Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

UNIVERSITÀ DEGLI STUDI DI PADOVA

Corso di Laurea Triennale in Ingegneria Informatica

Analisi di algoritmi di firma digitale post-quantum

Relatore

Prof. Luca Boldrin

Laureando

Tomas Lovato

Anno Accademico 2023 - 2024

Data di Laurea

27 Settembre 2024

Abstract

Questa tesi si propone di analizzare e confrontare alcuni algoritmi crittografici resistenti agli attacchi dei computer quantistici, noti come algoritmi post-quantum, recentemente valutati dal NIST (National Institute of Standards and Technology). L'obiettivo principale è valutare le performance di alcune implementazioni esistenti di algoritmi di firma digitale e di generazione delle chiavi in termini di sicurezza, efficienza e scalabilità. La tesi include una revisione della letteratura sugli algoritmi post-quantum, un'analisi comparativa delle soluzioni proposte, e una valutazione delle loro potenziali applicazioni in scenari reali. Questo lavoro mira a contribuire alla comprensione e alla valutazione delle tecnologie post-quantum emergenti, offrendo una base per future implementazioni sicure nell'ambito della protezione delle identità digitali.

This thesis aims to analyze and compare certain cryptographic algorithms resistant to attacks by quantum computers, known as post-quantum algorithms, recently evaluated by NIST (National Institute of Standards and Technology). The primary objective is to assess the performance of some existing implementations of digital signature algorithms and key generation in terms of security, efficiency, and scalability. The thesis includes a literature review on post-quantum algorithms, a comparative analysis of the proposed solutions, and an evaluation of their potential applications in real-world scenarios. This work aims to contribute to the understanding and assessment of emerging post-quantum technologies, providing a basis for future secure implementations in the field of digital identity protection.

Ringraziamenti

Desidero innanzitutto ringraziare il Professor Luca Boldrin per la sua preziosa guida nella stesura di questa tesi. La sua competenza e disponibilità mi hanno permesso di esplorare un settore affascinante e di raggiungere risultati che non avrei mai immaginato. Il suo suggerimento di approfondire gli algoritmi post-quantum mi ha aperto a nuove interessanti prospettive.

Un ringraziamento speciale va alla mia famiglia, che mi ha sempre sostenuto e incoraggiato a perseguire i miei obiettivi. La loro fiducia incondizionata nei miei confronti e nelle mie scelte si è trasformata in una forza indispensabile, soprattutto nei momenti in cui mi sentivo sopraffatto dagli esami o dalle mille difficoltà della vita. Un caloroso grazie a Roberta, Claudio e Sara.

Come non ringraziare, poi, tutte le persone che da tanti anni condividono con me gioie e difficoltà, accompagnandomi lungo i vari percorsi della vita. Ogni momento trascorso insieme è un tesoro inestimabile. Un grazie di cuore ad Alberto, Francesco, Matteo e Mattia. Spesso, tra un'attività e l'altra, mi capita di fermarmi e perdermi nei ricordi dei momenti più divertenti che abbiamo vissuto insieme... non considero questi istanti una distrazione, ma piuttosto un simbolo del prezioso legame che ci unisce.

In questa pagina non può mancare un *team* che si è rivelato importante nella mia vita: ispiro.tech. Da più di due anni faccio parte di questa realtà, resa speciale e stimolante dalle persone che ne fanno parte: Davide, Simone e Mattia.

Infine, vorrei ringraziare tutte le persone che ho avuto il piacere di conoscere più a fondo solo di recente. In particolare, un grande grazie a Tommaso, Leonardo, Hermann, Giulia, Greta, Filippo ed Enrico. Spero di poter approfondire il nostro rapporto, così come con tutti coloro che, per ragioni di spazio, non ho potuto menzionare.

La macchina può calcolare, ma è il cuore umano che dà valore ai numeri.

Indice

Elenco delle figure

Elenco delle tabelle

1

Introduzione

Da oltre due decenni la ricerca e sviluppo nel settore del Quantum Computing e, più in generale, della Quantum Technology, sta ottenendo risultati significativi che presto permetteranno a tali sistemi di interagire con la vita quotidiana. L'introduzione di principi della meccanica quantistica, come la *Quantum Superposition* (la teoria secondo cui le particelle subatomiche possono esistere in più stati contemporaneamente) e il *Quantum Entanglement* (il fenomeno che lega e fa interagire un gruppo di particelle anche a grandi distanze), promette di rivoluzionare il settore informatico. Tuttavia, queste innovazioni presenteranno nuove sfide per la sicurezza delle informazioni.

In particolare, i computer quantistici potranno sfruttare tecniche avanzate per compromettere i sistemi di crittografia e firma digitale attualmente utilizzati. Questo non è un problema che riguarda solo gli utenti più *specializzati* (che fanno uso di queste tecnologie per fini lavorativi e commerciali), ma interessa qualsiasi persona che utilizzi sistemi digitali. Molte tecnologie, tra cui il web, si basano sul protocollo TLS (Transport Layer Security). Ogni volta che accediamo a risorse del World Wide Web utilizzando il metodo HTTPS o inviamo una email, ci affidiamo a meccanismi integrati nel protocollo TLS che, mediante l'uso di chiavi simmetriche e asimmetriche, garantiscono la confidenzialità, integrità e autenticità dei contenuti. Queste caratteristiche sono essenziali: assicurano che la sorgente delle informazioni ricevute sia chi dichiara di essere (Autenticità), che i dati non vengano modificati durante il trasporto (Integrità) e che rimangano privati lungo tutto il percorso (Confidenzialità).

In una sorta di *effetto a cascata*, un attacco riuscito da parte di un computer quantistico potrebbe avere conseguenze devastanti su larga scala: ad esempio, gli aggiornamenti dei sistemi operativi, che prima dell'installazione automatica vengono verificati tramite meccanismi simili, potrebbero essere compromessi consentendo la diffusione di patch modificate con l'obiettivo di introdurre vulnerabilità nei sistemi operativi più diffusi.

Questi scenari sono preoccupanti se continueremo ad utilizzare i sistemi crittografici

attuali fino all'effettiva introduzione dei computer quantistici nel mercato. Per questo motivo, molti enti si stanno mobilitando per sostenere e favorire la ricerca nel campo della Post-Quantum Cryptography, con l'obiettivo di raggiungere uno standard e una transizione prima che tali minacce diventino realtà.

1.1 Tecnologie di firma digitale: il presente

Gli attuali sistemi di firma digitale più utilizzati si basano su AdES (Advanced Electronic Signature), una tipologia di firma elettronica progettata per offrire un elevato livello di sicurezza contro le tecnologie di attacco attualmente esistenti, dette anche *legacy*.

Questi sistemi sono ideali per dimostrare l'integrità della comunicazione firmata e l'unicità del firmatario, difatti AdES è uno standard a livello Europeo: la firma digitale prodotta con questi mezzi ha valore legale, cioè ha valore probatorio e difficilmente può essere ripudiata senza prove concrete [1].

Il processo di firma e verifica si basa su sistemi a chiave asimmetrica: il firmatario utilizza la propria chiave privata (*secret key*) per firmare un documento, e i destinatari possono verificare la validità della firma utilizzando la chiave pubblica (*public key*) del mittente [2]. Esistono diverse tipologie di firma AdES, tra cui:

1. AdES-BES (Basic Electronic Signature): è la forma più semplice, l'output del processo di firma del documento è composta dal documento stesso, la firma del documento (ottenuto tramite procedure di *hashing* e *padding*), più la concatenazione di eventuali attributi del documento.

2. AdES-T (Timestamped Electronic Signature): tra le informazioni in output vengono aggiunti dei campi (anch'essi firmati per maggiore integrità) che permettono di identificare la data e ora in cui è avvenuta la firma del documento.

3. AdES-C (Advanced Electronic Signature with Certificate Reference): in alcuni casi i documenti firmati devono rimanere verificabili nel lungo termine, per cui nell'output vengono aggiunti riferimenti alla chiave pubblica del firmatario oppure la chiave pubblica stessa.

Le diverse tipologie di firme AdES vengono inoltre declinate in quattro specifiche varianti, ciascuna ottimizzata per differenti tipi di documenti e contesti:

1. XAdES (XML AdES) è utilizzata per firmare documenti XML (Extensible Markup Language) [3].

2. CAdES (CMS AdES) è basato sul formato Cryptographic Message Syntax (CMS), anche noto come PKCS#7. Questo standard permette di firmare file PDF, Word,

o altri documenti elettronici non XML. CAdES garantisce l'integrità e l'autenticità dei documenti firmati, supportando funzionalità come il timestamping, l'inclusione di certificati e altre informazioni aggiuntive per conformarsi ai requisiti legali e normativi [3] .

3. PAdES è utilizzata per firmare esclusivamente file PDF (Portable Document Format) [4].

4. ASiC (Associated Signature Containers) permette di raggruppare in un unico contenitore (in formato ZIP) uno o più documenti e le loro firme elettroniche avanzate. È compatibile con le altre specifiche di firma elettronica avanzata come XAdES, CAdES, e PAdES, consentendo l'uso di queste firme all'interno del contenitore [3].

Questi sistemi fanno uso di algoritmi come RSA (Rivest-Shamir-Adleman Algorithm) oppure altri algoritmi basati su ECC (Elliptic Curve Cryptography). La debolezza di questi specifici sistemi basati su chiavi asimmetriche nei confronti dei quantum computers è nota da tempo: nel 1994, Peter Shor ha ideato un algoritmo quantistico (conosciuto come *Shor's Algorithm* [5]) che sfrutta le caratteristiche di RSA per comprometterne la sicurezza.

La sicurezza di RSA nei sistemi tradizionali (*legacy*) si basa sulla difficoltà di calcoli riguardanti i numeri primi e l'operazione di fattorizzazione. Tuttavia, i computer quantistici saranno in grado di violare la sicurezza di RSA utilizzando l'algoritmo di *Shor* in tempi esponenzialmente più rapidi rispetto agli attuali metodi di attacco.

Algoritmo	**Dimensione della chiave (in bit)**	**Livello di sicurezza in bit (computer attuale)**	**Livello di sicurezza in bit (computer quantistico)**
RSA-1024	1024	80	∼0
RSA-2048	2048	112	∼0
ECC-256	256	128	∼0
ECC-384	384	192	∼0
AES-128	128	128	∼64
AES-256	256	256	∼128

Tabella 1.1: Confronto del livello di sicurezza di un algoritmo se attaccato da un computer attuale e da un computer quantistico [6].

Per quanto riguarda i sistemi di crittografia basati su chiave simmetrica, questi risultano meno vulnerabili alla minaccia quantistica: si stima che il loro livello di sicurezza sarà dimezzato quando attaccati da computer quantistici [5]. Di conseguenza, mentre per gli algoritmi a chiave asimmetrica sono necessarie riformulazioni basate su nuovi costrutti

matematici, per quelli a chiave simmetrica potrebbe essere sufficiente raddoppiare la lunghezza delle chiavi per mantenere l'attuale livello di sicurezza. Anche in questo caso, la decisione di procedere in tale direzione dipende dall'interoperabilità con i sistemi attuali [6].

1.2 Il futuro della sicurezza: algoritmi Post-Quantum

Sebbene l'introduzione di computer quantistici sufficientemente potenti non sia prevista nel breve termine, è essenziale avviare la ricerca e la standardizzazione della Post-Quantum Cryptography quanto prima.

Gli attacchi non sono attualmente possibili, ma nulla impedisce a malintenzionati di effettuare una fase di *store* in questo periodo, cioè immagazzinare dati crittografati o firmati con le tecniche attuali per poi, una volta raggiunta la maturità tecnologica, decrittografarli o calcolare le chiavi private a partire dalle firme. Questa tecnica è nota come *Retrospective Decryption* [7]: in sintesi, tutti i dati elaborati con sistemi crittografici asimmetrici attuali dovrebbero essere considerati compromessi al momento dell'introduzione delle tecnologie quantistiche sul mercato.

Molte organizzazioni statali ed enti specializzati nel settore stanno proponendo attività di sviluppo per evitare anche questo tipo di scenario. Tra questi vi è il National Istitute of Standards and Tecnology [8]. Il progetto di standardizzazione avviato dal NIST nel Dicembre 2016 è stato utilizzato come base per la presente ricerca. L'intero processo è stato concepito come un'opera a medio-lungo termine, organizzata in *round*, in cui solo alcune tecnologie candidate possono procedere per ulteriori approfondimenti e perfezionamenti.

La standardizzazione riguarda le tecniche di PQC (Post-Quantum Cryptography), detta anche crittografia Quantum-Safe o Quantum-Resistant. Nel contesto della PQC si suppone che l'attaccante abbia a disposizione tecnologie comuni e quantistiche, mentre l'informazione attaccata è prodotta da algoritmi eseguiti su computer tradizionali.

Questa si differenzia dalla Quantum Cryptography (QC), in cui l'informazione da attaccare è anch'essa prodotta da tecnologia quantistica. La QC fa uso dei principi della meccanica quantistica per sviluppare tecniche di sicurezza che sono fondamentalmente diverse dalle tradizionali crittografie basate su algoritmi matematici. Invece di implementare algoritmi su un computer quantistico, QC sfrutta fenomeni quantistici come l'*entanglement* e il principio di indeterminazione per garantire la sicurezza dei dati. Un esempio noto di QC è la Quantum Key Distribution (QKD), che permette di generare e distribuire chiavi crittografiche in modo che qualsiasi tentativo di intercettazione venga immediatamente rilevato.

Il *NIST Post-Quantum Cryptography Standardization Process* [9] si concentra specificamente su algoritmi crittografici Quantum-Resistant basati su meccanismi a chiave asimmetrica. Inoltre, la competizione è stata divisa tra *signature schemes*, ovvero algoritmi per la firma digitale, e meccanismi di *public-key encryption*, cioè algoritmi di codifica e decodifica di informazioni. Nel primo caso, l'obiettivo è garantire l'autenticità e l'integrità del dato, mentre nel secondo si mira a mantenere la confidenzialità delle informazioni. Sebbene la teoria matematica e le tecniche utilizzate siano comuni ai due ambiti, i criteri di valutazione sono stati differenziati per esaminare ogni candidato in base all'obiettivo da raggiungere.

1.3 Obiettivi della ricerca

Lo scopo di questo elaborato è valutare i *signature schemes* candidati alla standardizzazione avviata dal NIST, concentrandosi in particolare sugli algoritmi che hanno partecipato e superato il terzo round del processo di selezione. Questi algoritmi rappresentano le soluzioni attualmente più promettenti per garantire la sicurezza delle firme digitali nell'era post-quantum.

L'obiettivo principale di questa ricerca non è quello di esplorare in dettaglio i principi matematici che rendono questi algoritmi resistenti ai computer quantistici, né di analizzare nel dettaglio il processo di selezione condotto dal NIST. L'intento è invece di confrontare le prestazioni di questi algoritmi, valutando la loro efficienza in termini di tempo di esecuzione, utilizzo delle risorse computazionali, e la loro interoperabilità con i sistemi esistenti.

Questa valutazione è cruciale in vista della transizione verso sistemi di firma digitale basati su Post-Quantum Cryptography. Infatti, mentre la sicurezza è la priorità assoluta, la praticabilità e l'efficacia operativa di questi algoritmi si riveleranno un ruolo decisivo per la loro adozione. Algoritmi che offrono un buon equilibrio tra sicurezza e prestazioni avranno maggiori probabilità di essere implementati su larga scala nei prossimi anni, soprattutto in contesti dove la compatibilità con i sistemi *legacy* è un fattore determinante.

Nei prossimi capitoli verranno descritti in dettaglio i test condotti, insieme all'ambiente di prova utilizzato e agli algoritmi candidati esaminati. L'obiettivo finale è fornire una valutazione alternativa a quella effettuata dal NIST, che possa guidare la scelta degli algoritmi più idonei per le future implementazioni di sistemi di firma digitale post-quantum, nell' ottica di un futuro in cui i computer quantistici saranno una realtà.

