, si obbliga a refresh periodici.

- *Protocol Identifier*: determina il tipo di protezione da applicare ai pacchetti, dunque anche chiavi e algoritmi associati.
- Altri parametri opzionali, la lista completa è definita nell'`RFC 1825`.

La SA è caratterizzata dall'essere un canale *simplex*, dunque al fine di stabilire un canale di comunicazione bidirezionale IPsec tra due entità occorrono due SA unidirezionali di verso opposto. La *Figura 2.6* mostra il tunnel virtuale in esecuzione tra i due host.

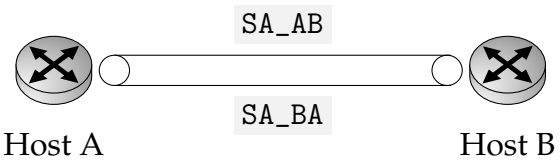

FIGURA 2.6: Security Association bidirezionali per realizzare una connessione VPN IPsec

2.3.3 Negoziazione SA

In un contesto come quello delle reti potremmo avere che lo stesso nodo ha connessioni IPsec multiple, le SA consentono di distinguere e identificare in modo univoco la configurazioni di sicurezza da applicare alla comunicazione. Tuttavia queste SA come si configurano?
IPsec prevede tecniche di negoziazione delle SA di tipo:

- *Manuale*: occorre configurare manualmente le chiavi e le impostazioni di sicurezza per ciascun dispositivo o punto finale di comunicazione.
- *Automatico*: si utilizzano protocolli per stabilire automaticamente le chiavi di crittografia e le politiche di sicurezza senza intervento umano diretto.

L'utilizzo di tecniche di negoziazione automatica offre un approccio sicuro, flessibile e scalabile alla gestione delle Security Association, un esempio di questo è IKE. Andiamo a vedere nel dettaglio IKE nella prossima sezione.

2.4 IKE

Il protocollo IKE (Internet Key Exchange) definisce una serie di scambi, come illustrato in *Figura 2.7*, al termine dei quali i due peer negoziano i parametri di sicurezza e le chiavi crittografiche necessarie per stabilire una Security Association (SA). Poiché le SA devono essere conosciute esclusivamente dai due terminali coinvolti, IKE assume il compito di apri pista, instaurando un canale sicuro per la comunicazione. Questo processo è possibile solo se i peer riescono a condividere un segreto attraverso un canale pubblico, e proprio questo è il compito principale del protocollo IKE.

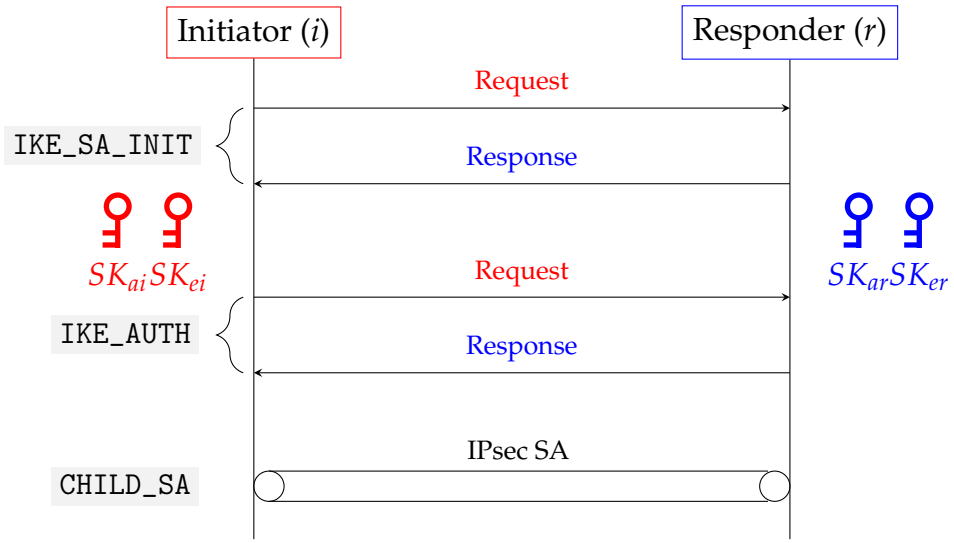

FIGURA 2.7: Fasi di Negoziazione del Protocollo IKEv2

`IKE_SA_INIT`

Lo scopo di questa prima fase è quello di creare una **IKE SA**, che consenta di rendere sicuri i successivi scambi di dati al fine di realizzare una **IPsec SA**. Quindi un primo scambio che consente di accordarsi su come avverranno le comunicazioni successive e la condivisione di un segreto. Per questo motivo i messaggi scambiati tra i due in questa fase contiene i payload specificati in *Tabella 2.2*.

Al termine di questo scambio i due peer completano il Key Exchange. Attraverso i payload che si sono scambiati, nel caso classico, sono in grado di derivare il *DH Shared Secret*. Questo insieme ai nonce, consente di ottenere i parametri di sicurezza della `IKE SA` al fine di instaurare il canale sicuro. Per approfondimenti fare riferimento all' Appendice.

Parametro	Descrizione
SA	Security Association, vengono negoziati gli algoritmi che verranno utilizzati per proteggere le comunicazioni.
KE	Key Exchange, e nel caso classico è l'esponente DH.
N	Nonce, un valore arbitrario utilizzato solo una volta in una comunicazione.

TABELLA 2.2: Tabella dei parametri e delle descrizioni del contenuto di un pacchetto `INIT`

`IKE_AUTH`

Il risultato della fase precedente è un canale sicuro su cui comunicare. Su questo hanno luogo gli scambi per instaurare la IPsec SA. In questa fase i nodi effettuano mutuamente autenticazione, che può avvenire mediante:

- PSK (Pre-Shared Key),
- EAP (Extensible Authentication Protocol) ,
- Certificati di chiave pubblica.

La modalità di autenticazione di maggior interesse per il nostro studio è quella mediante certificati `X.509`, poichè rappresenta la soluzione più scalabile. In questa fase i peer si scambiano i payload definiti in *Tabella 2.3*

Parametro	Descrizione
ID	Specifica l'identità del peer che può essere un indirizzo IP, un nome di dominio o un altro identificatore.
AUTH	Payload che deve essere firmato affinchè ci sia autenticazione. Questo per evitare attacchi MITM.
CERT	Si allega il certificato digitale di chiave pubblica.
CERTQ	Si fa richiesta al peer di fornire il certificato.
SA2	Si negoziano i parametri di sicurezza per la Child SA.
TS	Sta per Traffic Selector e specifica delle caratteristiche del traffico che il peer desidera utilizzare, come protocolli e porte.

TABELLA 2.3: Tabella dei parametri e delle descrizioni del contenuto di un pacchetto `AUTH`

Questi payload sono protetti tramite l'applicazione degli algoritmi e chiavi di autenticazione e cifratura derivate dallo scambio precedente. E al termine

di questo scambio i due peer hanno condiviso i parametri di sicurezza per stabilire la SA.

Altri

Oltre agli scambi `INIT` e `AUTH`, nel protocollo IKE sono presenti altre due tipologie di scambi: `CHILD_SA` e `INFORMATIONAL`. Questi scambi non verranno trattati in modo approfondito in quanto sono considerati accessori e non fondamentali per la comprensione della struttura principale del protocollo.

Lo scambio `CHILD_SA` viene utilizzato per creare e gestire una Security Association secondaria all'interno di un contesto di connessione già stabilito. Questo scambio consente la negoziazione di nuove chiavi e parametri di sicurezza specifici per il traffico, permettendo di separare le comunicazioni relative a diverse applicazioni o flussi di dati. È particolarmente utile in scenari in cui si desidera stabilire più tunnel sicuri senza dover ripetere l'intero processo di autenticazione di IKE. In sostanza, lo scambio `CHILD_SA` viene attivato quando un peer desidera negoziare ulteriori SAs senza dover ricominciare il processo di autenticazione completo.

D'altra parte, gli scambi `INFORMATIONAL` sono impiegati per inviare messaggi di controllo e informazioni tra i peer, senza richiedere azioni specifiche o modifiche alle SA esistenti. Questi messaggi possono segnalare eventi, come errori o notifiche riguardanti la chiusura delle SA, e sono essenziali per gestire lo stato delle connessioni in atto. Sebbene gli scambi `INFORMATIONAL` non influenzino direttamente la sicurezza delle comunicazioni, sono fondamentali per il monitoraggio e la gestione delle stesse.

2.4.1 Problemi

Il protocollo IKEv2 utilizza UDP come protocollo di trasporto per inoltrare i propri messaggi. La maggior parte dei messaggi scambiati tra i peer ha dimensioni relativamente piccole e non supera l'**MTU** di un pacchetto IP. Tuttavia, esistono scambi che richiedono il trasferimento di dati di dimensioni maggiori. Ad esempio, nel caso di autenticazione tramite chiave pubblica durante la fase di `IKE_AUTH`, è necessario trasferire il certificato, il cui peso può variare a seconda dello schema di firma utilizzato, arrivando anche a diversi kilobyte. In tali situazioni, può verificarsi la frammentazione a livello IP. Un altro problema è quello che riguarda il rendere lo scambio di chiave

svolto nella fase di INIT resistente ad attacchi quantum.

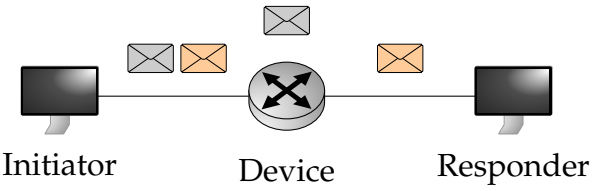

FIGURA 2.8: Drop dei pacchetti causati dai dispositivi degli ISP

IKE Fragmentation

Diverse prove hanno dimostrato che, nel caso in cui i peer si trovino in presenza di CGNAT (Carrier-grade NAT chiamato anche Large Scale NAT) , potrebbero non essere in grado di instaurare le SA. Questo comportamento è attribuibile al fatto che i dispositivi degli ISP non consentono il passaggio dei frammenti IP, scartando i pacchetti e, di conseguenza, bloccando le comunicazioni IKE. Questa situazione è schematizzata nella *Figura 2.8*. Il blocco dei pacchetti avviene poiché esistono numerosi vettori di attacco che si basano sulla frammentazione IP; per questo motivo, gli ISP implementano un filtro su tale tipologia di pacchetti. Sebbene in teoria uno dei requisiti del CGNAT, come definito dagli RFC, sia proprio quello di consentire la frammentazione, in pratica ciò non avviene.

Per affrontare questa problematica e consentire il passaggio dei messaggi attraverso i dispositivi di rete che non permettono il transito dei frammenti IP, l'RFC 7383 [6] introduce la *Message Fragmentation* di IKEv2. In questa modalità, la frammentazione dei messaggi è gestita direttamente da chi implementa IKEv2.

Intermediate

È essenziale avere la possibilità di eseguire uno o più scambi di chiavi post-quantum in combinazione con uno scambio di chiavi (EC)DH, in modo che la chiave condivisa risultante sia resistente agli attacchi da parte di computer quantistici. Poiché attualmente non esiste uno scambio di chiavi post-quantum che sia stato studiato con la stessa attenzione dell' (EC)DH, eseguire più scambi di chiavi con diversi algoritmi post-quantum insieme ai ben consolidati algoritmi di scambio di chiavi classici affronta questa preoccupazione, poiché

la sicurezza complessiva è almeno tanto forte quanto ciascuna delle singole primitive.

A tale scopo viene introdotto uno scambio aggiuntivo che viene eseguito tra la fase di `INIT` e la fase di `IKE_AUTH`, come mostrato in *Figura 2.9* in questo lo scambio è sia autenticato che cifrato tramite le chiavi negoziate dal primo scambio.

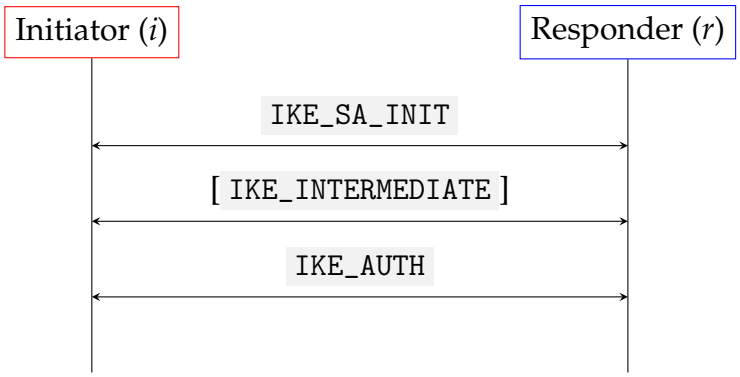

FIGURA 2.9: Scambio introdotto per consentire un KEM utilizzando primitive post-quantum

Questo scambio è stato posizionato tra dqueste due fasi poiché, in quella di `INIT` non è possibile applicare la frammentazione per motivi di sicurezza. Sebbene i messaggi siano generalmente di dimensioni contenute e non causino frammentazione IP, l'uso di scambi di chiavi QC-resistant potrebbe alterare questa situazione, in quanto tali chiavi pubbliche sono di dimensioni maggiori.

Lo scambio è stato concepito per il trasferimento di grandi quantità di dati. L'uso principale è il trasferimento delle chiavi pubbliche QC-resistant in modo tale da realizzare un **enforcing** delle chiavi negoziate tramite DH e rendere tali chiavi resistenti ad attacchi con computer quantistici. In termini tecnici, come definito dall'RFC [7], questo consente il *Multiple Key Exchange* e tali scambi aggiunti vengono aggiunti alla proposal con la sequente notazione `PQ_KEM_i`. Ad ogni ulteriore KEM si va ad aggiornare il KEY Material e si aggiornano le chiavi $SK_{a[i/r]}, SK_{e[i/r]}$.

Per specificare il supporto alle funzionalità descritte dai precedenti RFC, sono stati introdotti i seguenti flag:

- il flag `IKE_FRAGMENTATION_SUPPORT` indica che il peer supporta la frammentazione IKEv2; affinché possa essere utilizzata, entrambi i peer devono supportarla.

- il flag `INTERMEDIATE_EXCHANGE_SUPPORT` segnala che il peer è in grado di gestire scambi intermedi.

Al termine di questi scambi, per garantire la protezione dello scambio `IKE_AUTH` e degli scambi successivi, vengono utilizzate le ultime chiavi calcolate. Sebbene sia stata considerata l'implementazione della funzionalità di multiple key exchange, quella relativa a IKE fragmentation non è stata inclusa nel progetto. Questo perché, nel contesto delle comunicazioni satellitari, non è presente la necessità di gestire il NAT, il quale potrebbe introdurre ritardi significativi dovuti alla traduzione degli indirizzi. Pertanto, l'assenza di NAT nel sistema satellitare rende superflua l'applicazione della frammentazione IKE, poiché non si prevede la necessità di suddividere i pacchetti per l'inoltro attraverso dispositivi di rete che non supportano la frammentazione.

Capitolo 3

Scenario

In questo capitolo facciamo un introduzione su quello che è lo scenario che stiamo considerando per il nostro lavoro: quello delle comunicazioni satellitare. Vediamo nel dettaglio quali sfide introduce, le tecnologie utilizzate e come affrontare le possibili problematiche che possono derivare dall'applicazione della PQC.

3.1 Comunicazioni Satelliari

Le comunicazioni satellitari sono fondamentali per le infrastrutture moderne, poichè abilitano una vasta gamma di servizi. Negli ultimi decenni, con l'aumento della domanda di connettività globale e l'espansione delle reti di comunicazione, i satelliti sono diventati strumenti essenziali per garantire una copertura estesa. L'emergere delle costellazioni di satelliti in orbita bassa (LEO - Low Earth Orbit) sta cambiando il paradigma delle comunicazioni satellitari, offrendo vantaggi significativi rispetto ai satelliti geostazionari (GEO), un confronto tra le due orbite è mostrato in *Figura 3.1*. Questo cambio di paradigma insieme al quantum computer hanno portato diversi enti, tra cui l'Agenzia Spaziale Europea (ESA), ad affrontare nuove sfide.

3.1.1 Limitazioni

L'ambiente spaziale, caratterizzato da radiazioni intense, temperature estreme e lunghi periodi senza manutenzione, pone sfide significative in termini di progettazione e operatività dell'hardware e del software. Tra i principali vincoli per l'hardware satellitare troviamo:

- *Resistenza alle radiazioni*: i componenti elettronici sono progettati per resistere all'esposizione costante alle radiazioni spaziali.

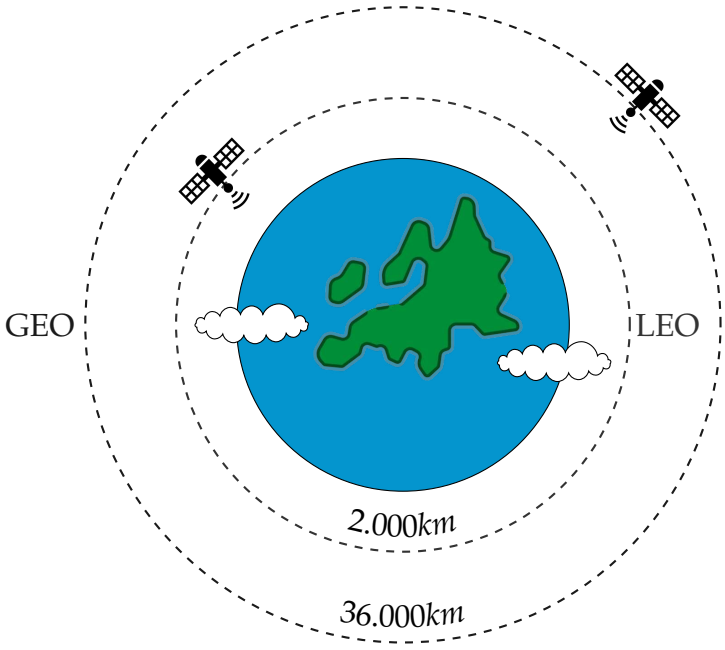

FIGURA 3.1: Confronto tra le orbite dei satelliti

- *Basso consumo energetico*: l'energia disponibile per le operazioni risulta limitata, quindi si usano processori a basso consumo e ad alta efficienza energetica, sacrificando potenza di calcolo.
- *Elaborazione in tempo reale*: per alcune tipologie di servizi è necessario che il processamento avvenga in tempo reale.
- *Compattezza*: a causa dello spazio limitato a bordo di un satellite, i componenti hardware devono essere progettati in modo estremamente compatto.

L'hardware limitato ha un impatto diretto sullo sviluppo del software per i satelliti. Rispetto al contesto terrestre, dove le risorse computazionali sono abbondanti, il software per i satelliti deve essere ottimizzato per funzionare su processori con bassa potenza di calcolo e memoria ridotta. Le principali sfide per gli sviluppatori sono:

- *Semplicità e ottimizzazione*: gli algoritmi devono essere semplici e ottimizzati per funzionare su hardware con risorse limitate.
- *Parallelismo limitato*: le operazioni devono essere eseguite in modo lineare o con limitato parallelismo, aumentando la complessità della progettazione.

- *Affidabilità assoluta*: il software deve essere robusto, sicuro e testato ampiamente in modo tale che possibile errori non abbiano conseguenza catastrofiche.

È fondamentale prestare particolare attenzione alle implementazioni crittografiche dato che in contesto come questo, risulta cruciale trovare un equilibrio tra sicurezza e prestazioni. Da un lato, è necessario proteggere le comunicazioni utilizzando algoritmi crittografici complessi; dall'altro, è essenziale garantire che le operazioni eseguite non compromettano l'operatività del satellite.

3.1.2 Stato Attuale

Per capire quale è la differenza in termini di hardware e software rispetto a quelli a cui siamo abituati introduciamo quelli che sono gli attuali standard impiegati in questo settore.

- Tra i processori utilizzati abbiamo *LEON3*, un processore open-source basato sull'architettura SPARC, progettato dall'ESA.
- ESA Power Interface Standard (ECSS-E-ST-20C): definisce come deve avvenire la distribuituzione dell'alimentazione elettrica all'interno dei satelliti.
- Cubesat Standard: definisce dimensioni compatte modulari (10x10x10 cm per 1U) per ridurre i costi e semplificare il lancio e la costruzione dei satelliti.
- Triple Modular Redundancy (TMR): nei sistemi critici spaziali si utilizza la ridondanza tripla modulare, per garantire l'affidabilità dei risultati tramite sistemi di voting.

Il sistema operativo utilizzato in queste applicazioni è RTEMS (Real-Time Executive for Multiprocessor Systems) che le caratteristiche di essere open-source, real-time e basato su GNU/Linux.

3.1.3 Sfide

Per rendere le comunicazioni satellitari sicure ad attacchi quantum occorre adottare algoritmi più complessi. Tuttavia, oltre a richiedere più memoria per la gestione delle chiavi, comportano un incremento significativo del carico di calcolo. Le sfide da affrontare sono:

- Aumento complessità computazionale: l'incremento delle dimensioni delle chiavi e della complessità degli algoritmi portano a un aumento del carico computazionale. Aggiungendo ulteriori vincoli su sistemi già limitati.
- Incremento della larghezza di banda necessaria: oltre all'aumento di carico si ha anche maggiore utilizzo della larghezza di banda, che in comunicazioni satellitari risuta già limitata.
- Compatibilità retroattiva: occorre adottare soluzioni ibride, in cui algoritmi crittografici classici coesistono con quelli post-quantum. Questo a causa dell'eterogeneità delle capacità dei satelliti in orbita.

3.2 Benchmarking

Per valutare l'applicabilità degli algoritmi di PQC nel contesto satellitare, è fondamentale comprendere prima di tutto quale è il loro impatto in degli ambienti che non sono soggetti a così tanti limiti e che consentono un confronto. Per farlo analizzeremo il loro utilizzo in ambienti desktop e lo paragoneremo con quello che è il loro utilizzo attuale.