2

Algoritmi Quantum Safe

Per la stesura dell'elaborato sono stati considerati solo i sistemi di firma digitale presentati al NIST e, per specializzare maggiormente la ricerca, sono stati evitati i sistemi crittografici. Pur raggiungendo scopi diversi, sistemi di firma digitale e sistemi crittografici utilizzano la stessa teoria alla base. In generale, la Post-Quantum Cryptography (PQC), come la crittografia classica, si compone di algoritmi che sfruttano problemi o teoremi matematici complessi da risolvere con l'obiettivo di evitare la compromissione delle chiavi utilizzate per firmare o crittografare una o più comunicazioni.

Nella PQC i problemi matematici devono essere difficili da risolvere sia per le tecnologie *legacy* che per i computer quantistici, garantendo così la sicurezza contro attacchi avanzati. È fondamentale continuare a fare ricerca sui modelli già conosciuti e su nuovi modelli da utilizzare, poiché la loro sicurezza non è certificata. L'unico aspetto che si può affermare è che, con le conoscenze attuali, non sono noti attacchi in grado di compromettere facilmente tali schemi.

In questo elaborato verranno trattate solo le formulazioni matematiche più utilizzate per la creazione di algoritmi PQC. Analizzando i risultati del terzo *round* del *NIST Post-Quantum Cryptography Standardization Process* [9], si nota che i finalisti appartengono principalmente a due classi di algoritmi PQC:

1. CRYSTALS Dilithium: Lattice Based;
2. FALCON: Lattice Based;
3. SPHINCS+: Hash Based;

Sebbene molte altre proposte, oltre a quelle elencate, utilizzino le stesse basi per creare sistemi crittografici o di firma digitale, esse sono tutte diverse. Gli algoritmi, pur partendo dalla stessa teoria matematica, possono essere differenziati dalle scelte implementative o dall'influenza di altri costrutti. Spesso, per la costruzione di un algoritmo di crittografia

vengono combinati più problemi *NP-Hard*, ovvero problemi notoriamente difficili da risolvere. L'originalità di questi algoritmi risiede nei problemi utilizzati e nel modo in cui essi sono interconnessi. La difficoltà *NP-Hard* è cruciale per la sicurezza: questi schemi crittografici si affidano al fatto che, anche utilizzando un computer quantistico, risolvere tali problemi rimane computazionalmente proibitivo.

Nelle sezioni successive verranno esposte le idee alla base dei modelli presentati, senza eccessivi approfondimenti matematici, per rendere chiaro il contesto in cui operano questi algoritmi.

2.1 Algoritmi basati su Lattice

Dei quindici candidati iniziali del terzo *round* del progetto NIST, sette utilizzano *Lattice*. Ciò è dovuto alle caratteristiche intrinseche del problema, che è altamente configurabile e per il quale è dimostrato che sia i casi medi che quelli peggiori sono difficili da risolvere. I *lattice based problems* rappresentano quindi una delle aree più promettenti e studiate nella crittografia post-quantistica.

Un *lattice* (o reticolo) è un insieme di punti distribuiti regolarmente in uno spazio euclideo multidimensionale, generato da combinazioni lineari intere di un insieme di vettori linearmente indipendenti. Si può immaginare un *lattice* come una griglia tridimensionale (o di dimensioni superiori) di punti, dove la posizione di ogni punto è determinata da combinazioni specifiche di alcuni vettori di base.

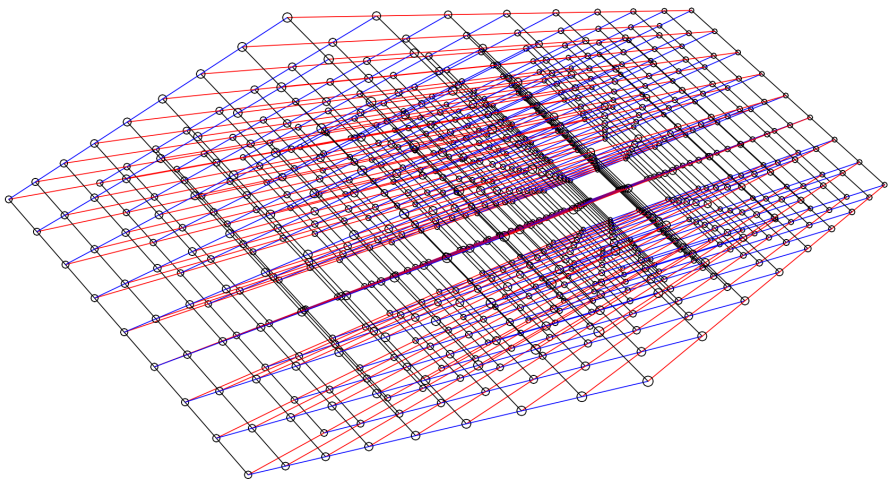

Figura 2.1: *Lattice* tridimensionale basato su un sottoinsieme di 9x9x9 punti. Fonte: [5]

Nella crittografia basata su *lattice*, la sicurezza degli schemi crittografici si basa sulla difficoltà di risolvere problemi matematici associati ai reticoli, come il problema del vettore più corto (*Shortest Vector Problem*, SVP) e il problema del vettore più vicino (*Closest*

Vector Problem, CVP). Il problema SVP richiede di trovare il vettore non nullo più corto in un reticolo, mentre il CVP consiste nel trovare il punto del reticolo più vicino a un punto dato che non appartiene al reticolo stesso [10]. Entrambi questi problemi sono notoriamente difficili da risolvere, specialmente in spazi ad alta dimensione.

2.1.1 Il Short Integer Solution Problem

In particolare il problema alla base dei reticoli è stato definito da Ajtai nel 1996, il *Short Integer Solution Problem (SIS)*:

> **Problema 3.5 (Il problema Short Integer Solution ($\mathbf{SIS}_{n,m,q,\beta}$))** Siano n, m, q interi positivi, e sia β un numero reale positivo. Data una matrice $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, scelta uniformemente a caso, lo scopo è trovare un vettore intero non nullo $\mathbf{z} \in \mathbb{Z}^m$ con norma euclidea $\|\mathbf{z}\| \leq \beta$ tale che $\mathbf{Az} = \mathbf{0} \in \mathbb{Z}_q^n$ [10]. (NIST IR 8413, 2022)

Cioè l'obiettivo del problema SIS è trovare un vettore con elementi *piccoli* che risolve una particolare equazione lineare con una matrice data. Questo problema è la base di molte costruzioni crittografiche per merito della sua difficoltà computazionale [7].

2.1.2 Estensioni del SIS Problem

Il problema *Learning With Errors* (LWE) è un'estensione del SIS, dove l'obiettivo è ancora risolvere un'equazione lineare, ma con l'aggiunta di un piccolo errore casuale ai risultati delle equazioni. Più precisamente, dato un vettore segreto, si generano dei prodotti scalari con alcuni vettori pubblici, e poi si aggiunge un piccolo errore casuale al risultato. L'obiettivo è recuperare il vettore segreto a partire da questi risultati *rumorosi*.

Il problema *Learning With Rounding (LWR)* è simile al LWE, ma anziché aggiungere un errore casuale, si "arrotonda" il risultato dei prodotti scalari ai valori più vicini. Questo arrotondamento introduce un'incertezza simile a quella introdotta dal rumore nel LWE, rendendo anche questo problema difficile da risolvere. LWR viene utilizzato in schemi crittografici efficienti poiché l'operazione di arrotondamento è spesso meno costosa computazionalmente rispetto all'aggiunta di un errore casuale.

Grazie alla grande varietà di formulazioni di problemi basati su lattice, è possibile creare sia sistemi di crittografia che sistemi di firma digitale con livelli di sicurezza molto elevati [5].

2.2 Algoritmi basati su Hash

Le funzioni *Hash* traducono stringhe di dati di lunghezza variabile, quali possono essere messaggi o interi documenti, in stringhe di lunghezza fissa. Nel fare ciò, garantiscono un insieme di proprietà fondamentali:

1. **Determinismo**: ogni volta che una funzione *hash* viene applicata allo stesso input, deve produrre sempre lo stesso output.

2. **Distribuzione uniforme**: gli output della funzione *hash* devono essere distribuiti uniformemente su tutto lo spazio possibile degli *hashcodes*.

3. **Resistenza alla collusioni**: conseguentemente alla distribuzione uniforme, la funzione *hash* deve essere ideata in modo che piccoli cambiamenti nell'input producano cambiamenti apparentemente casuali e significativi nell'output. Idealmente, deve essere impossibile trovare due input diversi che producano lo stesso output.

4. **Irreversibilità**: data la natura della funzione *hash*, non deve essere possibile ricostruire i passaggi effettuati da input a output per individuare l'input originale conoscendo solo l'output.

Da ciò si deduce che la funzioni *hash* sono resistenti ad attacchi di pre-immagine e di seconda pre-immagine [11], entrambe caratteristiche essenziali per la sicurezza dei sistemi crittografici:

1. Data una funzione *hash* h e un *hashcode* y, dovrebbe essere computazionalmente difficile trovare un input x tale che:

$$h(x) = y$$

2. Dato un input x_1 e il suo *hashcode* $h(x_1)$, dovrebbe essere computazionalmente difficile trovare un altro input x_2 tale che:

$$h(x_1) = h(x_2) \quad \text{con} \quad x_1 \neq x_2$$

Per le precedenti motivazioni, le funzioni *hash* sono uno dei *building blocks* fondamentali dei sistemi di firma crittografici e di firma digitale e, in certe condizioni, possono anche costituire l'unico *building block* di un sistema di firma digitale: è il caso degli algoritmi post-quantum *Hash Based* [7].

Anche per i computer quantistici è computazionalmente complesso individuare le pre-immagini di un *hashcode*, rendendo questi algoritmi sicuri (in termini di PQC) pur utilizzando concetti di crittografia classica senza ricorrere a problemi matematici particolarmente complessi. Gli algoritmi *Hash Based* non sono minacciati dallo *Shor's algorithm*,

ma lo sono dal *Grover's attack* [5], che può ridurre il livello di sicurezza di un sistema *Hash Based* a circa la metà rispetto a un attacco da parte di sistemi tradizionali (*legacy*).

In tal caso, è sufficiente aumentare la lunghezza degli *hashcode* generati da questi sistemi di crittografia e firma digitale per mantenere un elevato livello di sicurezza. Tuttavia, uno degli svantaggi di questo tipo di algoritmi è che richiedono tempi di computazione maggiori rispetto alle controparti *Lattice Based*.

Gli *hashcode* generati dalle funzioni *hash* per le operazioni di firma e verifica utilizzano un set di chiavi (chiave pubblica e privata) che in certi casi possono essere utilizzate più volte, in altri no. Tutto dipende da come viene ingegnerizzato l'algoritmo di generazione delle chiavi e di firma.

2.2.1 One Time Signatures vs Few Time Signatures

Nel 1975, l'intervento di Leslie Lamport nel campo delle firme digitali ha portato allo sviluppo delle *One-Time Signatures* (OTS), un approccio innovativo e semplice basato su funzioni hash crittografiche. Le OTS di Lamport utilizzano coppie di valori *hashati* per creare firme che possono essere utilizzate per garantire l'autenticità di un singolo messaggio. In uno schema OTS, la chiave privata è costituita da una serie di valori casuali, mentre la chiave pubblica è formata dagli hash corrispondenti di questi valori. Per firmare un messaggio, si selezionano alcuni dei valori dalla chiave privata e si rivelano, lasciando intatti gli altri. Questo metodo assicura che ogni chiave pubblica possa essere utilizzata per firmare solo un messaggio, rendendo lo schema semplice e altamente sicuro contro attacchi di retroazione (*Retrospective Decryption*), ma inefficace per un uso ripetuto, limitandone l'applicabilità a scenari molto specifici [5].

Per superare la limitazione intrinseca delle OTS di Lamport, ovvero la necessità di generare nuove coppie di chiavi per ogni messaggio firmato, è stato sviluppato l'approccio delle *Few-Time Signatures* (FTS) che utilizza la struttura ad albero di Merkle. L'albero di Merkle consente di combinare multiple OTS sotto un'unica chiave pubblica, che corrisponde alla radice dell'albero. Ogni foglia dell'albero è una chiave pubblica di una OTS, e la firma di un messaggio utilizza una di queste foglie insieme ai valori hash necessari per calcolare la radice. Questo approccio permette di utilizzare la stessa chiave pubblica per firmare più messaggi, riducendo significativamente l'overhead computazionale e di memoria rispetto a uno schema OTS puro, mantenendo al contempo un elevato livello di sicurezza. L'albero di Merkle rappresenta quindi un passo cruciale verso l'implementazione pratica di schemi di firma digitale basati su *hash*, ampliando il loro utilizzo a scenari con un numero moderato di messaggi da firmare [5].

Negli schemi FTS, il numero di firme che possono essere prodotte con una stessa chiave pubblica è limitato per garantire la sicurezza del sistema. Se si supera questo limite,

la probabilità che un attaccante possa sfruttare una collisione o un'altra vulnerabilità aumenta notevolmente. Questo limite è determinato principalmente dalla struttura dell'albero di Merkle utilizzato per combinare le firme one-time. Ogni volta che si firma un nuovo messaggio, si consuma una delle foglie dell'albero, riducendo progressivamente il numero di firme rimanenti [5].

2.2.2 Statefull vs Stateless

Con l'introduzione delle *Few-Time Signatures* (FTS), emerge una distinzione fondamentale tra schemi *stateful* e *stateless* nel contesto della gestione delle chiavi di firma. Questa distinzione riflette due approcci diversi alla gestione della sicurezza e della complessità operativa degli schemi di firma digitale basati su *hash* [11].

Negli schemi FTS *stateful*, ogni volta che si firma un nuovo messaggio, è necessario tenere traccia di quali chiavi one-time sono state utilizzate per evitare riutilizzi che comprometterebbero la sicurezza. In pratica, ciò significa che il firmatario deve mantenere uno stato interno che registra l'uso di ciascuna chiave privata all'interno della struttura ad albero. Questa gestione dello stato è essenziale per garantire che ogni chiave sia utilizzata una sola volta, impedendo potenziali attacchi che potrebbero derivare dalla firma multipla con la stessa chiave. Tuttavia, la necessità di mantenere e aggiornare questo stato introduce complessità, soprattutto in scenari distribuiti o in ambienti con risorse limitate, dove la perdita o la corruzione dello stato potrebbe risultare in errori critici e vulnerabilità di sicurezza.

Per superare tali complessità, sono stati sviluppati approcci *stateless*. In questi schemi non è necessario mantenere alcun registro delle chiavi già utilizzate. Tuttavia, l'assenza di uno stato comporta che ogni firma deve includere sufficienti informazioni per dimostrare la sua unicità e validità senza fare affidamento su uno stato preesistente. Questo implica che le firme diventano più grandi, poiché devono incorporare ulteriori dati per garantire la sicurezza, e che i tempi di calcolo aumentano, in quanto la generazione e la verifica delle firme richiedono operazioni più complesse [11].

La semplificazione introdotta dall'approccio *stateless* è quindi ottenuta al costo di un aumento della dimensione delle firme e dei tempi di computazione necessari per generarle e verificarle. Questi schemi sono progettati per operare in modo più indipendente e robusto rispetto alla necessità di mantenere uno stato coerente, ma richiedono maggiori risorse computazionali e di memoria per compensare la mancanza di uno stato gestito. Di conseguenza, gli schemi *stateless* tendono a essere più facili da implementare e gestire in ambienti distribuiti o con meno controllo centralizzato, ma con un compromesso in termini di efficienza e dimensioni dei dati crittografici.