3.2.1 Ambiente

Il protocollo che consideriamo è IPsec, con un focus particolare sulla sua implementazione in StrongSwan. L'implementazione delle primitive post-quantum è fornita dalla libreria `liboqs`. Questa fa parte del progetto open-source *OpenQuantumSafe* (OQS). Il quale punta a rendere disponibile l'infrastruttura crittografica necessaria per proteggere i sistemi informatici dall'avvento dei computer quantistici. L'obiettivo del progetto è quello di fornire:

- API standardizzate per supportare lo scambio chiavi e la firma digitale.
- Modularità, facilitando l'aggiunta di nuovi algoritmi.
- Prestazioni ottimizzate per diverse architetture hardware.

Si tratta di una raccolta di implementazioni di algoritmi crittografici post-quantum, KEM e SIG, e strumenti per integrarli in protocolli di sicurezza esistenti per sperimentare e testare il loro impatto. In *Tabelle 3.1* è presente una descrizione dettagliata dell'ambiente utilizzato per fare i test.

Componente	Descrizione
Hardware	
CPU	Ryzen 7-5825U (8 core, 3.8 GHz)
RAM	24 GB DDR4
Storage	1024 GB SSD NVMe
OS	
Distribuzione	Arch Linux
Kernel	Linux 6.10.10-arch1-1
Software	
Linguaggio	Bash Script
Strongswan	6.0.0beta Post-Quantum IKEv2 Daemon
liboqs	Version 0.9.2
Docker	Version 24.0.6

TABELLA 3.1: Descrizione dell'ambiente di test virtualizzato

3.2.2 Metodologia

Per ragioni progettuali e di portabilità del codice, il servizio Strongswan è stato containerizzato tramite l'utilizzo di Docker. Ciò consente di avere due istanze dello stesso servizio in esecuzione in grado di comunicare attaverso un'interfaccia virtuale.

```
+-------+                          +--------+
| carol | === Virtual Interf. === |  moon  |
+-------+                          +--------+
```

In secondo luogo siamo andati a definire quali sono le configurazioni da confrontare, riportate in *Tabella 3.2*, ognuna delle quali è caratterizzata da:

- **Chiper suite**: un insieme di algoritmi crittografici che determinano la sicurezza di una connessione in un protocollo di rete. Le cipher suite sono generalmente denominate seguendo una convenzione di naming standardizzata che riflette i componenti inclusi nella suite.

  `<ENCR>-<INTEG>-<KEM>`

- **Authentication Method**: sono supportati diverse modalità di autenticazione, tuttavia noi vogliamo vedere come si comportano gli schemi di firma post-quantum. Per questo motivo utilizzeremo l'autenticazione mediante certificati.

Nella nostra analisi, abbiamo scelto tre configurazioni distinte per le cipher suite, ognuna progettata per affrontare specifici aspetti delle tecnologie crittografiche attuali e future.
La prima configurazione utilizza esclusivamente *primitive classiche*. Questa scelta rappresenta il benchmark attuale delle tecnologie di crittografia e fornisce una base solida per confrontare le altre configurazioni. La seconda configurazione è composta da *primitive post-quantum*, e consente di analizzare le prestazioni e l'efficacia delle soluzioni crittografiche post-quantum nel contesto del protocollo. Infine, abbiamo implementato una *configurazione Ibrida*, la quale combina elementi delle primitive classiche e post-quantum. Questa scelta è progettata per garantire la transizione graduale verso l'utilizzo esclusivo di tecnologie quantum-resistant.

Name	**Chiper Suites**	**Firma Digitale**
A1	aes128ctr-sha256-ecp256	ECDSA
B1	aes128ctr-sha256-kyber1	dilithium2
C1	aes128ctr-sha256-ecp256-ke1_kyber1	falcon512
A3	aes192ctr-sha384-ecp384	ECDSA
B3	aes192ctr-sha384-kyber3	dilithium3
C3	aes192ctr-sha384-ecp384-ke1_kyber3	falcon1024
A5	aes256ctr-sha512-ecp521	ECDSA
B5	aes256ctr-sha512-kyber5	dilithium5
C5	aes256ctr-sha512-ecp521-ke1_kyber5	falcon1024

TABELLA 3.2: Cipher Suites suddivise per Livello di Sicurezza

Il testing viene realizzato attraverso uno script Bash. Nella Figura 3.2 è illustrata la struttura dei file necessari per automatizzare l'intero processo, consentendo così un'operazione completamente "zero-touch", che riduce al minimo l'intervento manuale. Tra questi abbiamo:

- `Dockerfile`: a partire da un'immagine Ubuntu si installa il servizio StrongSwan e si integra la libreria liboqs.
- `docker-compose`: consente di orchestrare i due container, in cui si specificano volumi e parametri di connessione tra i due.

Ad ogni container è associato a un volume, identificato con il proprio nome, che contiene le configurazioni specifiche (connessioni e certificati) per il

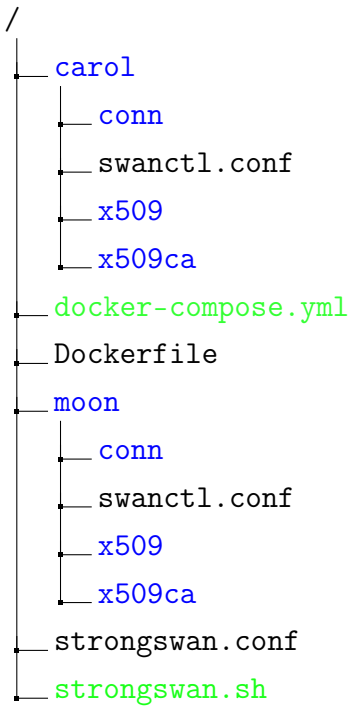

FIGURA 3.2: Struttura delle directory del progetto per automatizzare il processo di testing

daemon. Questa struttura facilita la modifica e il testing delle diverse configurazioni senza influenzare l'altro container, contribuendo a un ambiente di test più controllato e flessibile.
Per automatizzare il processo di testing e garantire un'esecuzione efficiente, è stato sviluppato uno script in Bash che gestisce l'intero flusso operativo. Questo script si basa esclusivamente su utility integrate di Linux, eliminando la necessità di installare software aggiuntivo. Esso non solo avvia i container e configura i parametri necessari, ma si occupa anche della raccolta e dell'analisi dei risultati. In *Figura 3.3* è riportato il suo diagramma di flusso in particolare le principali fasi.

3.2.3 Risutlati

Le principali metriche di interesse per il nostro studio sono state selezionate con l'obiettivo di valutare l'overhead introdotto dall'adozione delle nuove primitive crittografiche influenzi aspetti chiave come l'efficienza della comunicazione e la sicurezza. Queste sono:

- *Tempo Complessivo* per stabilire la SA tra i due, il tempo di elaborazione del singolo algoritmo non ci interessa poichè ampiamente descritto dalla libreria openquantumsafe.

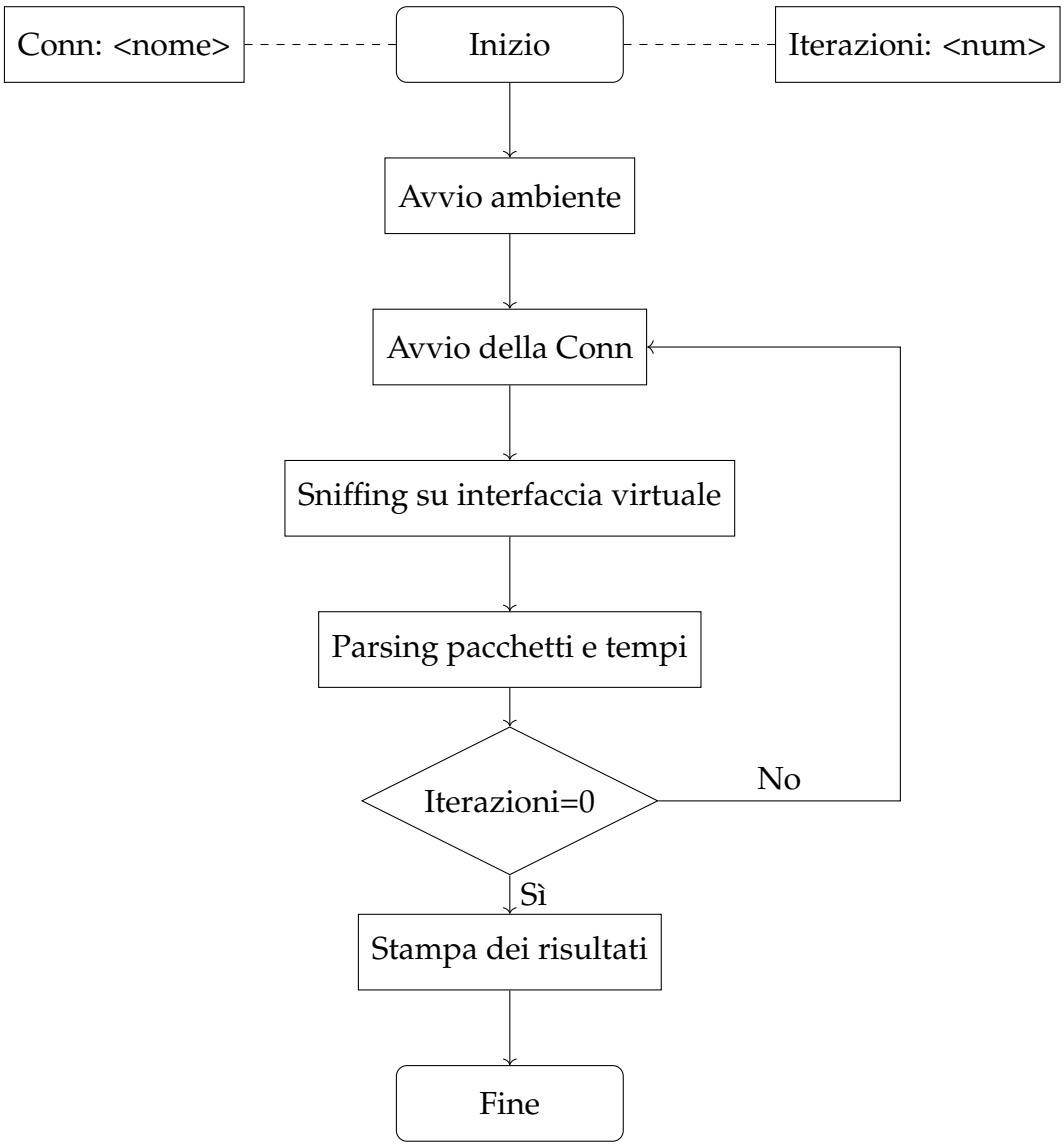

FIGURA 3.3: Diagramma di flusso dello script per il testing delle connessioni

- *Packet Size* che rappresenta la quantità di dati necessaria affinchè i due nodi siano in grado di stabilire una Security Association (SA).

In *Tabella 3.3* sono riportati i risultati delle simulazioni per ciascuna configurazione. Abbiamo preso in esame le principali metriche per ogni fase di scambio del protocollo, permettendo così un confronto più dettagliato tra le varie configurazioni. Inoltre, per ciascuno scambio è stata fatta una distinzione tra le operazioni dell'initiator e quelle del responder.
Le metriche riportate sono state ottenute iterando il processo di connessione 50 volte, mediando così i risultati per un'analisi più accurata delle prestazioni.

Nell'analisi che segue, lo scambio intermediate, sebbene riportato separatamente nella tabella, è considerato parte integrante del processo di `INIT`. Infatti, questo scambio viene utilizzato sia per trasmettere dati che non rientrano nel primo scambio, sia per effettuare un KEM post-quantum, il quale è comunque parte del processo di `INIT`.

Connessione	Scambio	Dati Trasferiti (Byte)		Tempo (ms)
		Inviati	Ricevuti	
A1	INIT	326	334	1.102
	INTE	-	-	-
	AUTH	632	570	0.977
B1	INIT	1062	1038	0.982
	INTE	-	-	-
	AUTH	6665	6663	1.1583
C1	INIT	342	350	0.709
	INTE	911	879	0.912
	AUTH	2623	2591	1.000
A3	INIT	358	366	1.407
	INTE	-	-	-
	AUTH	638	578	1.486
B3	INIT	1446	1358	0.767
	INTE	-	-	-
	AUTH	9210	9180	1.402
C3	INIT	374	382	1.642
	INTE	1303	1207	2.488
	AUTH	4845	4831	1.210
A5	INIT	394	402	1.181
	INTE	-	-	-
	AUTH	646	587	1.805
B5	INIT	1514	1514	0.887
	INTE	350	358	1.672
	AUTH	12501	12471	1.303
C5	INIT	410	418	1.108
	INTE	1729	1729	1.672
	AUTH	4856	4832	1.622

TABELLA 3.3: Risultati ottenuti dall'analisi delle configurazioni

I grafici riportati di seguito illustrano i risultati delle simulazioni effettuate

per le diverse configurazioni. In ascissa sono riportate le metriche specificate mentre in ordinata si trovano le configurazioni raggruppate per tipologia.

Packet Size

In *Figura 3.4 e 3.5*, sono riportati i grafici che considerano la dimensione dei pacchetti per gli scambi. Per lo scambio `INIT`, il passaggio da una configurazione classica ad una quantum non è poi così tanto significativa. Stessa cosa vale se si considerano le configurazioni ibride, che considerato eseguono due KEM sono contenute in dimensione. Tuttavia, lo scenario cambia per lo scambio `AUTH`, dove l'adozione di schemi di firma post-quantum introduce un overhead significativo dovuto principalmente alla maggiore dimensione delle chiavi.

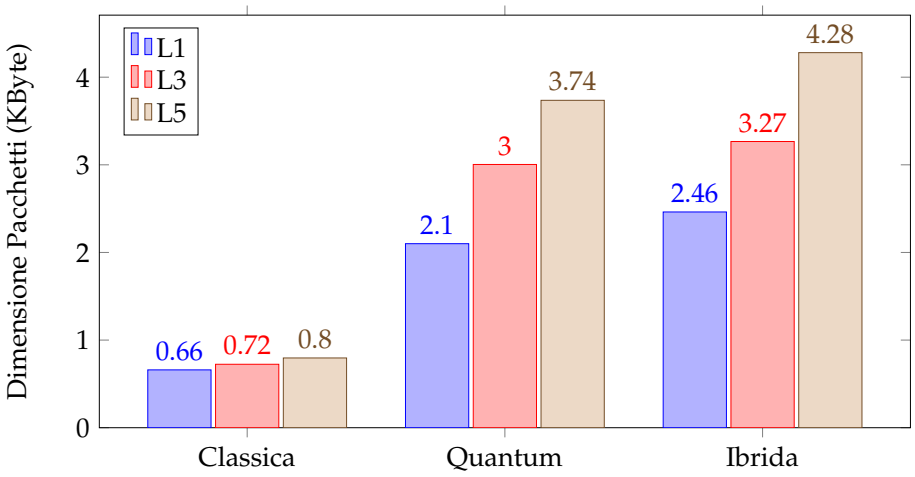

FIGURA 3.4: KByte totali dei messaggi scambiati nella fase di `INIT`, con la configurazioni raggruppate per tipologia

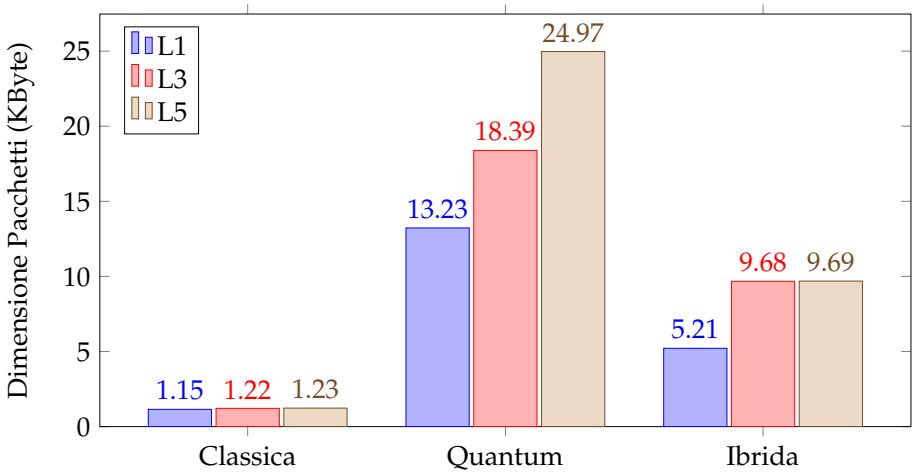

FIGURA 3.5: KByte totali dei messaggi scambiati nella fase di AUTH, con le configurazioni raggruppate per tipologia

Tempi

In *Figura 3.6 e 3.7*, sono riportati i grafici che considerano il tempo impiegato dai due nodi per completare lo scambio. Per la fase AUTH, i tempi di esecuzione risultano abbastanza simili tra le diverse configurazioni, senza variazioni significative. Al contrario, per la fase INIT, si osserva un aumento considerevole dei tempi nel caso delle configurazioni ibride, ciò è dovuto alla necessità di svolgere due KEM. Una nota importante è quella sulle prestazioni di Kyber che risulta molto veloce, anche rispetto alla configurazione classica. L'incremento nella configurazione L5 è dovuto al tempo necessario a riassemblare i pacchetti che contengono il keymaterial, dato che eccedono la dimensione massima.

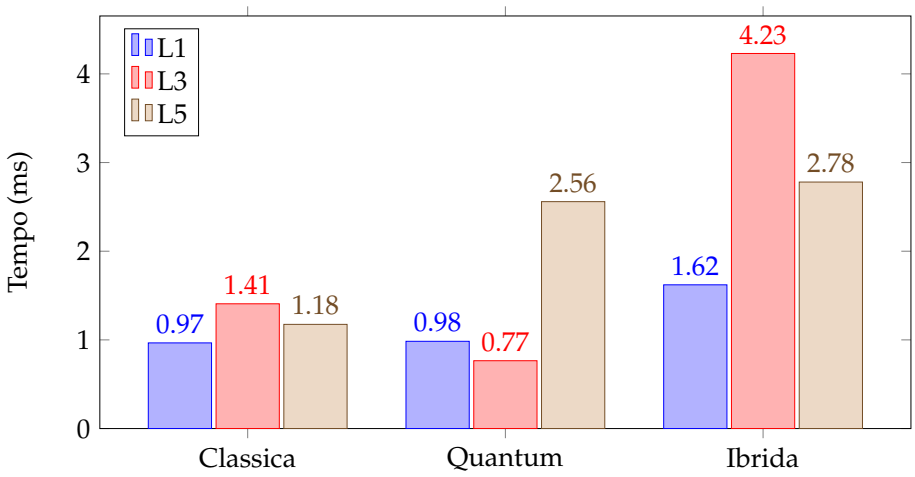

FIGURA 3.6: ms totali per la fase di INIT, con le configurazioni raggruppate per tipologia

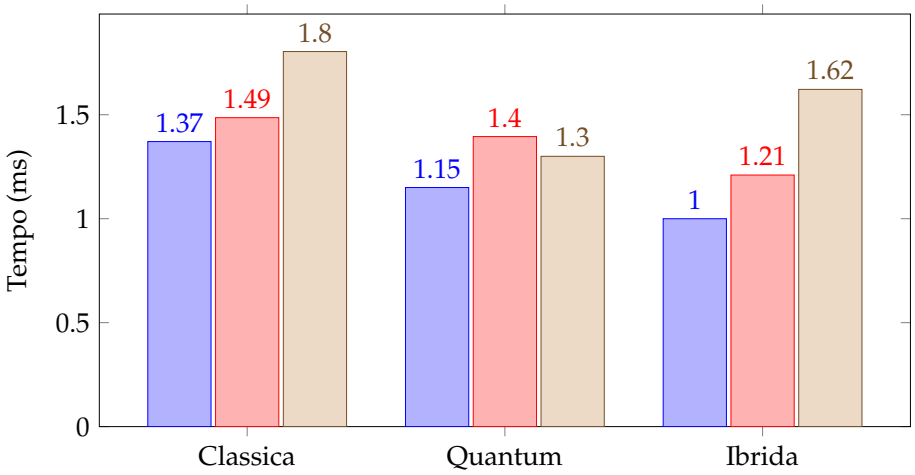

FIGURA 3.7: ms totali per la fase di AUTH, con le configurazioni raggruppato per tipologia

3.3 Osservazioni

L'analisi dei risultati evidenzia che il passaggio dall'utilizzo di primitive classiche a quelle post-quantiche comporta un incremento significativo delle dimensioni dei pacchetti. In particolare, si osservano aumenti percentuali notevoli: il 218% di incremento nella fase di INIT e un incremento pari al 1050% durante la fase di AUTH, solamente considerando il L1. Questi valori evidenziano una crescita esponenziale della dimensione dei dati trasmessi, il che solleva importanti questioni relative all'applicabilità di queste soluzioni in contesti operativi reali, come quello satellitare. In termini di prestazioni temporali, è interessante notare che, contrariamente all'aumento delle dimensioni, i tempi di esecuzione mostrano una riduzione in entrambi gli scambi. Questo potrebbe sembrare positivo, ma è fondamentale considerare che la diminuzione dei tempi non compensa l'incremento delle dimensioni.

L'implementazione di Strongswan è tale da potersi adattare a più scenari possibili. Infatti supporta numerosissime modalità di autenticazione e altre funzionalità che in un contesto limitato, come quello dei satelliti, non ha senso portarsi dietro dato che non trovano applicazione.
Per cercare di risolvere il prolema di applicare IKEv2 anche ad un contesto che presenta numerosi limitazioni computazionali, ci siamo imbattuti su **Minimal IKE**. Una versione ottimizzata che lascia lo stretto necessario in modo tale da poter applicare IKE anche negli scenari descritti. Tuttavia, attualmente non esistono implementazioni concrete. Pertanto, abbiamo intrapreso la creazione di una nostra implementazione, con l'obiettivo di fornire un punto di partenza

per questo tipo di scenario.