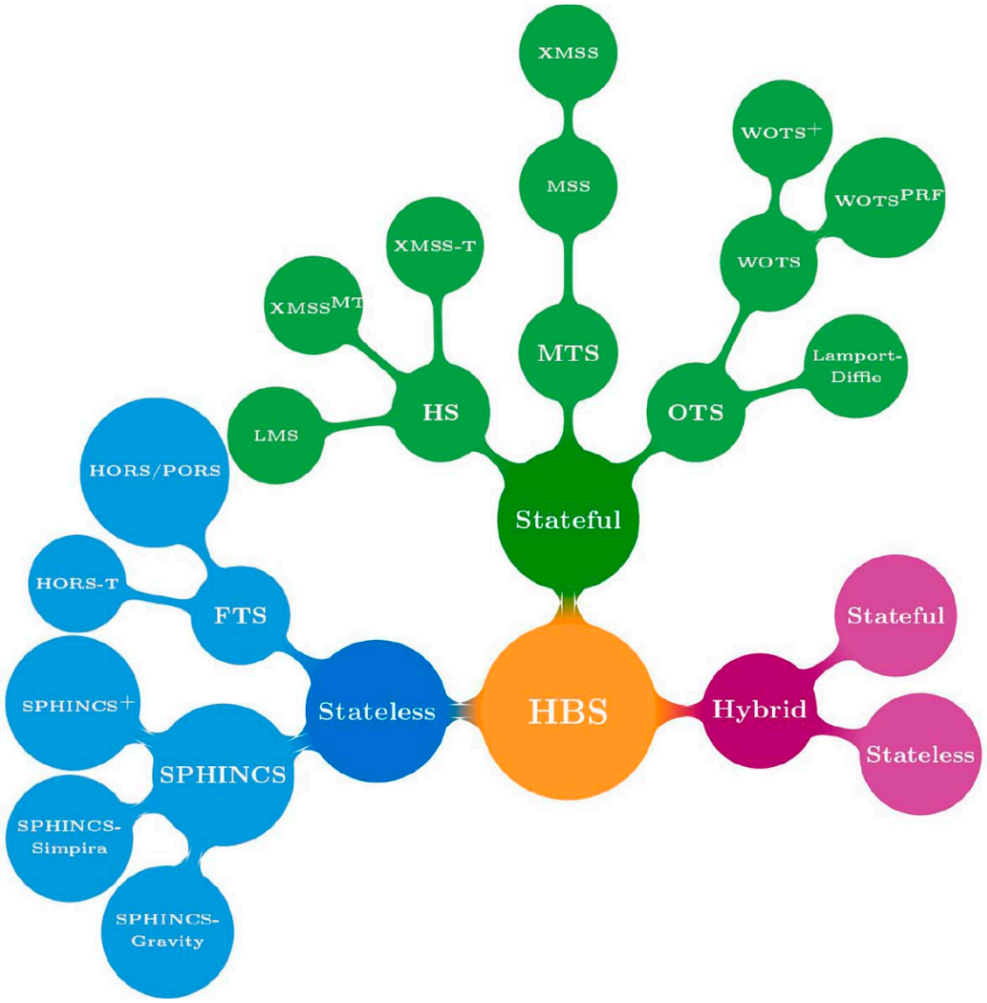

Figura 2.2: Tassonomia e ramificazione dei sistemi di firma *Hash Based*. Fonte: [11]

3

Signature Schemes analizzati

Come già anticipato nelle precedenti sezioni, per la selezione degli algoritmi e come fonte di informazione principale è stato utilizzato il progetto di standardizzazione del NIST[9]. Questo capitolo introdurrà nel dettaglio i punti chiave del progetto e i candidati presi in esame.

3.1 La standardizzazione attuata dal NIST

Il *NIST Post-Quantum Cryptography Standardization Process* [9] è un'iniziativa avviata nel 2016 con l'obiettivo di sviluppare nuovi standard per la crittografia post-quantum. Questo processo, che si estende su un periodo di medio-lungo termine, ha invitato ricercatori e sviluppatori di tutto il mondo a proporre soluzioni per la crittografia post-quantum. Entro un anno sono state ricevute più di ottanta candidature di cui la maggior parte rispettava i requisiti minimi e di accettabilità imposti. Dunque, tra il Q4 2017 e Q1 2019 sono stati valutati tutti i candidati ed è stato rilasciato il primo *NIST report*, con tutte le analisi e le conclusioni dedotte per ciascun candidato.

3.1.1 Svolgimento

Fin da subito il progetto è stato ideato come una selezione a *round*: inizialmente ne erano stati preventivati tre per raggiungere un grado di rifinitura adeguato sugli algoritmi proposti, tuttavia, il NIST ha indetto il quarto *round* per permettere ad alcune proposte di rientrare nella standardizzazione fin da subito. Nella tabella 3.1 vengono elencati i momenti chiave riguardo lo sviluppo dei *round*.

Dal rilascio della reportistica sul secondo *round* è stato introdotto il concetto di candidato finalista e candidato alternativo:

- Finalista: algoritmo che soddisfa le richieste e che, al momento della produzione del report, sembra idoneo all'implementazione dei nuovi sistemi PQC.
- Alternativo: algoritmo che per alcuni aspetti (di bassa rilevanza) non è adatto alla standardizzazione su cui però il NIST vuole continuare a dedicare ricerca e sviluppo per una miglior rifinitura.

Essenzialmente i candidati alternativi sono sistemi di firma digitale che raggiungono tutti i requisiti di sicurezza ma che hanno prestazioni inferiori rispetto ai finalisti. Per questo motivo il NIST permette loro di partecipare ai successivi *rounds*, allo scopo di introdurre modifiche indirizzate all'incremento delle performance [10].

Data	**Evento**
Febbraio 2016	Standardizzazione PQC: Annuncio e presentazione del bando del NIST, tenuta al PQCrypto 2016
Dicembre 2016	Avviso nel registro federale – Richiesta di nomine per algoritmi di crittografia post-quantum a chiave pubblica
Novembre 2017	Scadenza per la sottomissione per il processo di standardizzazione PQC del NIST
Dicembre 2017	Candidati del primo *round* annunciati. Inizio del periodo di commento pubblico sui candidati del primo *round*.
Gennaio 2019	Candidati del secondo *round* annunciati. NIST IR 8240, *Rapporto sul Primo Round del Processo di Standardizzazione della Crittografia Post-Quantistica del NIST*, pubblicato. Inizio del periodo di commento pubblico sui candidati del secondo *round*.
Aprile 2020	Il NIST ha invitato commenti da parte dei sottomittenti e della comunità per informare il processo decisionale per la selezione dei candidati del terzo *round*.
Luglio 2020	Finalisti del terzo *round* e candidati alternativi annunciati. NIST IR 8309, *Rapporto sul Secondo Round del Processo di Standardizzazione della Crittografia Post-Quantistica del NIST*, pubblicato. Inizio del periodo di commento pubblico sui candidati del terzo *round*.
Luglio 2022	Algoritmi candidati per la standardizzazione annunciati, insieme ai candidati alternativi che avanzano al quarto round. NIST IR 8413, *Rapporto sul Terzo Round del Processo di Standardizzazione della Crittografia Post-Quantistica del NIST*, pubblicato.
Agosto 2024	Pubblicazione dei primi standard su i finalisti del terzo *round*: CRYSTALS-Kyber, CRYSTALS Dilithium e SPHINCS+ [12]
In corso	Valutazione dei candidati inviati al quarto *round*: produzione del nuovo commento pubblico sulla base dei nuovi sviluppi e dei feedback degli esperti.

Tabella 3.1: Cronologia del processo di standardizzazione PQC del NIST [10]

Entrambe le tipologie di algoritmi vengono inviate al *round* successivo per la valutazione. Ovviamente tra un *round* e il successivo è possibile effettuare delle modifiche minori al proprio codice al fine di ottimizzare le prestazioni, le risorse richieste oppure al fine di correggere vulnerabilità individuate dal NIST o dal team di ricerca stesso.

Le valutazioni non sono definitive, per cui è possibile che un candidato alternativo divenga finalista (del *round*) oppure venga scartato. Ad esempio, uno dei finalisti del secondo report per quanto riguarda i *Digital Signature Systems* (DSS) è *Rainbow* [13], che è stato rifiutato durante il terzo round a causa di una vulnerabilità importante rilevata nella sicurezza dell'algoritmo. Al contrario *SPHINCS+* [14] è passato da essere un candidato alternativo ad essere un finalista.

3.1.2 Regolamentazione

Durante la fase di raccolta candidature dei *rounds*, tutti i partecipanti alla competizione hanno dovuto presentare un *Submission Package*. Il pacchetto consegnato era strutturato in una maniera ben definita dal NIST, al fine di poter valutare più facilmente il singolo candidato e di poterlo confrontare equamente con gli altri [15].

Ciascun *Submission Package*, consegnato per via digitale, doveva presentare al suo interno:

1. **Struttura dei file**: i file devono essere organizzati in cartelle con una separazione logica chiara. Il NIST non ha mai definito uno standard preciso, in generale l'idea è di suddividere il materiale in cartelle dedicate alla documentazione, al codice sorgente, ai test unitari e di performance.

2. **Documentazione**: ogni proposta deve essere accompagnata dai fondamenti teorici utilizzati, dalla documentazione del codice, un approfondimento su tutti gli aspetti implementativi e dalle istruzioni su come compilare ed eseguire il codice sorgente consegnato nei vari ambienti.

3. **API**: il NIST ha definito dei prototipi per le funzioni che accomunano tutti i sistemi di firma digitale, i candidati sono invitati a seguire tali prototipi per quanto possibile. Parte dei prototipi è riportato nella sezione di codice 1 [16].

4. **Codice sorgente**: deve essere incluso il codice C degli algoritmi necessari ad implementare il sistema di firma digitale. Possono essere incluse implementazioni anche in altri linguaggi di programmazione. È stato scelto il linguaggio C per l'ampia portabilità e il vasto supporto alla compilazione su vario hardware.

5. **Codice dei performance test**: vanno inclusi anche degli script per provare la correttezza e le performance di parte degli algoritmi o dell'intero sistema di firma digitale.

Listing 1 Codice C dei prototipi rilasciati dal NIST per la firma digitale [16].

```c
// === PROTOTIPO PER GENERAZIONE DELLE CHIAVI ===
// Questa funzione genera una coppia di chiavi pubblica e privata.
↪  La chiave pubblica viene memorizzata nel buffer 'pk', mentre
↪  la chiave privata viene memorizzata nel buffer 'sk'.
int crypto_sign_keypair(unsigned char *pk, unsigned char *sk);

// === PROTOTIPO PER FIRMA DEL MESSAGGIO ===
// Firma un messaggio utilizzando la chiave privata 'sk'. La firma
↪  e il messaggio 'm' vengono concatenati e memorizzati nel
↪  buffer 'sm'. La lunghezza totale del messaggio firmato viene
↪  memorizzata in 'smlen'.
int crypto_sign(unsigned char *sm, unsigned long long *smlen,
           const unsigned char *m, unsigned long long mlen,
           const unsigned char *sk);

// === PROTOTIPO PER VERIFICA DELLA FIRMA ===
// Verifica la firma di un messaggio utilizzando la chiave
↪  pubblica 'pk'. Se la verifica ha successo, il messaggio
↪  originale viene estratto e memorizzato nel buffer 'm' e la sua
↪  lunghezza in 'mlen'.
int crypto_sign_open(unsigned char *m, unsigned long long *mlen,
                const unsigned char *sm, unsigned long long smlen,
                const unsigned char *pk);
```

Nel caso un team presenti nuovamente la propria ricerca ad un *round* successivo, il *submission package* deve contenere anche un *changelog* con le modifiche apportate rispetto alla versione precedentemente valutata dal NIST.

Il codice C da consegnare in realtà deve includere due (o più) implementazioni dello stesso sistema, di seguito le ulteriori disposizioni della consegna:

- Versione di riferimento (*REF Version*): questo codice ha lo scopo di verificare l'effettivo funzionamento del sistema di firma. Deve essere compatibile con tutte (o quasi) le piattaforme hardware (x84, x64, ARM, eccetera). Il codice deve essere commentato e facilmente leggibile.

- Versione ottimizzata (spesso definita *AVX2 Version*): il team di ricerca deve includere una o più versioni degli stessi algoritmi ma con delle ottimizzazioni che

permettano migliori performance. Non è richiesto che il codice sia compatibile con tutte le piattaforme (alcune ottimizzazioni che sfruttano hardware dedicato potrebbero essere incompatibili con altri sistemi) e non deve essere necessariamente commentato.

Infine, dal secondo *round* il NIST richiede ai candidati di produrre un ulteriore documentazione il cui scopo è comprendere il livello di sicurezza del proprio algoritmo e le eventuali vulnerabilità che possono colpirlo o aspetti a cui è sensibile. Questo lavoro di ricerca svolto dai vari team viene poi valutato dal NIST ed esteso (se necessario) durante la stesura del report sui candidati [15].

3.1.3 Criteri di valutazione

Per decidere quali algoritmi sono da considerarsi finalisti, alternativi oppure rifiutati, il NIST ha specificato gli aspetti considerati durante la valutazione nella pagina dedicata al progetto [15]. Di seguito vengono elencati in ordine di importanza:

1. **Sicurezza**: è il criterio più importante su cui vengono valutati i candidati. Vengono verificate le teorie matematiche alla base del sistema di firma e successivamente la resistenza ad attacchi provenienti da computer quantistici.

2. **Costi e performance**: include dei test sull'efficienza computazionale in termini di tempo e spazio richiesti dagli algoritmi che compongono il sistema di firma digitale.

3. **Particolarità e caratteristiche di implementazione**: si valutano aspetti come la flessibilità di un sistema e l'eventuale facilità di adozione. Questo criterio valuta anche l'eventuale interoperabilità con gli attuali protocolli e standard.

3.1.4 I livelli di sicurezza

Definire la sicurezza di una proposta post-quantum è complesso poiché la valutazione viene effettuata su ciò che attualmente si conosce dei quantum computers e degli algoritmi che, eseguiti su di essi, possono minacciare le tecniche attuali (ad esempio *Shor's algorithm* e *Grover's algorithm*). In parte, viene calcolata tenendo conto di ipotesi sulle future capacità dei computer quantistici.

Anche la classificazione in livelli di sicurezza non è un aspetto banale da considerare. Una pubblicazione passata del NIST specifica un criterio con cui collegare la sicurezza degli algoritmi a chiave simmetrica con la sicurezza degli algoritmi a chiave asimmetrica in funzione della lunghezza delle chiavi in bits e dei tempi di compromissione richiesti. Questi stessi criteri ovviamente possono essere utilizzati per definire il grado di sicurezza atteso degli algoritmi post-quantum [17].

Il NIST si riferisce ai livelli di sicurezza tramite un numero da 1 a 5, dove 1 indica il livello di sicurezza più debole e 5 il più forte. Un livello è associato alla difficoltà computazionale richiesta per compromettere un certo algoritmo a chiave simmetrica. Un algoritmo a chiave asimmetrica (tra cui gli algoritmi di PQC trattati in questa tesi) appartiene ad un certo livello di sicurezza se la difficoltà computazionale richiesta per comprometterlo è nello stesso ordine di quella dell'algoritmo a chiave simmetrica associato a tale livello [17].

Nella maggior parte dei casi, i bits di sicurezza coincidono con la lunghezza delle chiavi degli algoritmi a chiave simmetrica associati al livello di sicurezza. Poiché l'indice di confronto del livello 3 è AES-128 (algoritmo a chiave simmetrica), i bit di sicurezza sono 128 per via della lunghezza della chiave.

Livello di Sicurezza	**Bits di Sicurezza**	**Algoritmo a Chiave Simmetrica**	**Lunghezza Modulus RSA (in bit)**
1	80	2TDEA	$L = 1024$
2	112	3TDEA	$L = 2048$
3	128	AES-128	$L = 3072$
4	192	AES-192	$L = 7680$
5	256	AES-256	$L = 15360$

Tabella 3.2: Definizione del livello di sicurezza secondo il NIST [17].