Capitolo 4

Implementazione di Minimal IKE

In questo capitolo, ci concentreremo sull'implementazione di Minimal IKE, un approccio progettato per ottimizzare le comunicazioni in scenari a risorse limitate, come quelli satellitari. Inizieremo analizzando la fase di progettazione e procederemo con l'implementazione, evidenziando le tecniche utilizzate per ridurre le risorse e l'occupazione di memoria. Minimal IKE semplifica il protocollo, eliminando funzionalità non necessarie e concentrandosi su ciò che è essenziale per garantire comunicazioni sicure ed efficienti.

4.1 Progettazione

Questo approccio consente di passare a un modello di rete in cui è il dispositivo a risorse limitate a contattare il server, evitando di portare con sé tutte le funzionalità necessarie per svolgere il ruolo di responder, un compito che non deve eseguire. Si passa così a un modello ad interrupt, che consente alle connessioni di essere effimere. Questa modalità si traduce in una riduzione dei costi computazionali e in un aumento della sicurezza, poiché le associazioni di sicurezza (SA) vengono create e distrutte continuamente, minimizzando così il tempo in cui le chiavi sono attive e riducendo l'esposizione a potenziali attacchi.

4.1.1 Requisiti

I requisiti che guideranno il processo progettuale e realizzativo dell'implementazione sono:

- **Utilizzo ottimizzato della memoria**: l'implementazione deve ridurre al minimo l'occupazione della memoria, garantendo efficienza anche in ambienti con risorse limitate.

- **Ruolo di initiator**: Deve svolgere esclusivamente il compito di initiator, evitando il carico di responsabilità del ruolo di responder.
- **Completamento dello scambio di messaggi**: Deve essere in grado di portare a termine lo scambio di messaggi necessario per stabilire una Security Association (SA).
- **Semplicità operativa**: L'implementazione deve evitare funzionalità non necessarie per semplificare il processo di negoziazione e ottimizzare le risorse computazionali.

Pur essendo semplici nella loro formulazione, il loro soddisfacimento rappresenta una sfida significativa. Inoltre tutti i requisiti necessari per l'implementazione sono definiti nell'RFC di riferimento [3], in modo da garantire la correttezza e l'interoperabilità del protocollo.

4.1.2 Architettura

La progettazione dell'architettura del software per l'implementazione di Minimal IKE si basa su requisiti di alta efficienza e ottimizzazione, rendendo il linguaggio C la scelta più adeguata. Infatti permette un controllo diretto sulle risorse di sistema. Il linguaggio C impone una strutturazione ben definita per affrontare in modo efficace la complessità del codice. Pertanto, nel prosieguo, verrà illustrato come organizzare le directory e i moduli previsti per l'implementazione.

Directory

Le directory sono state organizzate in modo strategico per facilitare la gestione e l'accessibilità del codice. La struttura è riportata in *Figura 4.1*.

```
src
├── config
├── utils
├── ike
├── nodes
├── Makefile
└── config.json
```

FIGURA 4.1: Struttura delle directory del progetto

È stata creata una directory denominata config, dedicata al parsing del file di configurazione, permettendo così di rendere le informazioni disponibili globalmente agli altri moduli. All'interno della directory ike sono state raccolte tutte le caratteristiche intrinseche del protocollo, comprese le funzioni e i meccanismi fondamentali. È stata anche introdotta una directory utils, contenente utility come funzioni crittografiche e specifiche per la conversione tra formati in modo da poter essere trasmessi sulla rete.

Moduli

Ogni directory ha un file header associato, riportati in *Tabella 4.1*, il che consente una chiara separazione dei compiti e una migliore organizzazione del codice. Questa separazione facilita la gestione delle funzionalità specifiche, migliorando la leggibilità e manutenibilità del progetto.

File	Descrizione
`config.h`	Esegue il parsing del file di configurazione ed estrae le impostazioni definite dall'utente.
`ike.h`	Contiene la definizione di tutte quelle che sono le strutture e costanti associate al protocollo.
`crypto.h`	Funzioni crittografiche utilizzate nel protocollo, tra cui le funzioni per KE e generazione di Nonce
`network.h`	Funzioni per la gestione delle comunicazioni di rete, in particolare le primitive per fare comunicazione tramite socket.
`utils.h`	Funzioni di utilità condivise tra i vari moduli.

TABELLA 4.1: File header e descrizione delle loro funzionalità

4.2 Implementazione

Per lo sviluppo del progetto, sono stati selezionati diversi strumenti già pronti che ottimizzano e semplificano il processo di implementazione.

Poiché è necessario sviluppare un software in grado di far comunicare due macchine remote, è fondamentale implementare un meccanismo di IPC (Inter Process Communication). In conformità con le specifiche fornite dall'RFC, è necessario utilizzare i *socket*, in particolare quelli UDP. Questi strumenti sono forniti dalla libreria di sistema `sys/socket.h`. Esistono diverse tipologie di socket, per lavorare con quelli di rete occorre includere la libreria

`netinet/in.h`, che definisce le strutture e le costanti necessarie per la gestione. Un altro aspetto da considerare è che lo stack TCP/IP (ovvero la suite di protocolli utilizzati dalla rete) utilizza la rappresentazione delle informazioni in formato **big endian**, principalmente per motivi di retrocompatibilità, poiché il progetto ARPANET è stato sviluppato in questo modo. Inoltre, questa rappresentazione semplifica l'implementazione dei protocolli di rete: il byte più significativo (MSB) sulla base del quale, in generale, si prendono le decisioni è il primo ad essere letto.
In contrapposizione, le macchine con cui siamo abituati a lavorare *desktop* utilizzano principalmente l'architettura **little endian**, il che rende necessari meccanismi per convertire le informazioni generate dal calcolatore nel formato definito dalla rete. Un esempio di queste conversioni è fornito dalla libreria `arpa/inet`, che gestisce le trasformazioni tra le diverse rappresentazioni di dati necessarie per garantire una corretta comunicazione in rete.
Tuttavia, questa si occupa principalmente della rappresentazione di indirizzi e porte, il che non è sufficiente per tutte le informazioni che devono essere comunicate. Per affrontare questa limitazione e garantire la corretta gestione di qualsiasi tipo di dato, è necessario utilizzare la libreria `endian.h`.

Per quanto riguarda la definizione della configurazione un formato ampiamente diffuso e semplice da leggere è quello `json`. Si è dunque deciso di adottare questo formato e fare il parsing di questo tramite la libreria `libcjson`. Mentre per quanto riguarda le implementazioni delle primitive crittografiche si è utilizzata la lireria `libcrypto`, ampiamente utilizzata per la sicurezza delle comunicazioni.

Codice

Nel contesto della comunicazione di rete, un pacchetto è composto principalmente da due parti: *header* e *payload*. Questa suddivisione è fondamentale per garantire la corretta trasmissione delle informazioni tra le macchine.

- L'header contiene informazioni cruciali su come gestire il payload, possiamo vedere il suo contenuto come indicazioni per il corretto processamento del contenuto del pacchetto.
- Il payload rappresenta il contenuto effettivo del pacchetto, ovvero i dati che si desidera trasmettere. Questo può includere messaggi, file o qualsiasi tipo di informazione utile all'applicazione.

Header

L'header di un pacchetto IKE, ha la struttura riportata in *Figura* 4.2. A partire da questa definizione, si è proceduto a creare una rappresentazione in C utilizzando la struttura `ike_header_t`, che offre una modalità chiara e strutturata per gestire i dati.

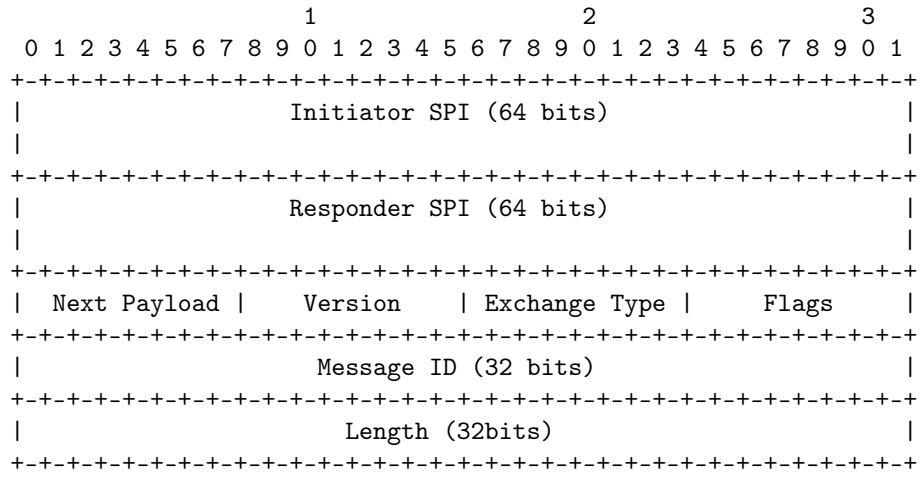

FIGURA 4.2: Formato IKE Header definito dall'RFC

```
typedef struct {
    uint64_t initiator_spi;
    uint64_t responder_spi;
    uint8_t next_payload;
    uint8_t version;
    uint8_t exchange_type;
    uint8_t flags;
    uint32_t message_id;
    uint32_t length;
} __attribute__((packed)) ike_header_t;
```

La struttura mostrata nel Codice 4.2 elenca i campi nell'ordine preciso stabilito dall'RFC. Questo ordinamento è cruciale per il processo di serializzazione, il quale consente di trasformare la struttura in una sequenza di byte per la trasmissione su rete. Qualsiasi disallineamento nei campi potrebbe generare interpretazioni errate dei dati da parte del destinatario, compromettere la comunicazione e minacciare la sicurezza.

In aggiunta, è necessario includere l'attributo `packed` nella definizione della struct. Questo attributo serve a prevenire l'aggiunta di padding tra i campi

della struttura, che è normalmente inserito dal compilatore per allineare i dati. L'aggiunta di padding può alterare le dimensioni complessive della struct, rendendola diversa da quella definita nell'RFC e causando potenziali problemi durante il processo di serializzazione e deserializzazione.

Payload

In IKE, ogni payload è preceduto da un *Generic Payload Header*, il cui compito è fornire informazioni essenziali sul payload successivo e sulla lunghezza di quello corrente. IKE prevede diverse tipologie di payload, ciascuna con un ruolo specifico, già definite nella sezione dedicata (fare riferimento alla sezione IKE). A scopo di spiegazione, consideriamo il caso in cui la tipologia del payload sia *Key Exchange*, il cui formato è mostrato in *Figura 4.3*.

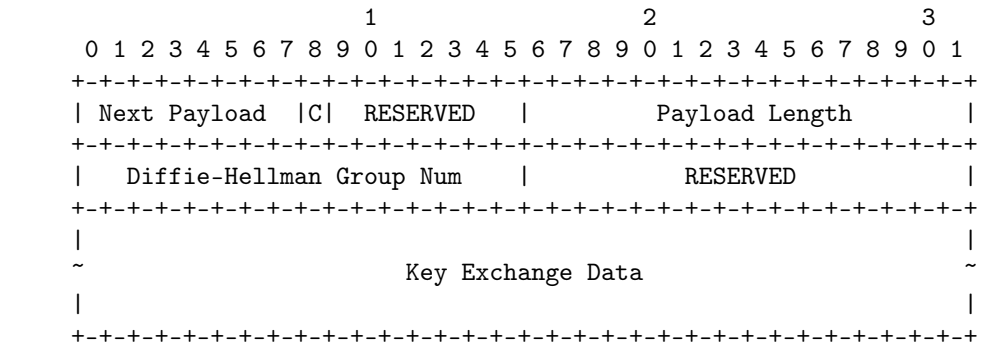

FIGURA 4.3: Formato KE Payload come specificato da RFC

```
typedef struct {
    uint8_t  next_payload;
    uint8_t  critical :1;
    uint8_t  reserved :7;
    uint16_t length;
} __attribute__((packed)) PayloadHeader;

typedef struct {
    uint16_t dh_group;
    uint16_t reserved;
    uint8_t ke_data[MAX_KEY_EXCHANGE_DATA_LEN];
} __attribute__((packed)) KeyExchange;
```

In Codice 4.2 e 4.2 abbiamo la rappresentazione tramite struct di un payload. Nel caso di `KeyExchange` l'informazione della lunghezza contenuta nel *Generic Payload Header* è fondamentale poichè, il materiale per il Key Exchange ha una

dimensione che varia in base all'algoritmo utilizzato. In *Figura 4.4* è riportata la rappresentazione dei collegamenti tra i payload nei pacchetti `INIT`.

Pacchetto

Ogni pacchetto in IKE è costituito da un header seguito da una serie di payload, la cui presenza è determinata dalla tipologia di scambio in corso. Il collegamento tra i diversi payload è garantito dal campo Next Payload presente nel Generic Payload Header; si può quindi concepire questa struttura come una lista singolarmente puntata. Tuttavia questa caratteristica può essere utilizzata solo in fase di ricezione del pacchetto.

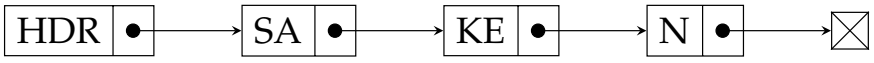

FIGURA 4.4: Pacchetto `INIT` come lista singolarmente puntata

```
typedef struct {
    PayloadHeader *header;
    void *data;
    size_t data_len;
    int type;
} PacketPayload;
```

Nel caso di creazione del pacchetto occore agire in un altro modo. Definiamo la struct `PacketPayload` come il tipo generico di payload. Conoscendo a priori l'ordine dei payload definiamo un array che riflette questo ordine. Questo consente una costruzione incrementale del pacchetto, tramite la serializzazione degli elementi. Occorre procedere in questo modo poiché è necessario conoscere le lunghezze dei singoli payload per calcolare il campo length nell'header, che deve contenere la dimensione complessiva del pacchetto.
Mentre nel caso in cui si esegue il parsing del pacchetto, quindi in ricezione della risposta, si procede nell'attraversamento del pacchetto e nel mentre viene popolata una lookup table.

```
typedef struct LookupEntry {
    uint8_t type;              // Tipo di payload
    uint8_t* data;             // Dati del payload
    uint16_t length;           // Lunghezza del payload
    struct LookupEntry* next; // Puntatore al prossimo elemento
} LookupEntry;
```

```
void add_to_lookup(LookupEntry** head, uint8_t type, uint8_t* data,
    uint16_t length) {
    LookupEntry* new_entry = create_lookup_entry(type, data, length);
    new_entry->next = *head;
    *head = new_entry;
}

void parse_payload(uint8_t* buffer, size_t buffer_len, NextPayload
    current,  LookupEntry** lookup_table){
    if (current == 0) {
        free(buffer);
        return;
    }
    uint8_t next_payload = buffer[0];
    uint16_t length = ntohs(*(uint16_t*)&buffer[2]);
    size_t payload_len = length - GENERIC_PAYLOAD_HEADER_DIM;
    uint8_t* payload_data = malloc(payload_len);
    memcpy(payload_data, &buffer[GENERIC_PAYLOAD_HEADER_DIM],
    payload_len);
    add_to_lookup(lookup_table, current, payload_data, payload_len);
    free(payload_data);

    if (next_payload != 0) {
        size_t next_buffer_size = buffer_len - length;
        parse_payload(buffer +length, next_buffer_size,next_payload,
    lookup_table);
    }
}
```

La funzione `parse_payload` può essere concepita come un metodo per attraversare il pacchetto come se fosse una lista, sfruttando il campo `next_payload` come puntatore all'elemento successivo. Ad ogni passaggio, la funzione popola una lookup table, in cui le coppie chiave-valore sono tipo di payload e il payload di quel tipo. Queste sono impiegate per ottimizzare operazioni di ricerca e recupero, consentendo di ottenere risultati in modo più efficiente.

Socket

Dopo aver esaminato il processo di generazione e ricostruzione dei pacchetti, è importante affrontare la questione di come questi vengano inviati e ricevuti. Questo processo è reso possibile grazie all'utilizzo dei socket, che in un

ambiente C, richiede le seguenti operazioni:

- Utilizzare la funzione `socket()` per creare un socket e ottenere il suo file descriptor. Il parametro `AF_INET` specifica che il socket opererà con indirizzi IPv4. Esistono altri *domini di comunicazione*, come `AF_UNIX`, `AF_CAN`, e `AF_INET6`. Il parametro `SOCK_DGRAM` indica che il socket utilizzerà una comunicazione a datagrammi (senza connessione).
- Attraverso la struttura `sockaddr_in` si definiscono le caratteristiche dell'indirizzo IP e della porta per la comunicazione di rete, già in formato big-endian. Successivamente, il socket creato deve essere associato a questo indirizzo tramite la funzione `binding`.

Alla fine di questo procedimento il programma è in grado di comunicare con l'esterno tramite una coppia `IP:Porta`. Nel caso dell'initiator questa coppia è specificata nel file di configurazione.

```
typedef struct {
    int socketfd;
    uint64_t spi;
    struct sockaddr_in node;
} Initiator;

int initiator_init(Initiator *initiator, const Node *conf){
    initiator->socketfd = socket(AF_INET, SOCK_DGRAM, 0);
    memset(&initiator->node, 0, sizeof(initiator->node));
    initiator->node.sin_family = AF_INET;
    initiator->node.sin_port = htons(conf->port);
    inet_pton(AF_INET, conf->ip, &initiator->node.sin_addr);
    if (bind(
        initiator->socketfd,
        (struct sockaddr *)&initiator->node, sizeof(initiator->node)
        ) < 0)
    {
        perror("bind");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

Il codice mostra la rappresentazione dell'initiator mediante e struct e la funzione che si occupa di specificare quelli che sono parametri che gli consentono di

comunicare. Una definizione analoga l'abbiamo per il responder che tuttavia non necessita di un socket. Ora andiamo a vedere come avviene lo scambio di messaggi.
La funzione `sendto()` consente l'invio di dati tramite un socket non connesso. Per farlo, richiede diversi parametri: il file descriptor del socket, un buffer che contiene i dati da inviare, la lunghezza di questi dati, e le informazioni relative all'indirizzo del destinatario, contenute all'interno della struct `sockaddr_in`.

```
sendto(
    initiator.socketfd,
    packet,
    packet_len,
    0,
    (struct sockaddr*)&responder.node,
    sizeof(responder.node)
);
recvfrom(
    initiator.socketfd,
    response,
    MAX_RESPONSE_DIM,
    0,
    (struct sockaddr *)&response,
    &len
);
```

La funzione `recvfrom()` consente di ricevere dati da un socket non connesso, come nel caso di socket UDP. Per utilizzarla, è necessario fornire diversi parametri: il file descriptor del socket, un buffer in cui memorizzare i dati ricevuti, la lunghezza del buffer, flag per opzioni aggiuntive (che possono essere impostati a zero per il comportamento predefinito), un puntatore a una struttura per memorizzare le informazioni sull'indirizzo del mittente, e un puntero alla variabile che contiene la dimensione della struttura dell'indirizzo. Andando a confrontare l'indirizzo del mittente ricevuto dal pacchetto possiamo andarlo a confrontarlo con quello del responder per essere sicuri che stiamo veramente parlando con lui.

Endianness

Nel campo della programmazione e della comunicazione tra sistemi, un aspetto fondamentale da considerare è l'endianness, che si riferisce al modo

in cui i dati vengono memorizzati in memoria. Esistono principalmente due formati di endianness: big endian e little endian.

- Nel formato big endian, il byte più significativo (Most Significant Byte, MSB) viene memorizzato all'indirizzo di memoria più basso. Ciò significa che i dati vengono letti partendo dal valore più alto.
- Al contrario, nel formato little endian, il byte meno significativo (Least Significant Byte, LSB) occupa l'indirizzo di memoria più basso.

Questa differenza di rappresentazione, mostrata in *Figura 4.5*, può causare problemi di interoperabilità quando sistemi diversi tentano di comunicare tra loro.

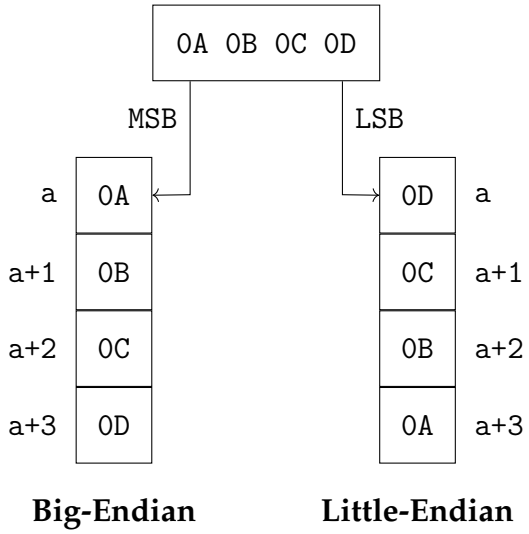

FIGURA 4.5: Rappresentazione grafica del problema della endianness

Per affrontare questi problemi di compatibilità e garantire che i dati vengano trasmessi e ricevuti correttamente, abbiamo implementato le seguente soluzione. Per garantire che il processo di conversione possa essere applicato a tutte le strutture, comprese eventuali aggiunte future, la funzione accetta come parametri un tipo generico, che rappresenta i byte da convertire, e un parametro che specifica il tipo della struttura da elaborare. Questo approccio consente di descrivere i campi della struttura da convertire, assicurando che i campi con dimensioni di 16, 32 e 64 bit, che sono suscettibili al problema della endianness, vengano gestiti correttamente. La descrizione di questi campi è possibile grazie all'array `field_descriptor_t`, che contiene per ogni elemento l'offset del campo da convertire e la sua lunghezza in bit. La funzione che gestisce la conversione cicla su questo array, applicando la funzione di conversione appropriata al valore dereferenziato in base alla sua lunghezza.