Nella tabella 3.2 vengono riassunte le definizioni del NIST per i vari livelli di sicurezza. Ad esempio RSA con *Modulus* da 3072 bits appartiene al livello di sicurezza 3, cioè la difficoltà computazionale richiesta da un computer attuale (*legacy*) per la compromissione è circa quello richiesto per compromettere AES-128.

Implementazioni di RSA con *Modulus* inferiori hanno livelli di sicurezza inferiori. Per i sistemi *legacy* RSA-1024 possiede una sicurezza di livello 1 (cioè comparabile con un algoritmo a chiave simmetrica ad 80 bits), tuttavia per la tabella 1.1 è noto anche che il livello di sicurezza rispetto ad attacco proveniente da un quantum computer è circa nullo (a causa della vulnerabilità all'algoritmo di Shor).

Al contrario, gli attuali algoritmi PQC sono studiati per mantenere lo stesso livello di sicurezza assegnato e quindi i rispettivi tempi di computazione in caso di attacco, sia se l'attacco proviene da sistemi tradizionali, sia se proviene da computer quantistici.

Il NIST è intenzionato a standardizzare per primi gli algoritmi PQC che offrano i livelli di sicurezza compresi tra 1 e 3, poiché essi sono i livelli attualmente più utilizzati per gli attuali sistemi di crittografia e firma (RSA e ECC) in quanto offrono il giusto equilibrio tra sicurezza e prestazioni. Tuttavia, il NIST è interessato anche a proposte che permettano di raggiungere i livelli di sicurezza più elevati così da poter essere già pronti ad un futuro più prossimo che necessita di maggior sicurezza [10].

3.1.5 Ulteriori aspetti di sicurezza

Il NIST infine valuta la sicurezza dei candidati anche in maniere alternative alla definizione di livello di sicurezza. Quest'ultimo valuta la debolezza ad attacchi crittanalitici diretti, cioè attacchi il cui obiettivo è compromettere i sistemi attraverso analisi matematica o combinatoria delle informazioni di input e output (ad esempio attacchi *brute force* per la ricerca della chiave).

Il secondo aspetto valutato per tutti i candidati è la resistenza a:

1. **Side Channel Attacks**: attacchi informatici che sfruttando informazioni *collaterali* al processo di firma o crittografia, ad esempio i tempi di esecuzione o le risorse utilizzate per un singolo processo [10].

2. **Multikey Attacks**: sfruttando uno scenario in cui sono utilizzate più chiavi, gli attaccanti possono cercare di dedurre informazioni sulle chiavi sfruttando le correlazioni tra gli input e gli output delle operazioni effettuate con le diverse chiavi.

Infine, un altro aspetto rilevante è la *Forward Secrecy*: è una proprietà dei sistemi crittografici o di firma digitale. Essa garantisce che in caso di compromissione della chiave privata in un certo momento, tutte le comunicazioni precedenti a tale data rimangano sicure e valide. È necessario quindi che ogni sessione di firma risulti indipendente dalle altre sebbene le chiavi coinvolte siano le stesse.

3.1.6 Algoritmi pronti per la standardizzazione

Come già anticipato, questa ricerca si concentra sul valutare i *Signature Schemes* che hanno passato il terzo *round*, siano essi finalisti oppure alternativi, e che sono stati dichiarati come pronti al processo di standardizzazione da parte del NIST.

Tra questi, alcuni di essi hanno recentemente terminato il processo di standardizzazione, dunque il NIST invita gli amministratori dei sistemi che sfruttano principi di firma digitale (o algoritmi di crittografia a chiave asimmetrica) ad implementare le nuove specifiche rilasciate [12].

Signature Scheme	**Stato di partenza**	**Stato di uscita**
CRYSTALS Dilithium	Finalista	Standardizzato (08/2024)
FALCON	Finalista	Standardizzazione in corso
Rainbow	Finalista	Rifiutato
GeMSS	Alternativo	Rifiutato
Picnic	Alternativo	Rifiutato
SPHINCS+	Alternativo	Standardizzato (08/2024

Tabella 3.3: Schemi di firma digitale ammessi al terzo *round* e i loro risultati [10].

La tabella 3.3 sottolinea che solo tre dei candidati al *NIST Post-Quantum Cryptography Standardization Process* [9] sono stati selezionati per l'effettiva standardizzazione (di cui due recentemente standardizzati). Gli altri tre candidati sono stati rimossi poiché affetti da una vulnerabilità che ha compromesso in maniera significativa la sicurezza.

Nelle prossime sezioni verranno approfondite nel dettaglio le caratteristiche di ciascuno dei tre candidati.

3.2 CRYSTALS Dilithium

CRYSTALS Dilithium è uno dei più promettenti schemi di firma digitale *Lattice Based*, sviluppato come parte del progetto CRYSTALS (*Cryptographic Suite for Algebraic Lattices*)[18]. CRYSTALS è una suite che comprende due primitive appositamente studiate per la standardizzazione del NIST: *Kyber* per i meccanismi di *secure key-encapsulation* (KEM) e *Dilithium* per i sistemi di firma digitale. Questo progetto partecipa quindi ad entrambe le partizioni della competizione indetta dal NIST per standardizzare gli algoritmi crittografici resistenti agli attacchi quantistici [19].

3.2.1 Sicurezza e basi teoriche

I problemi *NP-Hard* su cui si basa CRYSTALS Dilithium sono:

1. SVP (Shortest Vector Problem): ricerca del vettore più corto che collega due punti del reticolo senza passare per l'origine.

2. LWE (Learning With Errors): estensione del *SIS problem* già citato in precedenza, costituisce il cuore della sicurezza della chiave pubblica [10].

Il processo di firma si basa sull'approccio *Flat-Shamir with aborts*, una tecnica che riduce la probabilità di fallimento della firma tramite un *riavvio controllato*: se durante il processo di firma alcuni vincoli non sono soddisfatti, l'intera procedura viene ripetuta più volte fino a che non si ottiene un risultato valido e sicuro [20].

Per assicurarsi che tutti i tentativi siano diversi tra un riavvio e l'altro, viene mantenuto un contatore dei tentativi il cui valore è incluso nel contenuto da firmare, cosicché durante il calcolo degli hash si ottengano sempre valori unici. La robustezza di queste tecniche è supportata da numerose analisi che ne dimostrano la resistenza agli attacchi (tradizionali e quantistici) [20].

CRYSTALS Dilithium implementa un algoritmo di firma *zero-knowledge*, ovvero durante la produzione della firma e nella firma stessa non viene rilasciata alcuna informazione che può portare alla scoperta della chiave privata utilizzata. Questo aspetto è di cruciale

importanza poiché garantisce sicurezza anche se la stessa chiave privata è utilizzata per un numero elevato di firme. Questa è un'altra delle conseguenze del *rejection sampling* introdotto precedentemente con il tema del *riavvio controllato* [20].

3.2.2 Caratteristiche

CRYSTALS Dilithium è progettato fin dalla sua prima versione per essere altamente modulare. Possiede numerosi parametri che permettono di adattare il suo livello di sicurezza in base alle necessità richieste, siano esse efficienza o maggiore protezione. Questa flessibilità permetterà allo schema di offrire un livello adeguato di sicurezza nel lungo temine [21].

Un'altra caratteristica distintiva sono le ridotte dimensioni della chiave pubblica. Rispetto agli algoritmi Lattice Based che utilizzano campionamenti Gaussiani già esistenti CRYSTALS Dilithium presenta una chiave pubblica inferiore di circa 2.5 volte. Questo vantaggio è stato ottenuto dal team di ricerca rimuovendo completamente il campionamento Gaussiano. Quest'ultimo, negli algoritmi *Lattice Based*, è utilizzato per generare vettori e polinomi necessari a costruire i reticoli su cui svolgere i problemi di SVP o LWE. La casualità dei vettori generati serve a rendere complessa la risoluzione di tali problemi.

CRYSTALS Dilithium utilizza delle distribuzioni binomiali per generare campioni: esse sono molto più semplici da implementare e anche più efficienti. Tuttavia, il motivo che più giustifica la rimozione del campionamento gaussiano è la sua vulnerabilità ai *side channel attacks*, soprattutto quelli basati sulla misura dei tempi di esecuzione.

La novità di CRYSTALS Dilithium è che per primo, rispetto ad altri candidati, è stato in grado di offrire processi di firma, verifica e generazione delle chiavi a tempo costante, rendendo lo schema di firma resistente alle minacce basate sul *timing* [21].

Un altro fattore che offre protezione dai *side channel attacks* è la dimensione costante delle firme prodotte. Questo aspetto non solo semplifica l'implementazione e la rende più compatibile con gli attuali protocolli, ma garantisce maggiore resistenza agli attacchi criptoanalitici che sfruttano le variazioni di dimensioni tra coppie di input e output [21].

Purtroppo, CRYSTALS Dilithium non è ancora pronto per la sua esecuzione nell'ambito IoT (*Internet of Things*). I dispositivi IoT sono sistemi caratterizzati da hardware limitato ma che possono offrire servizi di automazione, per questo motivo negli ultimi anni il loro utilizzo è in continua crescita. La ragione principale per cui questo schema di firma non è compatibile con parte di questi sistemi è l'utilizzo della memoria: spesso la richiesta di RAM (Random Access Memory) risulta essere eccessiva [20]. Il team di ricerca di CRYSTALS Dilithium ha già annunciato che lavoreranno a miglioramenti per una maggiore compatibilità verso questi dispositivi.

3.2.3 Versioni disponibili

Nei *submission packages* CRYSTALS-Dilithium è sempre stato consegnato nelle due versioni minime richieste. La versione di riferimento (REF) implementa i meccanismi fondamentali dello schema di firma digitale, tra cui le primitive fondamentali a garantire il tempo costante e le dimensioni costanti per la resistenza agli attacchi criptoanalitici. La versione ottimizzata viene definita, dal team di ricerca di CRYSTALS Dilithium, *AVX2 Version*. Questa particolare implementazione dell'algoritmo sfrutta le ottimizzazioni hardware disponibili solo per alcune architetture di processori, tra cui quelle prodotte da *Intel* e *AMD*.

Il termine *AVX2* significa *Advanded Vector Extensions 2*: i processori con questa funzionalità implementano una serie di istruzioni che ottimizzano le operazioni sui vettori a livello hardware. Per tale motivo, la *AVX2 Version* di CRYSTALS Dilithium porta miglioramenti significativi nelle operazioni di espansione del *seed* nelle matrici polinomiali e nelle operazioni di moltiplicazione polinomiale [20].

Essendo che gli algoritmi *Lattice Based* lavorano principalmente con reticoli, matrici e vettori, essi sono profondamente potenziati dall'introduzione di questo set di istruzioni macchina nei processori. Ovviamente questo implica che la versione ottimizzata di CRYSTALS Dilithium non è compatibile con tutti i sistemi. Ad esempio, a differenza della *REF Version*, la *AVX2 Version* non è compatibile con la maggior parte dei processori ARM [9].

3.2.4 Changelog

CRYSTALS Dilithium partecipa fin dal primo *round* al *NIST Post-Quantum Cryptography Standardization Process*[9], tuttavia con il tempo ha subito delle modifiche più o meno rilevanti. Di seguito un breve riassunto dei cambiamenti [21]:

1. Introduzione del comportamento randomizzato: inizialmente lo schema era esclusivamente deterministico. Con la nuova versione la generazione del *seed* viene ottenuta in maniera casuale. Il *seed* è un valore utilizzato a sua volta per generare numeri pseudo-casuali. Nella versione deterministica dell'algoritmo, tale valore era ottenuto da un hash della chiave privata: ciò garantiva efficienza e riproducibilità ma introduceva anche dei rischi [*Round 1* $\rightarrow$ *Round 2*].

2. Ottimizzazione e introduzione di nuovi parametri per tutte le versioni di CRYSTALS Dilithium: ciò ha aumentato la flessibilità dell'algoritmo, rendendolo più configurabile lato prestazioni o sicurezza [*Round 2* $\rightarrow$ *Round 3*].

3. Riduzione dei bit della chiave pubblica e semplificazione del meccanismo di campionamento, dal punto di vista implementativo [*Round 2* → *Round 3*].

3.3 FALCON

FALCON è l'acronimo di *Fast Fourier Lattice Based Compact Signatures over NTRU*. Partecipa alla competizione dal primo *round* ed è fin da subito diventato uno dei principali candidati alla standardizzazione per via delle sue peculiarità [22].

3.3.1 Sicurezza e basi teoriche

FALCON è uno *signature scheme Lattice Based* ma, a differenza di CRYSTALS Dilithium, poggia le propria fondamenta teoriche su altri paradigmi e problemi. Più specificatamente utilizza il problema NTRU così definito:

> **Problema 3.6 (Il problema *Search-NTRU*$_{R,q,D,\gamma}$)** Sia q un intero positivo, γ un numero reale positivo, e sia R un anello della forma $R = \mathbb{Z}_q[x]/\Phi$ (dove Φ è un polinomio monico). Dato un elemento $h \in R$ estratto da una distribuzione D, tale che esistano coppie non nulle $(f, g) \in R^2$ che soddisfano $h \cdot f = g \mod q$ e hanno piccole norme euclidee $\|f\|, \|g\| \leq \sqrt{q}/\gamma$, trovare una tale coppia (f, g) [10]. (NIST IR 8413, 2022)

La sicurezza di FALCON deriva dalla difficoltà di risolvere equazioni basate su questo tipo di reticoli, una caratteristica che lo rende particolarmente adatto a resistere agli algoritmi quantistici come quello di Shor, che compromettono gli schemi tradizionali come RSA [23].

Un'altra importante differenza rispetto a CRYSTALS Dilithium è il paradigma utilizzato. Mentre il precedente candidato basa il proprio processo sull'approccio *Flat-Shamir with aborts* , FALCON utilizza lo schema *hash-and-sign*. La sicurezza di questo schema è già stata dimostrata nel modello Oracle pre-quantistico e post-quantistico [23].

Come suggerisce la definizione dell'acronimo FALCON, l'obiettivo principale che il team di ricerca voleva raggiungere era la compattezza della chiave pubblica e della firma. Inizialmente avevano valutato di produrre un *signature scheme Hash Based* tuttavia questo avrebbe portato a una dimensione delle chiavi molto ridotta a discapito di una dimensione eccessiva della firma del messaggio. Al contrario, implementando un *Lattice Based* basato su NTRU, sono riusciti a minimizzare l'impatto in memoria di questi due oggetti [23].

La dimensione della firma non è costante, bensì firmando più volte lo stesso messaggio con la stessa chiave privata, si possono ottenere lunghezze della firma (in bit) diverse.

Questo perché durante il processo di generazione della firma vengono utilizzati dei valori casuali inclusi nella firma stessa. Secondo gli studi del team di ricerca di FALCON durante la produzione del *submission package*, questo aspetto non dovrebbe esporre vulnerabilità a *side channel attacks*, tuttavia è possibile configurare FALCON per produrre firme di lunghezza costante, ottenute tramite lo *zero-padding* della firma originale.

Nonostante FALCON utilizzi il *gaussian sampling*, è stato comunque trovato un modo di implementare i processi di firma e verifica in tempi costanti, andando a rimuovere le vulnerabilità ai *side channel attacks* legati ai tempi di esecuzione.

Purtroppo nuovi studi, successivi al report del terzo *round* del NIST, hanno rilevato che FALCON è suscettibile ad attacchi *side channel* basati sull'analisi elettromagnetica (EM): questi studi hanno dimostrato che è possibile estrarre le chiavi segrete misurando la radiazione elettromagnetica emessa dalle CPU durante l'esecuzione dei processi di FALCON [24].

Gli studi in questione sono stati eseguiti su sistemi ARM Cortex-M4 e, secondo i test, sarebbero sufficienti qualche migliaio di operazioni di firma per ottenere abbastanza informazioni per la ricostruzione della chiave privata. Gli autori di questa ricerca suggeriscono anche dei metodi per mitigare la vulnerabilità [24]:

1. **Hiding**: tecnica che mira a rendere costante il consumo energetico dell'hardware durante i procedimenti più delicati oppure che aggiunge un segnale di disturbo intervallando tali procedimenti con l'esecuzione di routine apposite.

2. **Masking**: tecnica il cui scopo è randomizzare i valori coinvolti nei procedimenti da mascherare. Rimuovere la correlazione tra i valori intermedi, input e output rende i *side channel attacks* inefficienti.

3.3.2 Caratteristiche

Uno dei punti di forza di FALCON è l'efficienza computazionale ottenuta tramite l'uso del *Fast Fourier Sampling*. Questo metodo permette allo schema di firma digitale di eseguire rapidamente le operazioni di generazione e verifica della firma.

Durante il campionamento FALCON svolge molte operazioni che coinvolgono numeri *floating point* (virgola mobile) con 53 bit di precisione. Questo aspetto tecnico rende l'implementazione degli algoritmi molto complessa ed efficiente solo per processori che possiedono una FPU (Floating Point Unit) dedicata. La precisione dei calcoli in virgola mobile è essenziale per garantire il livello adeguato di sicurezza: se si utilizza una bassa precisione la sicurezza diminuisce, al contrario aumentare troppo la precisione aumenta i tempi di esecuzione [23].

L'obiettivo principale di FALCON (ovvero la compattezza) è stato raggiunto: tra i candidati è quello ad ottenere la dimensione inferiore considerando l'unione di chiave pubblica e firma della comunicazione. Questo caratteristica abbatte di molto i tempi di trasmissione delle comunicazioni firmate, altro aspetto cruciale da valutare per l'implementazione degli algoritmi post-quantum nei sistemi di firma digitale [10].

Difatti, in relazione ai livelli di sicurezza raggiunti (1 e 5), i tempi di trasmissione sono pari o inferiori a quelli ottenibili con gli attuali sistemi, come RSA.

Come CRYSTALS Dilithium, FALCON ha un design modulare, ovvero sono disponibili un insieme di parametri che permettono di configurare l'equilibrio tra sicurezza e prestazioni. Per rendere più sicura l'implementazione di NTRU e delle distribuzioni gaussiane, i set di parametri sono limitati. Per questo motivo i livelli di sicurezza raggiunti da FALCON sono *solo* 1 e 5 [23].

L'utilizzo dell'approccio *hash-and-sign* permette a FALCON di implementare una funzionalità unica tra i candidati, ovvero il Message Recovery: chi verifica una firma è in grado di recuperare l'intero messaggio attraverso delle elaborazioni sulla firma stessa [23]. Questo meccanismo ha diversi vantaggi e limitazioni:

1. **Riduzione del tempo di trasmissione**: non è più necessario trasmettere la comunicazione originale e la firma separatamente ma è sufficiente trasmettere solo la firma. In certi casi può ridurre i tempi di trasmissione.

2. **Aumento delle dimensioni della firma**: includere sufficienti informazioni per la ricostruzione del messaggio aumenta la dimensione della firma.

3.3.3 Versioni disponibili

Le versioni del codice sorgente C consegnate dal team di ricerca di FALCON sono le stesse discusse per CRYSTALS Dilithium:

1. **REF Version**: versione compatibile con la maggior parte dei sistemi esistenti, il cui scopo è documentare il modello di funzionamento dello schema di firma digitale.

2. **AVX2 Version**: versione ottimizzata per eseguire più velocemente calcoli che coinvolgono vettori e matrici. Questo codice sorgente tuttavia risulta compilabile solo su processori che supportano le istruzioni macchina AVX2.

Nelle proprie pubblicazioni, il team di FALCON ha introdotto delle informazioni riguardo ai test effettuati. Sono riusciti ad eseguire FALCON su processori ARM, ad esempio il Cortex-M4, e hanno studiato come interfacciare FALCON con il protocollo TLS 1.3 [23].

Inoltre, come da direttive del NIST, dal secondo *round* nella documentazione sono incluse anche le vulnerabilità di sicurezza che i candidati stessi hanno rilevato durante lo sviluppo [23]. Il team di FALCON ha identificato vulnerabilità contro i seguenti attacchi attualmente conosciuti:

1. **Key Recovery**: l'attaccante cerca di ricostruire la chiave privata a partire da una serie di firme effettuate con tale chiave.

2. **Forgery**: l'attaccante riesce a generare una firma valida senza conoscere la chiave privata. Anche in questo caso l'attaccante deve entrare in possesso di diverse firme effettuate con tale chiave privata.

3.4 SPHINCS+

SPHINCS+ (*Stateless Practical Hash Based Incredibly Nice Cryptographic Signature*) è uno *signature scheme stateless Hash Based*, progettato per resistere agli attacchi quantistici. È stato il primo schema di questo tipo a essere proposto nel contesto del *NIST Post-Quantum Cryptography Standardization Process*[9], con l'obiettivo di garantire sicurezza anche contro attacchi criptoanalitici quantistici e mantenendo la resistenza classica basata su funzioni hash tradizionali [14].

3.4.1 Sicurezza e basi teoriche

SPHINCS+ rappresenta un'evoluzione significativa rispetto al suo predecessore SPHINCS. Una delle innovazioni chiave di SPHINCS+ è l'introduzione della struttura ad *hypertree*, che consente l'uso di un numero molto elevato di coppie di chiavi per le *few-time signature* (FTS) [25]. Questo permette a SPHINCS+ di mantenere la sicurezza anche in scenari dove è richiesto un gran numero di firme, migliorando la resistenza agli attacchi multi-target. Questi attacchi, che mirano a compromettere la sicurezza cercando di sfruttare la possibilità di generare molteplici firme con la stessa chiave, sono efficacemente mitigati grazie alla struttura e alle tecniche impiegate in SPHINCS+ [26].

Tra i principali vantaggi di SPHINCS+ vi è la sua semplicità teorica e la ridotta dimensione delle chiavi (sia pubbliche che private). Questo deriva dall'utilizzo di algoritmi basati su hash che introducono poche nuove assunzioni teoriche rispetto ad altri approcci come quelli *Lattice Based* [27]. Tuttavia, il compromesso è una firma di dimensioni significativamente maggiori e una velocità di firma e verifica più lenta rispetto ad altri candidati [28].

Un altro punto critico è la potenziale vulnerabilità di SPHINCS+ ai *side channel attacks* di tipo elettromagnetico (EM). Il team di SPHINCS+ ha riconosciuto questa

debolezza e sta esplorando metodi per mitigare tali attacchi modificando il modo in cui i valori privati vengono calcolati durante la generazione delle firme [25].

Attualmente, il team di SPHINCS+ non ha identificato attacchi specifici che possano sfruttare direttamente le strutture delle funzioni hash utilizzate per le operazioni di firma. Di conseguenza, le valutazioni di sicurezza di SPHINCS+ sono state basate principalmente su attacchi generici, che mirano a vulnerabilità comuni a tutte le implementazioni Hash Based. Questo approccio è stato considerato sufficiente, dato che molti studi hanno già confermato la robustezza degli algoritmi *Hash Based* [27].

3.4.2 Caratteristiche

Il design di SPHINCS+ è incentrato sulla resistenza agli attacchi quantistici, una proprietà che lo distingue da altri *signature schemes Hash Based*. Tuttavia, la scelta di implementare uno schema *stateless* comporta dei compromessi significativi. Infatti, rispetto a soluzioni *stateful*, SPHINCS+ presenta una dimensione della firma e un tempo di firma notevolmente maggiori. Questo incremento delle risorse necessarie per la firma deriva dalla necessità di mantenere l'integrità del sistema senza conservare uno stato tra una firma e l'altra [10].

Le prestazioni di SPHINCS+, in particolare per quanto riguarda il tempo di firma, sono generalmente inferiori rispetto agli altri candidati NIST, con differenze che possono arrivare fino a un ordine di grandezza. Nonostante ciò, SPHINCS+ offre potenzialità di ottimizzazione significativa, specialmente attraverso l'utilizzo di GPU o l'impiego di architetture multi-core per eseguire i calcoli in parallelo [28].

3.4.3 Versioni disponibili

SPHINCS+ offre implementazioni basate su tre diverse funzioni hash: SHA256, SHAKE256 e Haraka. Ognuna di queste implementazioni presenta caratteristiche uniche e livelli di ottimizzazione differenti. SHA256 e SHAKE256 sono le implementazioni più osservate e supportate, in parte perché Haraka, pur offrendo output di dimensioni inferiori, non raggiunge i livelli di sicurezza richiesti dal NIST [27].

Mentre con le prime due implementazioni si è in grado di raggiungere i livelli di sicurezza 1, 3 e 5, con l'implementazione Haraka lo stesso team di SPHINCS+ sostiene che attualmente non sia possibile superare il livello di sicurezza 2. Il NIST inoltre afferma che sono necessari ulteriori studi per la ricerca di vulnerabilità su questa versione, prima di un eventuale ammissione alla standardizzazione. SHA256 e SHAKE256 invece sono più standard, motivo per cui sono già state approvate [27].

Ogni implementazione si suddivide poi in due istanze, *simple* e *robust*, che danno origine a 6 combinazioni totali. Esse differiscono principalmente per il compromesso tra velocità e sicurezza. Le istanze *simple* offrono prestazioni più elevate, ma a scapito di un leggero decremento del livello di sicurezza rispetto alle istanze *robust* [27].

Anche se le versioni ottimizzate utilizzano istruzioni AVX2, simili a quelle utilizzate in altri schemi come Falcon e Dilithium, SPHINCS+ rimane comunque uno schema con prestazioni relativamente inferiori in termini di tempo di firma [10].

4

Metodologia

L'obiettivo di questo capitolo è presentare l'approccio implementativo utilizzato per eseguire il *performance testing* di ciascuno dei candidati selezionati, al fine di trarre ulteriori conclusioni rispetto ai report del NIST che dal terzo *round* hanno avviato le attività di standardizzazione.

4.1 Metriche di Performance

Le metriche di performance significative per gli obiettivi della ricerca sono:

1. **Key Sizes**: dimensione della chiave pubblica e della chiave privata. Spesso quella ritenuta più rilevante è la *public key* poiché viene solitamente inglobata nella firma della comunicazione, quindi necessita di una fase di trasferimento il cui tempo e costo deve essere minimizzato. La *private key* deve rimanere nei dispositivi del firmatario, dunque la sua lunghezza ha meno valore in termini di costi associati all'intero sistema di firma.

2. **Signature Sizes**: sono molto rilevanti anche le dimensioni delle firme. Quando inglobate in dei pacchetti, la loro dimensione influenza i tempi di comunicazione associati al loro trasferimento da mittente a destinatario. Inoltre, le firme possono essere rilevanti anche da un punto di vista di sicurezza. Firme di lunghezza variabile potrebbero implicare vulnerabilità ai *side channel attacks*.

3. **Signature Time**: il tempo richiesto all'algoritmo per firmare un determinato messaggio con la chiave privata. L'output del processo di firma è la produzione della firma stessa.

4. Validation Time: il tempo necessario all'algoritmo per verificare se una firma è stata generata dalla chiave privata corretta, cioè per verificare l'autenticità ed integrità del messaggio.

5. **Key Generation Time**: questo tempo misura la generazione della coppi di chiavi. Rispetto ai precedenti due tempi è considerato meno rilevante poiché tale procedura è eseguita un numero inferiore di volte rispetto alla firma e verifica delle comunicazione. Una volta generata una coppia di chiavi, la cui validità può durare anni, si possono eseguire un elevato numero di operazioni di firma e verifica.

Oltre alle metriche sopra elencate, altre caratteristiche non misurate direttamente, ma rilevanti per l'analisi, includono:

1. **Occupazione in memoria**: in certi casi, come per lo sviluppo di algoritmi compatibili con dispositivi IoT (Internet of Things), è necessario monitorare le dimensioni di RAM e ROM necessarie all'esecuzione dell'algoritmo a causa di risorse hardware limitate. L'esecuzione di algoritmi di firma è fondamentale anche nei dispositivi IoT poiché essi interagiscono con altre risorse on-line o nella stessa rete privata, dovendo quindi utilizzare protocolli come TLS per verificare l'autenticità e integrità delle trasmissioni [10].

2. **Costi di trasmissione**: anche se difficili da misurare direttamente a causa delle variazioni nella connessione di rete, è importante comprendere le dimensioni totali delle comunicazioni per ottimizzarle [10].

3. **Natura dell'algoritmo**: gli obiettivi secondari degli algoritmi PQC non sono tutti uguali. Ciascun team di ricerca ha dato importanza a diversi aspetti dell'ottimizzazione. Alcuni si sono concentrati sulle procedure *constant time*, tendenzialmente più lente ma sicure, altri ad implementazioni a basso utilizzo di memoria, altri alla minimizzazione delle dimensioni degli output.

4.2 Ambiente di Sviluppo

In questa sezione viene descritto l'ambiente di sviluppo utilizzato per la realizzazione dei test, includendo i linguaggi di programmazione, gli strumenti, e l'hardware impiegato.

4.2.1 Linguaggi di Programmazione

Per lo sviluppo del *performance testing* sono stati utilizzati due linguaggi:

- C: Utilizzato per la compilazione dei codici sorgente degli schemi di firma digitale e per la creazione ed esecuzione dei test di performance.

- Python: Impiegato per la fase di raccolta e rappresentazione dei dati di output dei vari test di performance.

4.2.2 Sistema Operativo

Il sistema operativo utilizzato per lo sviluppo è *Ubuntu 22.04.4* in esecuzione su *Windows Subsystem for Linux (WSL)*. Ubuntu è stato fondamentale poiché è un ambiente adatto alla compilazione ed esecuzione di codice sorgente C, allo stesso tempo Windows ha supportato la fase di ricerca e di visualizzazione dei grafici ottenuti.

4.2.3 Versionamento del Codice

Per la gestione del codice sorgente è stato utilizzato *Git*, con repository ospitati su *GitHub*. Il suo utilizzo è stato fondamentale per la gestione del versioning: il controllo delle modifiche e delle revisioni del codice.

4.2.4 Hardware Utilizzato

Lo sviluppo e l'esecuzione dei test è stato eseguito su macchine dotate delle seguenti specifiche hardware:

	PC #1	**PC #2**
Modello	Samsung Book 4 Ultra	Dell Inspiron 5406 2n1
CPU	Intel Ultra U9 185H	Intel Core i7-1165G7
Numero di Core/Thread	12 Core / 24 Thread	4 Core / 8 Thread
Frequenza Max CPU	5.1 GHz	4.7 GHz
Hyper-Threading Attivo	Sì	Sì
Virtualizzazione Attiva	Sì	Sì
Cache L3	24 MB	12 MB
GPU	NVIDIA RTX 4070	Integrated Intel Iris Xe
RAM	32 GB DDR5	16 GB DDR4

Tabella 4.1: Configurazioni hardware dei PC utilizzati per lo sviluppo e il testing

Il PC #1 è stato utilizzato per lo sviluppo e il testing. Considerare solo i test eseguiti su questa macchina, in particolare i tempi di esecuzione, può risultare fuorviante a causa della potenza computazionale sopra alla media. Per tale motivo i test sono stati ripetuti anche nel PC #2, dalle prestazioni più limitate.