```
void convert_to_big_endian(void *data, PayloadType type) {
    field_descriptor_t *fields = NULL;
    size_t fields_num = 0;
    switch(type){
        case IKE_HEADER: {
            field_descriptor_t tmp[] = {
            { offsetof(ike_header_t, initiator_spi), FIELD_UINT64 },
            { offsetof(ike_header_t, responder_spi), FIELD_UINT64 },
            { offsetof(ike_header_t, message_id), FIELD_UINT32 },
            { offsetof(ike_header_t, length), FIELD_UINT32 }
            };
            fields = (field_descriptor_t *) tmp;
            fields_num = sizeof(tmp) / sizeof(tmp[0]);
            break;
        };
        ...
    }
    for (size_t i = 0; i < fields_num; i++) {
        void *field_ptr = (uint8_t *)data + fields[i].offset;
        switch (fields[i].type) {
            case FIELD_UINT16: {
                uint16_t *value = (uint16_t *)field_ptr;
                *value = htobe16(*value);
                break;
            }
            case FIELD_UINT32: {
                uint32_t *value = (uint32_t *)field_ptr;
                *value = htobe32(*value);
                break;
            }
            case FIELD_UINT64: {
                uint64_t *value = (uint64_t *)field_ptr;
                *value = htobe64(*value);
                break;
}}}}
```

Chiave Pubblica

Il codice riportato consente all'initiator di generare la propria chiave pubblica, da utilizzare durante lo scambio `INIT`. Questa operazione viene eseguita

tramite la libreria OpenSSL, che offre tipi di dati specifici e meccanismi volti a garantire che la generazione delle chiavi avvenga in modo *constant-time*, riducendo così il rischio di attacchi basati su canali laterali (*side-channel attacks*). Dopo aver generato la chiave privata, viene derivata la corrispondente chiave pubblica, che successivamente viene serializzata in un formato byte per essere trasmessa al peer remoto durante lo scambio di messaggi.

```
OSSL_PARAM params[2];
EVP_PKEY *pkey_local = NULL;
EVP_PKEY_CTX *pctx = EVP_PKEY_CTX_new_from_name(NULL, "DH", NULL);

params[0] = OSSL_PARAM_construct_utf8_string("group", "modp_2048", 0);
params[1] = OSSL_PARAM_construct_end();

EVP_PKEY_keygen_init(pctx);
EVP_PKEY_CTX_set_params(pctx, params);
EVP_PKEY_generate(pctx, &pkey_local);
BIGNUM *pub_key = NULL, *priv_key = NULL;

EVP_PKEY_get_bn_param(pkey_local, OSSL_PKEY_PARAM_PUB_KEY, &pub_key);
EVP_PKEY_get_bn_param(pkey_local, OSSL_PKEY_PARAM_PRIV_KEY, &priv_key);
size_t ke_data = BN_num_bytes(pub_key);
BN_bn2bin(pub_key, ke.ke_data);
```

Capitolo 5

Analisi e Sviluppi

In questo capitolo viene esaminata l'implementazione e riportati i miglioramenti necessari a garantire la piena funzionalità del sistema e, di conseguenza, la sua capacità di operare in ambienti reali. Inoltre, verrà presentato un rapido resoconto dei risultati ottenuti finora, evidenziando i principali vantaggi in termini di efficienza e sicurezza raggiunti.

Analisi

L'RFC relativo al protocollo trattato è un documento estremamente complesso e tecnico, e considerando che si tratta di un protocollo di sicurezza, le difficoltà aumentano ulteriormente. Nella sezione precedente sono stati presentati alcuni estratti del codice. Tuttavia, ci sono molte altre funzionalità più complesse che non sono state esplorate in dettaglio, come la gestione del buffer per la creazione dei pacchetti, il parsing della configurazione, che attualmente supporta solo le impostazioni essenziali e l'autenticazione mediante certificati *X.509*. Inoltre, non sono state trattate altre pratiche di sicurezza avanzate che sarebbe opportuno includere in un'implementazione completa. Nonostante l'ampia gamma di funzionalità già implementate, il lavoro non può essere considerato completo. Infatti allo stato attuale l'implementazione considera solamente i primi due scambi, ovvero quelli che consentono di instaurare il canale sicuro: la IKE SA. Tuttavia manca la parte che riguarda l'utilizzo di questo canale per instaurare una Child Security Associations (`Child_SA`). La mancanza di gestione delle `Child_SA` rappresenta una limitazione significativa, poiché è proprio in questa fase che avviene la configurazione finale per la trasmissione dei dati protetti attraverso IPsec. Un ulteriore aspetto riguarda l'interfacciamento con il kernel del sistema operativo. Per far sì che le regole di sicurezza negoziate da IKE siano effettivamente applicate a livello di sistema, è necessario che queste vengano trasferite al kernel, che si occupa di gestire la rete e il traffico IP. In particolare, il kernel utilizza queste regole per applicare

le primitive crittografiche, garantendo le proprietà di sicurezza dei pacchetti IP in transito. Senza questa integrazione, le SA rimangono confinate all'interno dell'implementazione utente del protocollo IKE, senza poter influenzare il comportamento del traffico di rete reale. Il kernel deve essere informato delle SA attraverso meccanismi specifici, come le interfacce di rete o specifici moduli di sistema (ad esempio, `XFRM` nei sistemi Linux, che gestisce le policy di sicurezza IPsec). Solo una volta che queste SA vengono "inserite" nel kernel, il sistema operativo può applicare le politiche di sicurezza alle connessioni di rete, garantendo che le comunicazioni siano protette in conformità con quanto negoziato.

Durante la progettazione, sono state prese in considerazione numerose problematiche di sicurezza, come la protezione da attacchi di replay, l'ottimizzazione della gestione della memoria e l'uso di moduli per delegare compiti specifici. Tuttavia, molte criticità emergono solo quando il software interagisce con l'ambiente reale e viene utilizzato in contesti operativi quotidiani.

Risultati

Per testare l'interazione del software con sistemi esterni e valutarne il corretto funzionamento, è stato configurato un ambiente di test in cui l'implementazione sviluppata è stata messa in comunicazione con un'istanza di strongSwan in ascolto su un socket diverso. strongSwan è una delle implementazioni più diffuse di IKEv2 e IPsec, e rappresenta uno standard di riferimento nell'ambito della sicurezza delle reti. L'obiettivo di questo test era duplice:

- *Verifica della compatibilità*: poiché strongSwan segue rigidamente le specifiche IKEv2 [2], far comunicare la nostra implementazione con strongSwan è un metodo per confermare che il protocollo è stato implementato correttamente, almeno per quanto riguarda la negoziazione delle Security Associations (SA) iniziali.
- *Monitoraggio delle prestazioni*: oltre alla compatibilità, si voleva osservare l'impatto della comunicazione su un ambiente locale in termini di tempi di negoziazione, latenza e utilizzo delle risorse. Queste informazioni risultano cruciali per valutare l'efficienza e la scalabilità dell'implementazione.

Le interazioni hanno confermato la corretta comunicazione della IKE SA tra l'implementazione custom e strongSwan, dimostrando che i messaggi di inizializzazione e risposta erano conformi allo standard IKEv2.

Il principale risultato introdotto da questa implementazione è che consente di applicare sicurezza delle comunicazioni tramite IPsec in contesti in cui non era possibile: ovvero in cui si hanno a disposizione risorse limitate. Il cambio di paradigma elimina la necessità di utilizzare software come strongSwan, ricco di funzionalità non necessarie nel contesto specifico. Grazie a questa semplificazione, si ottiene un notevole miglioramento sia nell'occupazione di memoria sia nel carico sulla CPU. Invece di un carico costante, il software adotta un modello a interrupt, attivandosi solo quando necessario.
Come risultato dell'analisi, possiamo osservare che, a parità di configurazione, la versione minimale di IKE risparmia il **20%** della dimensione del pacchetto rispetto a implementazioni più complesse come quella di strongSwan. I tempi di elaborazione risultano paragonabili. È importante notare che un confronto diretto dell'implementazione fornita con strongSwan non è pienamente applicabile. Nonostante ciò, l'implementazione minimale richiede solo $9,820KB$ di memoria, un valore significativamente ridotto, che la rende particolarmente adatta per ambienti con risorse limitate.

	Strongswan	**Implementazione**	**Confronto**
Pacchetto `INIT`	$518KByte$	$418KByte$	-20%
Memoria occupata	$18MByte$	$9MByte$	-50%
Tempi per l'`INIT`	$3.09ms$	$3.04ms$	$\approx$
Ruoli	*Initiator*	*Initiator*	$=$
	Responder	-	-

TABELLA 5.1: Confronto tra alcuni parametri dell'esecuzione di StrongSwan e quelli dell'implementazione fornita

Sviluppi

Dall'analisi effettuata, emerge che l'implementazione attuale, pur avendo raggiunto un livello funzionale, necessita di ulteriori sviluppi per essere completa e pronta all'uso. Questi sviluppi sono fondamentali per migliorare la compatibilità, la sicurezza e la robustezza del sistema. Quelli su cui si concentreranno principalmente gli sforzi sono: gestione delle **Child Security Associations** e quella fondamentale di **interfacciamento con il kernel**. Queste sono essenziali per rendere operative le SA (Security Associations) negoziate tramite IKEv2. Infatti su sistemi Linux è il kernel a gestire le politiche di sicurezza e le chiavi IPsec attraverso una suite di funzionalità conosciuta come `XFRM` (transformations framework), che viene utilizzata per trasformare e

proteggere i pacchetti di rete. La comunicazione tra l'implementazione IKEv2 e il kernel avviene tipicamente tramite l'uso del Netlink, un protocollo per lo scambio di informazioni tra il kernel e lo spazio utente.

Un altro aspetto su cui lavorare è quello che rigurda la verifica della robustezza e della sicurezza dell'implementazione. Questa va protetta da vari tipi di attacchi, come attacchi di replay, attacchi di man-in-the-middle e tentativi di sfruttare vulnerabilità nei messaggi IKE per alterare lo stato della sessione. Un ulteriore problema è la gestione sicura della memoria, soprattutto per evitare buffer overflow e altre vulnerabilità correlate alla gestione dinamica della memoria. Per proteggere l'implementazione da questi attacchi risulta fondamentale l'utilizzo di test. Le principali tipologie di test da svolgere sono:

- **Fuzzing**: per testare la robustezza del parsing dei pacchetti, il software dovrà essere sottoposto a input casuali per verificare come gestisce messaggi malformati o corrotti.
- **Analisi statica e dinamica**: strumenti come Valgrind o AddressSanitizer saranno utilizzati per rilevare eventuali problemi nella gestione della memoria e nell'accesso a risorse critiche.
- **Penetration testing**: simulando attacchi reali, è possibile individuare punti deboli o lacune nel software che potrebbero essere sfruttate da attaccanti per compromettere la sicurezza delle comunicazioni.

Capitolo 6

Conclusioni

In questo lavoro di tesi è stato studiato ed analizzato il funzionamento di IKEv2 con sicurezza quantum. Il funzionamento del protocollo è stato verificato nel contesto delle comunicazioni satellitari, andando a verificare che nonostante l'utilizzo di primitive post-quantum, i tempi di esecuzione sono nettamente inferiori rispetto ai tempi di latenza dovuti alle comunicazioni dei satelliti. L'ambiente di testing sviluppato, grazie a un'attenta scelta delle tecnologie utilizzate, ha garantito isolamento, portabilità e automazione, dimostrando la fattibilità dell'uso di IKEv2 negli scenari previsti dal progetto ESA QUALITA. In particolare, ha evidenziato che, in linea teorica, è possibile proteggere le comunicazioni satellitari utilizzando primitive crittografiche post-quantum, e a permesso di identificare quali tra gli algoritmi attualmente standardizzati risultano idonei per questo tipo di applicazione.

L'analisi approfondita di strongSwan, l'implementazione di IKEv2 considerata, dato che rappresenta uno standard per le comunicazioni tramite IPsec, ha permesso di individuarne i punti di forza e le debolezze. Pur essendo una soluzione completa, adatta alla maggior parte degli scenari, risulta eccessivamente sovradimensionata per ambienti che necessitano solo di un client VPN minimale. Per esempio il caso satellitare, in cui a causa della limitatezza delle risorse, applicare soluzioni del genere non è ottimale. A partire da questa riflessione, sono state condotte ulteriori ricerche che hanno portato all'identificazione dell'RFC di Minimal IKE. Questo standard propone un approccio semplificato per la gestione dell'autenticazione e della sicurezza delle comunicazioni, riducendo le funzionalità rispetto alle implementazioni tradizionali come strongSwan, ma risultando ideale per scenari considerati. Tuttavia, è stato riscontrato che non esistono implementazioni concrete di Minimal IKE, se non una proof of concept in Python, il che ha portato allo sviluppo di una propria implementazione minimale, incentrata sull'efficienza e l'uso ridotto delle risorse.

L'obiettivo è stato quindi quello di creare un client VPN leggero, capace di interagire con un'istanza server più potente, che gestisse centralmente tutte le connessioni in un modello client-server, segnando un cambio di prospettiva. Lo sviluppo si è concentrato sull'implementazione delle funzionalità necessarie, con un'attenzione particolare all'efficienza sia computazionale che nell'uso della memoria. Ciò ha richiesto un approfondimento di tecniche complesse a basso livello, come la gestione del pool di buffer, la creazione di lookup table e la manipolazione di byte.

Tuttavia, considerata la complessità dell'argomento, questa implementazione rappresenta una base solida da cui partire per ulteriori sviluppi, poiché è stato possibile ridurre del 20%, a parità di configurazione, la dimensione dei pacchetti e di ottenere un'impronta in memoria notevolmente ridotta, mantenendo tempi di elaborazione paragonabili a quelli delle soluzioni esistenti. Queste caratteristiche la rendono un buon punto di partenza per approfondimenti futuri. Essa potrà essere utile non solo nei contesti dei progetti che si stanno portando avanti in collaborazione con l'ESA, ma anche nell'ambito delle reti IoT, offrendo la possibilità di garantire comunicazioni più sicure in scenari dove, in passato, ciò era impensabile a causa del sovradimensionamento delle soluzioni in uso e delle risorse limitate. Questo lavoro potrebbe così costituire un'ottima base per un progetto di dottorato.

Appendice A

Approfondimento Tecnico

A.1 IKE Notation

A.1.1 Authentication

L'autenticazione dei peer avviene effettuando il sign (o calcolando il MAC) di un payload che dipende dagli scambi precedenti. In particolare questo payload è cmposto da un ottetto che viene autenticato in base alla modalità di aunteticazione scelta:

- Nel caso di *PubKey* questo viene firmato con la chiave privata del peer e ne viene allegato il certificato di chiave pubblica .
- Nel caso di *PSK* l'AUTH payload viene generato a partire dalla chiave condivisa a cui viene aggiunge della non predicibilità tramite del padding e una prf.

A.1.2 IKE SA

Le chiavi in una IKE SA vengono derivate a partire dagli attributi dei differenti degli scambi. In particolare al termine del primo scambio viene calcolato il:

$$SKEYSEED = PRF(N_i|N_r, g^{ir})$$

A partire da questo seeder vengono generati i prametri di sicurezza da utililizzare per la IKE SA, questi sono derivati nel seguente modo:

$$\{SK_d|SK_{ai}|SK_{ar}|SK_{ei}|SK_{er}|SK_{pi}|SK_{pr}\} = PRF + (SKEYSEED, N_i|N_r, SPI_i, SPI_r)$$

Chiave	Descrizione
SK_d	Utilizzata per generare il keymaterial per le CHILD_SA
SK_a	Chiavi per autenticare gli scambi successivi, una per direzione
SK_e	Chiavi per cifrare gli scambi successivi, una per direzione
SK_p	Chiavi utilizzata per generare l'AUTH Payload, una per direzione

TABELLA A.1: Chiavi e loro utilizzo

A.1.3 IPsec SA

Nel caso di una SA questa può essere generata automaticamente dopo l'auth oppure attraverso l'apposito scambio di questo tipo il keymaterial a partire dal quale vengono derivati i parametri di sicurezza è ottenuto nel seguente modo:

$$KEYMAT = prf + (SK_d, N_i|N_r)$$

A.2 Docker

Il dockerfile in cui andiamo a containerizzare il servizio è quello in, riferimento al file. A partire da un'immagine ubuntu andiamo ad installare tutte le utility necessarie, dopodichè si scarica il sorgenti di oqs e si compila con i parametri specificati. Si fa la stessa cosa per strongswan e in fase di confiugrazione si specificano i parametri. Infine si fa un clean-up del sistema dalle dipendenze necessarie solo per la compilazione.