4.2.5 Strumenti di sviluppo

L'ambiente di sviluppo integrato (IDE) utilizzato per la scrittura e il debugging del codice è *Visual Studio Code*, scelto per la sua leggerezza, estensibilità e il supporto avanzato per l'integrazione con sistemi di sviluppo remoto. Per la compilazione del codice C è stato utilizzato *gcc 11.4.0*, mentre per l'esecuzione del codice Python è stato impiegato l'interprete *Python 3.12*, entrambi in esecuzione su Ubuntu all'interno di WSL. Inoltre, Visual Studio Code è stato configurato con l'estensione per l'integrazione di WSL, facilitando lo sviluppo e il debug direttamente all'interno dell'ambiente Ubuntu.

4.3 Implementazione

Prima di procedere all'analisi, in questa sezione vengono presentate più a fondo le modalità di testing dal punto di vista dello sviluppo.

4.3.1 Sorgenti del codice

Come prima fase è stato necessario ottenere il codice sorgente C dei vari schemi di firma digitale (in tutte le loro versioni) da una fonte ufficiale, in particolare l'interesse era rivolto alle pubblicazioni che hanno partecipato al terzo *round* del *NIST Post-Quantum Cryptography Standardization Process* [9].

Di seguito una lista delle sorgenti per ciascun candidato:

- CRYSTALS-Dilithium: ottenuto direttamente dal repository GitHub del progetto. https://github.com/pq-crystals/dilithium [29]
- FALCON: il codice sorgente è disponibile nel sito dedicato. https://falcon-sign.info/impl/falcon.h.html [30]
- SPHINCS+: ottenuto dal repository GitHub del progetto. https://github.com/sphincs/sphincsplus [31]
- RSA: per i test su RSA verranno utilizzate le libreria del sistema operativo per OpenSSL.

Il codice sviluppato per l'esecuzione dei test è disponibile in un repository pubblico su GitHub: https://github.com/LovatoTomas/Analisi_di_algoritmi_di_firma_digitale_post-quantum [32]

4.3.2 Obiettivi e struttura del codice

Per l'implementazione dei performance test sono stati utilizzati come base gli script di test scritti dai team di ricerca per ciascuna delle loro implementazioni. Tuttavia, i loro erano dei *validation test*, ovvero script il cui scopo era simulare il naturale funzionamento dello schema di firma e di verificare che ciascun modulo riportasse risultati corretti.

Ad esempio, dopo la firma di un messaggio fisso con una specifica chiave privata, alcuni di questi *validation test* controllavano che:

- la firma fosse valida con la chiave pubblica corretta: in questo caso il test di verifica doveva ritornare esito positivo.
- la firma fosse valida con la chiave pubblica corretta ma con l'alterazione del messaggio originale oppure della firma di esso: in questo caso l'esito doveva essere negativo.
- la firma fosse valida con una chiave pubblica differente: anche in questo caso l'esito doveva essere negativo.

Per evitare di modificare la struttura dei singoli repository e le configurazioni di compilazione, i *performance test* sono stati implementati come modifica dei file di *validation test*. Sono stati rimosse tutte le prove superflue, lasciando la struttura principale del processo di firma:

1. Generazione della coppia di chiavi
2. Firma di un messaggio con la chiave privata
3. Verifica della firma con la chiave pubblica

Partendo da questa base comune è stato introdotto il codice per:

1. Misurare i tempi di ciascuna operazione
2. Misurare le dimensioni delle chiavi e delle firme
3. Esportare in formato *standard* tutti i risultati dei test

Per rendere la misura dei tempi più precisa, ciascuna operazione viene eseguita ripetutamente un numero elevato di volte, successivamente i tempi raccolti da ciascuna iterazione vengono mediati per ottenere una stima più fine del tempo trascorso. Con la tecnica utilizzata, la precisione nella misurazione dei tempi può arrivare ai nanosecondi.

Per rendere ancor più generico il contesto di misurazione dei tempi, non è stato utilizzato un messaggio fisso per le operazioni di firma e verifica ma ad ogni ripetizione del flusso

firma e verifica viene generato un nuovo messaggio casuale. Generando dinamicamente il messaggio sono state sviluppate due ulteriori classi di test per confrontare tra loro i vari schemi di firma. La prima modalità potrebbe essere definita come *stress test* poiché i messaggi generati hanno una lunghezza incrementale, partendo da stringhe di decine di bytes fino ad arrivare a milioni di caratteri. Questi test hanno un significato limitato: permettono di comprendere come i vari algoritmi di firma e verifica si comportano dal punto di vista dei tempi a dei messaggi di lunghezza sempre maggiore. Ad esempio, potrebbero evidenziare la differenza tra firmare il testo di una mail corta e firmare un intero documento PDF. Tuttavia, negli attuali sistemi di firma digitale il processo di firma e verifica non viene mai applicato direttamente al messaggio in sé, ma ad una sua versione ridotta ottenuta tramite funzioni hash.

Per questo motivo, la seconda classe di test prevede di effettuare l'hashing del messaggio generato e di misurare i tempi di firma e verifica non sul messaggio stesso ma sul suo *hashcode*. Così facendo si ottengono dei risultati più simili alla reale implementazione di questi sistemi qualora vengano standardizzati.

Ovviamente durante la fase di analisi verrà data maggiore rilevanza a questa seconda classe di test, mentre i test eseguiti direttamente sul messaggio di lunghezza incrementale saranno utilizzati per fare brevi note, qualora venissero assunti comportamenti particolari.

Per effettuare l'hashing dei messaggi è stata installata la libreria *OpenSSL* nel sistema operativo Ubuntu, successivamente nel codice C dei vari test è stata inclusa tale libreria per utilizzarne le seguenti funzioni:

```
// Includere la libreria OpenSSL per l'hashing
#include <openssl/sha.h>
// Funzione per calcolare l'hash SHA256, parametri:
// - const unsigned char *data: puntatore al messaggio da hashare.
// - size_t len: lunghezza del messaggio in byte.
// - unsigned char *out: puntatore al buffer per l'hash (32 byte).
unsigned char *SHA256(const unsigned char *data, size_t len, unsigned
↪  char *out);
// Funzione per calcolare l'hash SHA512, parametri:
// - const unsigned char *data: puntatore al messaggio da hashare.
// - size_t len: lunghezza del messaggio in byte.
// - unsigned char *out: puntatore al buffer per l'hash (64 byte).
unsigned char *SHA512(const unsigned char *data, size_t len, unsigned
↪  char *out);
```

Al termine dei test, gli script salvano i risultati in un file di testo organizzato per record strutturati tramite caratteri separatori. La struttura dei record in output è riportata nella sezione di codice successiva.

```
|MLEN|MTOTLEN|PUBLEN|PRVLEN|SIGLEN|KGTM|SIGTM|CHECKTM|HASHSZ|

// Descrizione dei campi:
// MLEN = Lunghezza in byte del messaggio in input
// MTOTLEN = Lunghezza in byte messaggio + firma del messaggio
// PUBLEN = Lunghezza in byte della chiave pubblica generata
// PRVLEN = Lunghezza in byte della chiave privata generata
// SIGLEN = Lunghezza in byte della firma del messaggio
// KGTM = Tempo di generazione chiavi in nanosecondi
// SIGTM = Tempo di firma in nanosecondi
// CHECKTM = Tempo di verifica in nanosecondi
// HASHSZ = Dimensione dell'hashcode del messaggio
```

Per gestire in maniera centralizzata la configurazione e l'avvio dei test per ogni tipologia di schema di firma, è stato creato uno script centralizzato (chiamato `start_analytics.c`) il cui scopo è avviare i comandi di compilazione dei singoli progetti e successivamente avviare i performance test per ogni versione degli schemi di firma, tutti con gli stessi parametri (numero di iterazioni delle operazioni). Di seguito un estratto di tali comandi:

```
// Compilazione dell'implementazione di FALCON AVX2
make -C ./FALCON/avx2/
// Esecuzione dei test su FACLON con hashing del messaggio con SHA-256
// - FILE_OUTPUT_1 = nome file dei risultati di FALCON 512 AVX2
// - FILE_OUTPUT_2 = nome file dei risultati di FALCON 1024 AVX2
// - NUM_IT = numero di iterazioni su cui mediare i tempi
// - INCR = fattore di incremento della lunghezza del messaggio generato
./FALCON/avx2/test_falcon_sha256 FILE_OUTPUT_1 FILE_OUTPUT_2 NUM_IT INCR
```

4.3.3 Generazione dei grafici

Al termine della fase di test, viene avviato un ultimo script python il cui scopo è leggere tutti i file di testo in output e generare i grafici che confrontano tra loro versioni dello

stesso schema oppure tra loro schemi diversi. Per la generazione dei grafici in output è stata utilizzata la libreria *matplotlib*.

Sono state create diverse tipologie di grafici, ciascuna per intrecciare sorgenti dati differenti e visualizzarle nella modalità che più si adatta al contesto. La tabella 4.2 presenta l'insieme dei grafici generati nella cartella `plot` del repository GitHub [32].

Input dati	**Tipologia**	**Descrizione**
Dimensione chiavi	Istogramma	Per ogni algoritmo evidenzia la dimensione della chiave privata e della chiave pubblica e le mette in scala con quelle degli altri algoritmi candidati
Dimensione firma	Istogramma	In questo caso, per ogni algoritmo è presente una sola colonna che rappresenta la dimensione (in bytes) della firma del messaggio
Tempistiche di keygen	Istogramma	Ogni algoritmo (definito per tipologia e metodo di implementazione) vengono mostrati i tempi di generazione delle chiavi e comparate con quelle di altri algoritmi
Tempistiche di firma	Istogramma e diagramma a linee	Vengono messi in relazione i tempi di firma degli algoritmi, differenziati per famiglia, implementazione e messaggio in input (se a lunghezza variabile o fissa)
Tempistiche di verifica	Istogramma e diagramma a linee	Comparazione dei tempi di verifica degli algoritmi analizzati, differenziati per famiglia, implementazione e messaggio in input (se a lunghezza variabile o fissa)

Tabella 4.2: Tipologie di grafici generati a partire dai dati raccolti

Nel prossimo capitolo sarà possibile visualizzare i grafici di ciascuna delle categorie presentate nella tabella 4.2. Ciascun diagramma coinvolge un insieme di circa 3 o 4 algoritmi. Solitamente le analisi vengono effettuate:

1. Tra le differenti versioni di una stessa famiglia di algoritmi, ad esempio CRYSTALS Dilithium 2, CRYSTALS Dilithium 3 e CRYSTALS Dilithium 5.

2. Tra differenti implementazioni di uno stesso algoritmo, ad esempio le implementazioni *REF* e *AVX2* di CRYSTALS Dilithium 2.

3. Tra le stesse implementazioni di diverse famiglie di algoritmi. Questi insiemi di diagrammi sono i più utili per poter confrontare direttamente i candidati tra loro.

Inoltre, i diagrammi sono stati generati per ogni hardware di test: per il PC#1 le immagini si trovano nella cartella `/plot/20240822_i9/` mentre per il PC#2 è stato utilizzato il percorso `/plot/20240822_i7/`. Le due cartelle contengono rispettivamente i grafici i cui dati in input sono esclusivamente gli output dei *performance test* su quello specifico hardware. Il responsabile della generazione di questi plots è lo script `start_performancegraphs.py`.

Per mettere a confronto i comportamenti dei vari algoritmi, se eseguiti in hardware differenti, è stato sviluppato lo script `start_comparisongraphs.py`: nella cartella `/plot/comparison/` del repository sono contenuti i grafici che evidenziano la differenza di prestazioni di uno stesso algoritmo se eseguito in macchine differenti.

Di seguito viene proposto un estratto del codice python necessario alla generazione del diagramma a linee per i tempi di generazione delle chiavi. Questa versione del grafico è specializzata nella comparazione di algoritmi eseguiti in diverso hardware.

```python
# Definizione delle librerie utilizzate
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

# Gruppi di file di output da confrontare + nome e titolo del grafico
file_groups = [ ... ]
# Dizionario per mappare i nomi degli algoritmi a nomi user-friendly
algorithm_name_mapping = { ... }
# Dizionario per mappare ciascun algoritmo ad un colore fisso
plot_colors = { ... }

# Per ogni insieme / gruppo di algoritmi va generato un grafico:
for file_paths in file_groups:
    # Lettura del gruppo di files di output (uno per algoritmo)
    df_all = read_multiple_files(file_paths[0])
    # Mapping user-friendly del nome degli algoritmi
    df_all['algorithm'] = df_all['algorithm'].map(algorithm_name_mapping)
    # Creazione del grafico con la media dei tempi di keygen
    create_keygen_time_histogram(df_all, './plot/comparison/Time_Keygen/'
    ↪  + file_paths[1], file_paths[2])

# Funzione per leggere e unire i dati da più file (confronto algoritmi)
# Aggiunge anche colonne di metadati sull'esecuzione dell'algoritmo
def read_multiple_files(filenames):
    dataframes = [read_data(file) for file in filenames]
    for df, file in zip(dataframes, filenames):
```

```
28          df['algorithm'] = file.split('/')[3]  # estrai il nome
            ↪ dell'algoritmo dal nome del file
29          # Aggiunta colonna per distinguere tra REF e AVX2
30          if 'avx2' in file:
31              df['version'] = 'AVX2'
32          else:
33              df['version'] = 'REF'
34          # Aggiunta colonna per distinguere tra PC#1 e PC#2
35          if 'i9' in file:
36              df['device'] = 'PC#1 i9 - AVX2'
37          else:
38              df['device'] = 'PC#2 i7 - AVX2'
39      return pd.concat(dataframes, ignore_index=True)
40
41  # Funzione per leggere i dati dal file
42  def read_data(filename):
43      # Apertura e lettura file di output
44      with open(filename, 'r') as file:
45          lines = file.readlines()
46      # Tokenizzazione dei dati in "tabelle"
47      data = []
48      for line in lines:
49          line = line.strip()
50          # Divisione dei dati in varie colonne, in base allo standard di
            ↪ output
51          if line:
52              fields = line.split('|')
53              data.append([int(fields[1]), int(fields[2]), int(fields[3]),
                ↪ int(fields[4]), int(fields[5]), float(fields[6]),
                ↪ float(fields[7]), float(fields[8]), int(fields[9])])
54      # Impostazione nome delle colonne
55      columns = ['msg_len', 'signed_msg_len', 'pub_key_size',
        ↪ 'priv_key_size', 'signature_size', 'keygen_time', 'sign_time',
        ↪ 'verify_time', 'hash_len']
56      df = pd.DataFrame(data, columns=columns)
57      return df
58
```

```
59 # Funzione per creare un istogramma per la media dei tempi di keygen
60 def create_keygen_time_histogram(df_all, output_file, title):
61     # Calcola la media dei tempi di keygen per ogni algoritmo e versione
62     df_mean_keygen_time = df_all.groupby(['algorithm',
       ↪  'device'])['keygen_time'].mean().reset_index()
63     # Forzare l'ordine degli algoritmi come appare nell'elenco di input
64     df_mean_keygen_time['algorithm'] =
       ↪  pd.Categorical(df_mean_keygen_time['algorithm'],
       ↪  categories=df_all['algorithm'].unique(), ordered=True)
65     # Crea un grafico a barre (istogramma)
66     plt.figure(12,4)
67     sns.barplot(x='algorithm', y='keygen_time', hue='device',
       ↪  data=df_mean_keygen_time, palette=plot_colors)
68     # Configurazione etichette e legenda
69     plt.xlabel('Algorithms and Versions')
70     plt.ylabel('Average Keygen Time (seconds)')
71     plt.title('Device Comparison - Average Keygen Time - ' + title)
72     plt.legend(title='Device')
73     # L'asse y utilizza una scala logaritmica
74     plt.yscale('log')
75     # Salvataggio del grafico
76     save_or_show_plot(nome_output_file)
```

Per la generazione delle altre tipologie di grafici vengono definite funzioni molto similari, che variano principalmente per i dati presi in esame durante la creazione del `plot`. Le funzioni di lettura dei dati da file di testo, di mapping dei nomi e di salvataggio delle immagini rimangono in comune al codice specializzato nella produzione dei diagrammi. Il codice completo è disponibile nel repository del progetto [32].

4.3.4 Verifica dei risultati

Per verificare l'affidabilità dei risultati ottenuti implementando i test direttamente sui codici sorgenti degli schemi di firma digitale, sono stati effettuati dei test di verifica su un ulteriore libreria open-source: *LibOQS* [33].

LibOQS è una libreria in linguaggio C sviluppata dal progetto Open Quantum Safe (OQS) che fornisce implementazioni di algoritmi crittografici resistenti agli attacchi da parte di computer quantistici. Il vantaggio di utilizzare *LibOQS* rispetto ad un implementazione diretta degli schemi di firma è che *LibOQS* si comporta come un *wrapper* su

tutti gli algoritmi che ingloba, ovvero fornisce un interfaccia comune con cui utilizzarli tutti alla stessa maniera. Per tale motivo l'implementazione degli stessi *performance test* è risultata semplificata poiché l'algoritmo di test è stato unico per tutti gli schemi provati.

Il processo di verifica ha avuto successo: nella maggior parte dei casi i risultati ottenuti sono compatibili con quelli ottenuti dalle prove dirette. I casi che portano differenze rilevanti verranno discusse nel prossimo capitolo. Tali risultati sono disponibili, oltre che in forma testuale (file di output), anche in forma visiva. Difatti, le categorie di grafici elencate nella tabella 4.2 sono state utilizzate anche per mettere a confronto le singole caratteristiche degli algoritmi eseguiti con l'implementazione diretta, prodotta ad hoc, e con l'implementazione di *LibOQS*. Alcuni di questi grafici sono inclusi nel successivo capitolo.

4.3.5 Pseudocodice dei test

In questa sezione viene riportato lo pseudocodice utilizzato per lo svolgimento dei *performance test*. Essendo una scrittura ad alto livello, essa è compatibile con gli script scritti per tutti i sistemi di firma digitale considerati (CRYSTALS Dilithium, FALCON e SPHINCS+) e anche per i test effettuati tramite la libreria *LibOQS*.

Il risultato della sua esecuzione è la creazione di file di output contenenti i dati sulle prestazioni dei vari algoritmi analizzati, per ogni loro versione eseguita.

Algorithm 1 Test delle Prestazioni degli Schemi di Firma Post-Quantum

Input: ITER (Numero di ripetizioni per ogni configurazione, valore di default 100), OUT (Nome del file di output), HASHING (Booleano per attivare/disattivare l'hashing del messaggio), HASH (Algoritmo di hashing, "sha256" o "sha512")
Output: File di output con i risultati dei test per ogni configurazione del messaggio
Inizializza le variabili locali:
$keypair \leftarrow None$
$message \leftarrow None$
$signature \leftarrow None$
$gen\_times \leftarrow []$ //Array dei tempi (da mediare) per la generazione delle chiavi
$sign\_times \leftarrow []$ //Array dei tempi (da mediare) per la firma del messaggio
$verify\_times \leftarrow []$ //Array dei tempi (da mediare) per la verifica della firma
file.**open**(OUT, `'w'`)
for lunghezza del messaggio $msg\_len = 64$ B **to** 16 MB con step di $msg\_len \times 2$ **do**
    for $i = 1$ **to** ITER **do**
        Memorizza il tempo di generazione chiavi e lo inserisce nell'array:
        $start\_timer \leftarrow$ time.now()
        $keypair \leftarrow$ pqc_keygen()

16: $end\_timer \leftarrow$ time.now()
17: gen_times.append($end\_timer - start\_timer$)
18: **end for**
19: Memorizza la dimensione finale delle chiavi:
20: $pub\_key\_len \leftarrow$ len($keypair.public\_key$)
21: $priv\_key\_len \leftarrow$ len($keypair.private\_key$)
22: Calcola il tempo medio di keygen:
23: $avg\_gen\_time \leftarrow$ sum(gen_times)$/ITER$
24: Misurazione delle performance di firma e verifica
25: **for** $i = 1$ **to** ITER **do**
26: Genera un messaggio random di lunghezza msg_len:
27: $message \leftarrow$ generate_random_message(msg_len)
28: Esegue l'hashing del messaggio se la prova corrente lo prevede:
29: **if** HASHING è attivo **then**
30: **if** HASH = "sha256" **then**
31: $message \leftarrow$ sha256($message$)
32: **else if** HASH = "sha512" **then**
33: $message \leftarrow$ sha512($message$)
34: **end if**
35: **end if**
36: Memorizzo la lunghezza dell'attuale messaggio (o hashcode):
37: $message\_len \leftarrow$ len($message$)
38: $start\_timer \leftarrow$ time.now()
39: Firma il messaggio:
40: $signature \leftarrow$ pqc_sign($keypair.private\_key, message$)
41: $end\_timer \leftarrow$ time.now()
42: $sign\_times$.append($end\_timer - start\_timer$)
43: $start\_timer \leftarrow$ time.now()
44: Verifica la firma:
45: $valid \leftarrow$ pqc_verify($keypair.public\_key, signature, message$)
46: $end\_timer \leftarrow$ time.now()
47: $verify\_times$.append($end\_timer - start\_timer$)
48: **end for**
49: Memorizza la dimensione della firma:
50: $signature\_len \leftarrow$ len($signature$)
51: Calcola i tempi medi di firma e verifica:
52: $avg\_sign\_time \leftarrow$ sum($sign\_times$)$/ITER$
53: $avg\_verify\_time \leftarrow$ sum($verify\_times$)$/ITER$

54: Scrivi i risultati nel file *OUT* e stampa a schermo:
55: *out_res* ← **format_output_record**(*msg_len*, *msg_len+signature_len*, *pub_key_len*, *priv_key_len*, *signature_len*, *avg_gen_time*, *avg_sign_time*, *avg_verify_time*, *message_len*)
56: **print**(*out_res*)
57: file.**write**(OUT, *out_res*)
58: **end for**
59: file.**close**(OUT)

Di seguito viene proposto uno dei file di output generati dall'algoritmo 1. In particolare questi risultati sono legati all'esecuzione di CRYSTALS Dilithium 2, in versione *AVX2*, senza *hashing* del messaggio.

```
|MLEN|MTOTLEN|PUBLEN|PRVLEN|SIGLEN|KGTM|SIGTM|CHECKTM|HASHSZ|
|32|2452|1312|2528|2420|0.000025|0.000117|0.000019|32|
|64|2484|1312|2528|2420|0.000017|0.000049|0.000018|64|
|128|2548|1312|2528|2420|0.000020|0.000064|0.000029|128|
|256|2676|1312|2528|2420|0.000018|0.000064|0.000021|256|
|512|2932|1312|2528|2420|0.000019|0.000060|0.000021|512|
|1024|3444|1312|2528|2420|0.000018|0.000056|0.000022|1024|
|2048|4468|1312|2528|2420|0.000021|0.000059|0.000023|2048|
|4096|6516|1312|2528|2420|0.000018|0.000100|0.000028|4096|
|8192|10612|1312|2528|2420|0.000018|0.000177|0.000036|8192|
|16384|18804|1312|2528|2420|0.000019|0.000088|0.000073|16384|
|32768|35188|1312|2528|2420|0.000025|0.000116|0.000108|32768|
|65536|67956|1312|2528|2420|0.000020|0.000211|0.000176|65536|
|131072|133492|1312|2528|2420|0.000019|0.000334|0.000330|131072|
|262144|264564|1312|2528|2420|0.000031|0.000613|0.000677|262144|
|524288|526708|1312|2528|2420|0.000023|0.001232|0.001293|524288|
|1048576|1050996|1312|2528|2420|0.000027|0.002656|0.002760|1048576|
|2097152|2099572|1312|2528|2420|0.000035|0.005268|0.005662|2097152|
|4194304|4196724|1312|2528|2420|0.000039|0.010812|0.011078|4194304|
|8388608|8391028|1312|2528|2420|0.000037|0.021413|0.021896|8388608|
|16777216|16779636|1312|2528|2420|0.000041|0.043897|0.044748|16777216|
```

Attivando la funzionalità di *hashing* con *SHA-256* invece, i risultati delle prestazione temporali cambiano radicalmente e l'ultimo parametro di ciascun output viene adattato al contesto.

La sezione successiva si riferisce sempre a CRYSTALS Dilithium 2, in versione *AVX2*, per semplicità di comparazione.

MLEN	MTOTLEN	PUBLEN	PRVLEN	SIGLEN	KGTM	SIGTM	CHECKTM	HASHSZ
32	2452	1312	2528	2420	0.000020	0.000054	0.000022	32
64	2452	1312	2528	2420	0.000028	0.000067	0.000027	32
128	2452	1312	2528	2420	0.000020	0.000051	0.000021	32
256	2452	1312	2528	2420	0.000025	0.000048	0.000022	32
512	2452	1312	2528	2420	0.000025	0.000049	0.000022	32
1024	2452	1312	2528	2420	0.000026	0.000052	0.000018	32
2048	2452	1312	2528	2420	0.000021	0.000051	0.000023	32
4096	2452	1312	2528	2420	0.000018	0.000048	0.000018	32
8192	2452	1312	2528	2420	0.000019	0.000055	0.000021	32
16384	2452	1312	2528	2420	0.000019	0.000051	0.000019	32
32768	2452	1312	2528	2420	0.000020	0.000056	0.000018	32
65536	2452	1312	2528	2420	0.000021	0.000056	0.000019	32
131072	2452	1312	2528	2420	0.000019	0.000054	0.000021	32
262144	2452	1312	2528	2420	0.000024	0.000058	0.000021	32
524288	2452	1312	2528	2420	0.000023	0.000054	0.000020	32
1048576	2452	1312	2528	2420	0.000026	0.000063	0.000021	32
2097152	2452	1312	2528	2420	0.000032	0.000089	0.000022	32
4194304	2452	1312	2528	2420	0.000035	0.000065	0.000027	32
8388608	2452	1312	2528	2420	0.000035	0.000057	0.000020	32
16777216	2452	1312	2528	2420	0.000038	0.000055	0.000025	32

I grafici generati a partire da tali files di output verranno introdotti nel prossimo capitolo.

5

Risultati e Discussioni

In questo capitolo verranno presentati i grafici ottenuti dai *performance test* eseguiti sugli algoritmi in standardizzazione. Ad ogni grafico seguirà una breve argomentazione, se necessaria. Per praticità vengono allegati solo i grafici più significativi, tuttavia nel repository GitHub del progetto [32] sono disponibili ulteriori immagini che mettono in relazione aspetti diversi. Il materiale è disponibile nella cartella *plot* del progetto.

La fase di analisi si concentrerà in particolare sulle versioni ottimizzate dei candidati, spesso definite con il termine *AVX2* poiché rappresenta una forma di ottimizzazione comune tra le varie implementazioni. Sono stati prodotti anche grafici che mettono in relazione le caratteristiche delle versioni *REF* e *AVX2*: spesso si rileva un aumento delle prestazioni (in termini di tempi di esecuzione) tra il 50% e il 100%.

5.1 CRYSTALS Dilithium

Le versioni di CRYSTALS Dilithium rilasciate per ogni implementazione sono Dilithium 2, Dilithium 3 e Dilithium 5. Come suggerisce il nome, esse sono legate ai livelli di sicurezza offerti, rispettivamente 2, 3 e 5.

Tra queste versioni le principali differenze sono legate ai parametri di generazione della coppia di chiavi e alla generazione della firma. Le dimensioni e tempistiche di ciascuna variante sono riassunte nella tabella 5.1.

In generale, le dimensioni delle chiavi e delle firme sono fisse a parità di versione scelta, questo garantisce una maggior resistenza ai *side channel attacks* e una maggior facilità di standardizzazione.

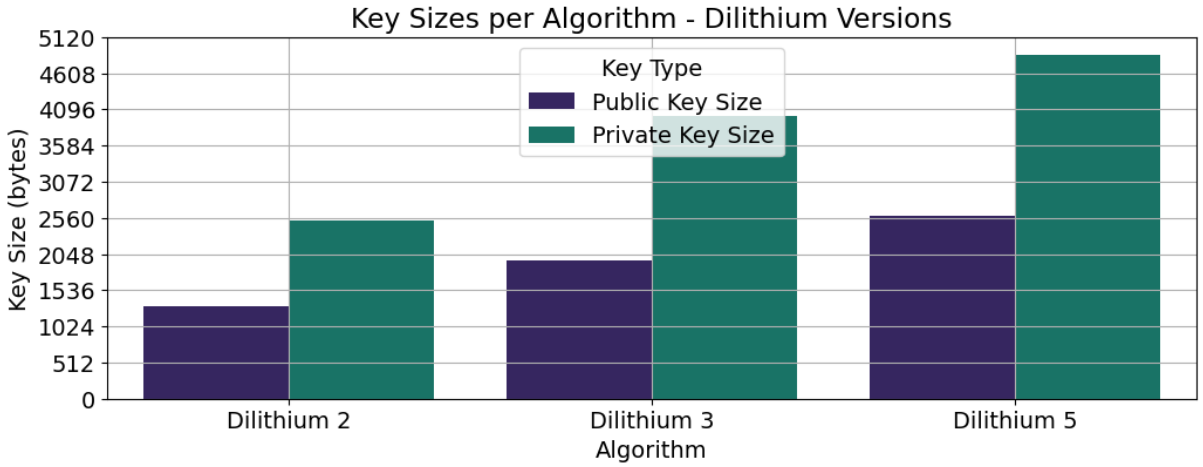

Figura 5.1: Dimensione delle chiavi private e pubbliche per le versioni di CRYSTALS Dilithium

Il grafico 5.1 evidenzia la differenza di dimensioni tra le chiavi pubbliche e le chiavi private. Inoltre, ogni gruppo di colonne rappresenta una delle versioni di Dilithium: si evince che la dimensione delle chiavi cresce con andamento lineare al livello di sicurezza offerto.

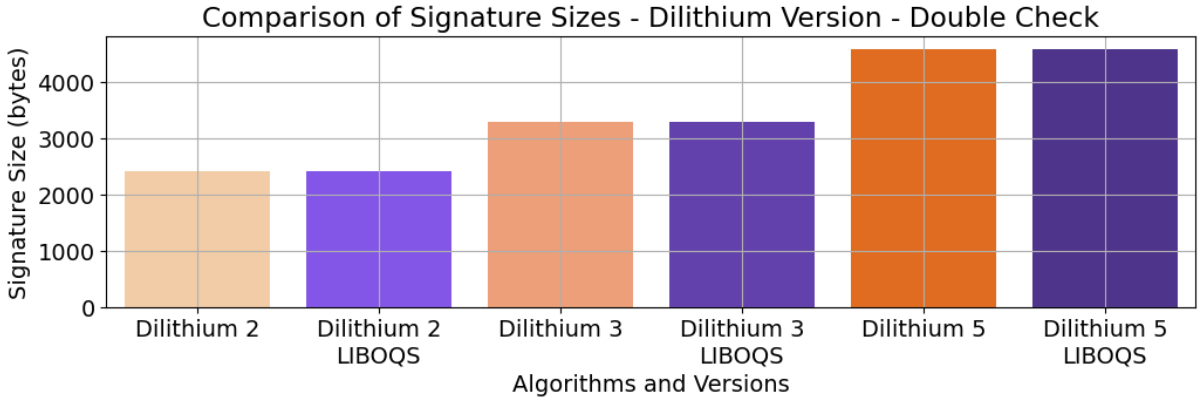

Figura 5.2: Confronto delle dimensioni delle firme tra l'implementazione diretta e l'implementazione LibOQS per CRYSTALS Dilithium

Il grafico 5.2 mostra uno dei test di verifica tra l'implementazione diretta e l'implementazione tramite libreria *LibOQS*. Si può notare come i dati coincidano per ogni versione inoltre, come per le chiavi anche la dimensione della firma sembra aumentare in dimensione con andamento lineare al livello di sicurezza.