```
FROM ubuntu:22.04
RUN DEV_PACKAGES="wget␣unzip␣bzip2␣make␣gcc␣libssl-dev␣cmake␣
    ninja-build"
RUN apt-get update
    && apt-get install -y iproute2 iputils-ping nano
       $DEV_PACKAGES
    && mkdir /liboqs && cd /liboqs
    && wget linksourcecodeliboqs\$LIBOQS\_VERSION.zip
    && unzip $LIBOQS_VERSION.zip && cd liboqs-$LIBOQS_VERSION
    && mkdir build && cd build &&
    && cmake -GNinja -DOQS_USE_OPENSSL=ON \
        -DBUILD_SHARED_LIBS=ON \
        -DCMAKE_INSTALL_PREFIX=/usr \
        -DCMAKE_BUILD_TYPE=Release \
        -DOQS_BUILD_ONLY_LIB=ON ..
    && ninja && ninja install
    && cd / && rm -R /liboqs
    && mkdir /strongswan-build && cd /strongswan-build
    && wget linksourcecode/strongswan-$VERSION.tar.bz2
    && tar xfj strongswan-$VERSION.tar.bz2 && cd strongswan-
       $VERSION
    && ./configure --prefix=/usr
        --sysconfdir=/etc
        --disable-ikev1
        --enable-frodo
        --enable-oqs
        --enable-silent-rules
    && make all && make install
    && cd / && rm -R strongswan-build
    && ln -s/usr/libexec/ipsec/charon charon
    && apt-get -y remove \$DEV\_PACKAGES
    && apt-get -y autoremove && apt-get clean
    && rm -rf /var/lib/apt/lists/*
EXPOSE 500 4500
ENTRYPOINT ["./charon" ]
```

Le opzioni `cap_add` sono utilizzate per aggiungere capacità specifiche ai container Docker, consentendo loro di eseguire operazioni che normalmente richiederebbero privilegi di root. Queelli che servono per il nostr setup sono:

- `NET_ADMIN` consente al processo all'interno del container di eseguire operazioni di amministrazione della rete. Fondamentale per configurare le interfacce di rete e gestire il routing e i firewall, operazioni chiave per VPN e IPsec.
- `SYS_MODULE` consente di caricare e scaricare moduli del kernel. Fondamentale per caricare moduli del kernel per supportare funzionalità IPsec che non sono già caricate.
- `SYS_ADMIN` è una delle capacità più potenti e può consentire una vasta gamma di operazioni di amministrazione del sistema.

```
services:
    moon:
        build: ./
        container_name: moon
        cap_add:
            - NET_ADMIN
            - SYS_ADMIN
            - SYS_MODULE
        volumes:
            - ./moon:/etc/swanctl
            - ./strongswan.conf:/etc/strongswan.conf

    carol:
        build: ./
        container_name: carol
        depends_on:
            - moon
        cap_add:
            - NET_ADMIN
            - SYS_ADMIN
            - SYS_MODULE
        volumes:
            - ./carol:/etc/swanctl
            - ./strongswan.conf:/etc/strongswan.conf
```

A.3 Certificati

Strongswan mette a disposizione un'utility `pki` per la gestione della public key infrastrucutre. Tramite questa utility sono stati generati i certificati necessari per la fase di mutua autenticazione tra i due peer. Il confronto tra le primitive di firma digitale utilizzate nelle configurazioni sono riportate in *Tabelle A.2*.

Schema	Chiave private	Certificato
ECDSA	227	530
falcon512	3.0k	2.4k
falcon1024	5.6k	4.5k
dilithium2	5.3k	5.4k
dilithium3	8.2k	7.6k
dilithium5	10k	10k

TABELLA A.2: Confronto dimensioni Schemi di Firma

Di seguito vengono riportati i comandi per generare inizialmente le chiavi, utilizzate successivamente per creare i certificati di chiave pubblica, partendo dalla Root CA. Successivamente, queste chiavi pubbliche vengono firmate e utilizzate per ciascun peer nel processo di autenticazione.

```
pki --gen --type scheme --outform pem > "CA.type.pem"
pki --self --type priv --in "CA.type.pem"
    --ca
    --lifetime 3652
    --dn "CN=Root CA"
    --outform pem > "CA.type.cert.pem"

pki --issue
    --cacert "CA.type.cert.pem"
    --cakey "CA.type.pem "
    --type priv --in "Peer.pubkey.pem "
    --lifetime 1461
    --dn "CN=Peer"
    --outform pem > "Peer.type.cert.pem"
```

Bibliografia

[1] Peter W Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[2] C. Kaufman et al. *RFC 7296: Internet Key Exchange Protocol Version 2 (IKEv2)*. USA, 2014.

[3] T. Kivinen. *RFC 7815: Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation*. USA, 2016.

[4] Seyoon Ragavan e Vinod Vaikuntanathan. "Space-efficient and noise-robust quantum factoring". In: *Annual International Cryptology Conference*. Springer. 2024, pp. 107–140.

[5] Gorjan Alagic et al. "Status report on the third round of the NIST post-quantum cryptography standardization process". In: (2022).

[6] V. Smyslov. *RFC 7383: Internet Key Exchange Protocol Version 2 (IKEv2) Message Fragmentation*. USA, 2014.

[7] CJ. Tjhai et al. *RFC 9370: Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2)*. USA, 2023.