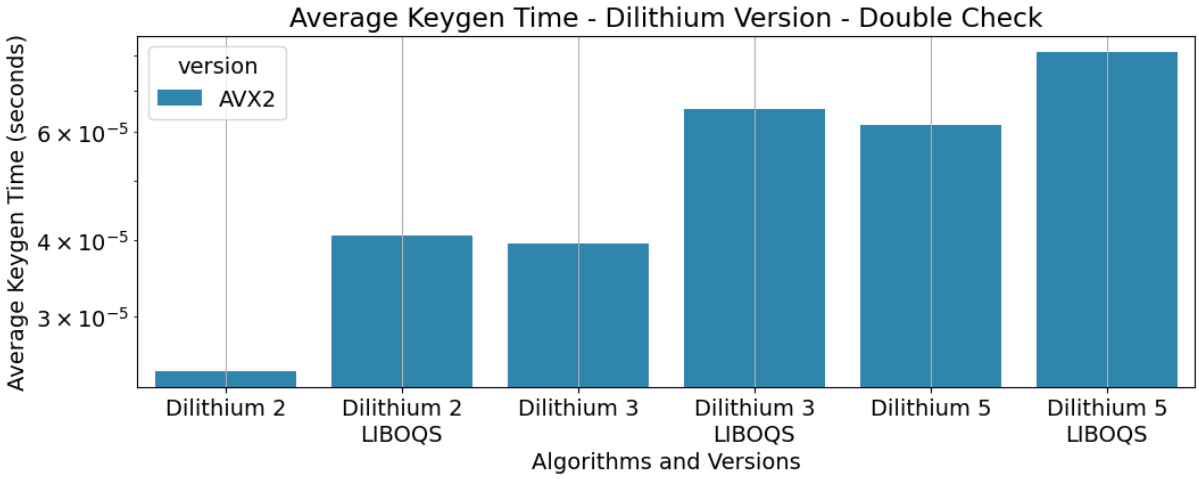

Figura 5.3: Confronto dei tempi di generazione chiavi tra l'implementazione diretta e l'implementazione LibOQS per CRYSTALS Dilithium

I tempi di *keygen* nel grafico 5.3 mostrano una maggior rapidità di esecuzione nell'implementazione diretta. Probabilmente questo accade perché la libreria *LibOQS* è un *wrapper* degli algoritmi originali, dunque come tale aggiunge dell'*overhead*. In ogni caso, le tempistiche si trovano sullo stesso ordine di grandezza, dunque le differenze non sono significative.

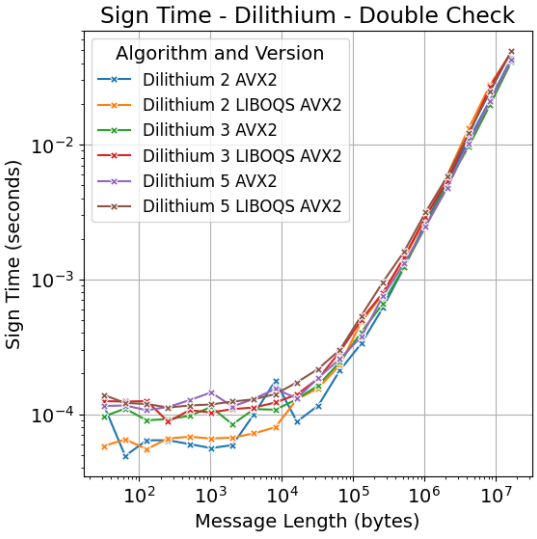

Figura 5.4: Controllo dei tempi di firma con LibOQS per CRYSTALS Dilithium con messaggi di lunghezza variabile

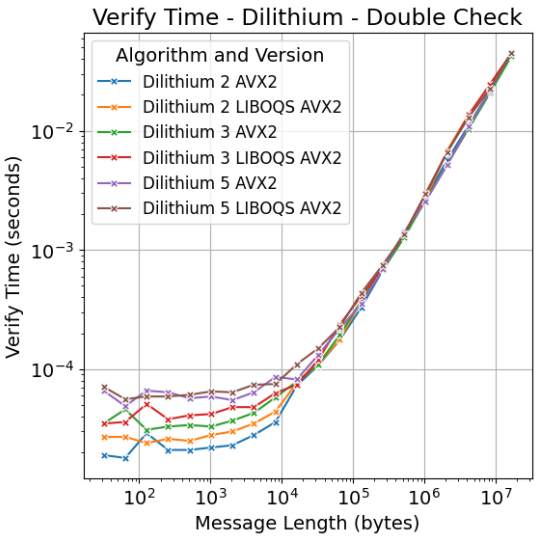

Figura 5.5: Controllo dei tempi di verifica con LibOQS per CRYSTALS Dilithium con messaggi di lunghezza variabile

Il grafico 5.4 e 5.5 evidenziano come CRYSTALS Dilithium comporti un aumento dei tempi esponenziale (l'asse dei tempi ha scala logaritmica) quando la dimensione del messaggio da firmare o verificare cresce oltre l'ordine dei KiloBytes.

In ogni caso, utilizzando operazioni piuttosto efficienti (tra vettori e matrici) i tempi rimangono comunque ridotti anche in caso di messaggi nell'ordine dei MegaBytes.

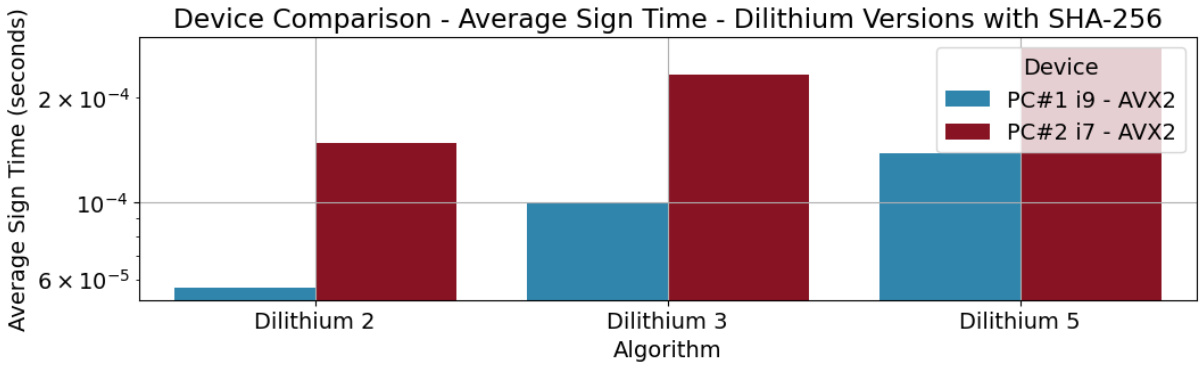

Figura 5.6: Tempi di firma con messaggi di lunghezza fissa su PC#1 e PC#2 per CRYSTALS Dilithium

Il grafico 5.6 è più significativo del precedente per i tempi di firma poiché considera il comune scenario in cui il messaggio subisce un processo di *hashing* prima della creazione della firma. In questo caso l'*hashing* è stato effettuato con i sistemi attualmente più utilizzati, cioè *SHA-256* e *SHA-512*. Con entrambe queste tecniche i tempi di firma hanno esecuzioni più rapide sui dispositivi utilizzati per la fase di test. In entrambi i dispositivi di test i tempi di firma sono direttamente proporzionali al livello di sicurezza offerto dall'algoritmo utilizzato: algoritmi più sicuri implicano tempi di esecuzione maggiori.

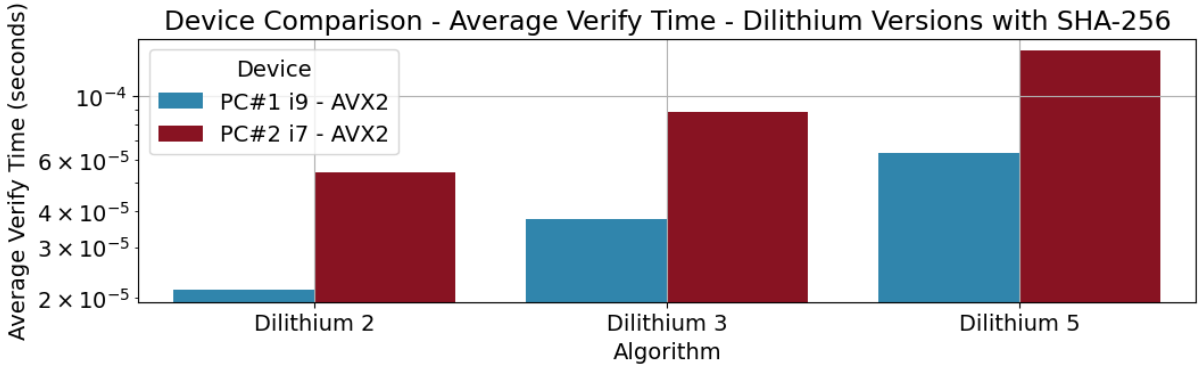

Figura 5.7: Tempi di verifica con messaggi di lunghezza fissa su PC#1 e PC#2 per CRYSTALS Dilithium

Allo stesso modo il grafico 5.7 è significativo per i tempi di verifica della firma. Le funzioni di *hashing* utilizzate sono le medesime cioè *SHA-256* e *SHA-512*. In generale si può notare come in entrambi i dispositivi le operazioni di verifica richiedano inferiore tempo rispetto alla firma. Quest'ultima è una caratteristica che accomuna la maggior parte dei sistemi di firma digitale. È intrinseca nel loro metodo di funzionamento poiché la verifica di un messaggio richiede operazioni di minor complessità.

Successivamente viene proposta una tabella (5.1) riassuntiva con le medie dei tempi e le occupazioni in memoria di chiavi e firme. Quest'ultima mette in relazione le versioni *AVX2* e *REF* di cui non vengono presentati i grafici.

	Dilithium 2	**Dilithium 3**	**Dilithium 5**
Dimensione chiave privata (bytes)	2528	4000	4864
Dimensione chiave pubblica (bytes)	1312	1952	2592
Dimensione firma (bytes)	2420	3293	4595
Livello di sicurezza (bytes)	2	3	5
REF			
Tempo medio Key Gen (ns)	68	118	174
Tempo medio firma SHA256 (ns)	277	431	567
Tempo medio firma SHA512 (ns)	254	430	519
Tempo medio verifica SHA256 (ns)	72	117	180
Tempo medio verifica SHA512 (ns)	71	113	169
AVX2			
Tempo medio Key Gen (ns)	23	46	66
Tempo medio firma SHA256 (ns)	52	105	138
Tempo medio firma SHA512 (ns)	56	113	146
Tempo medio verifica SHA256 (ns)	20	36	61
Tempo medio verifica SHA512 (ns)	20	40	67

Tabella 5.1: Confronto tra versioni *REF* e *AVX2* di CRYSTALS Dilithium. Tempi riferiti all'esecuzione su PC#1

5.2 FALCON

Per ogni implementazione, sia essa ottimizzata o di riferimento, le versioni disponibili sono *FALCON 512* e *FALCON 1024*, i cui livelli di sicurezza sono rispettivamente 1 e 5. Questo candidato alla standardizzazione offre un inferiore numero di livelli di sicurezza disponibili, tuttavia essi sono i valori agli estremi della scala.

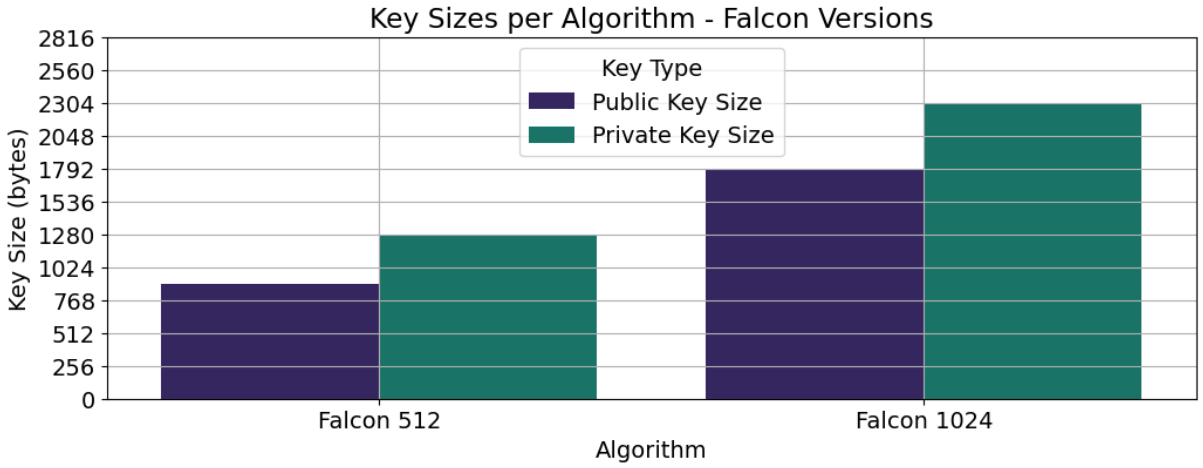

Figura 5.8: Dimensione delle chiavi private e pubbliche per le versioni di FALCON

Nonostante ciò, dal grafico 5.8 è possibile notare come la differenza in dimensioni tra le chiavi non sia eccessiva. Al contrario, proprio perché uno degli obiettivi secondari di FALCON era la compattezza, le sue chiavi sono tra le più corte se prendiamo in considerazione solo i sistemi di firma digitale *Lattice Based*.

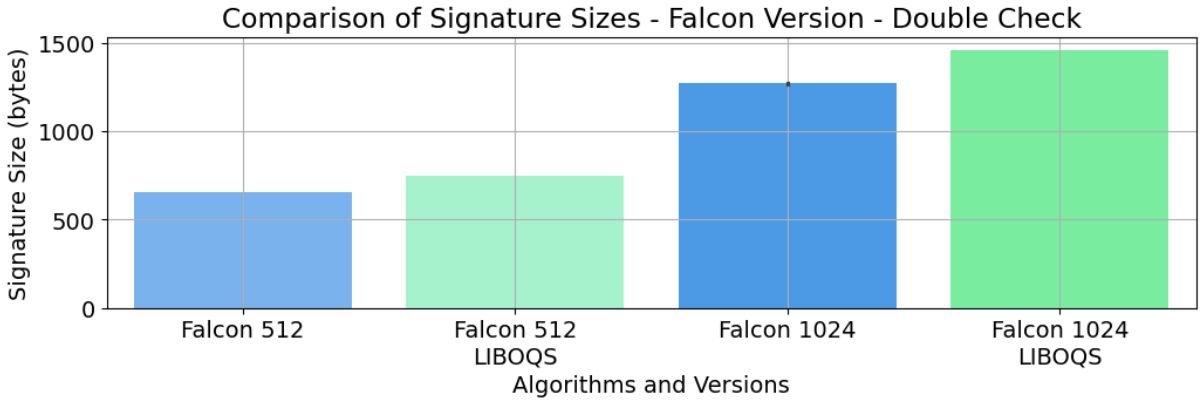

Figura 5.9: Confronto delle dimensioni delle firme tra l'implementazione diretta e l'implementazione LibOQS per FALCON

Le stesse caratteristiche si notano nel grafico 5.9: la dimensione della firma è piuttosto limitata. Si nota un altro aspetto importante: al contrario di CRYSTALS Dilithium, la dimensione della firma tra l'implementazione diretta e l'implementazione tramite libreria *LibOQS* differisce. Ciò accade perché l'implementazione diretta utilizza una versione di FALCON che genera la firma più compatta possibile, rendendo la dimensione delle firme non costante. Invece la libreria *LibOQS* implementa una versione di FALCON con *zero-padding* sulla firma, così da rendere l'uso del sistema di firma più uniforme e più resistente ai *side channel attacks*.

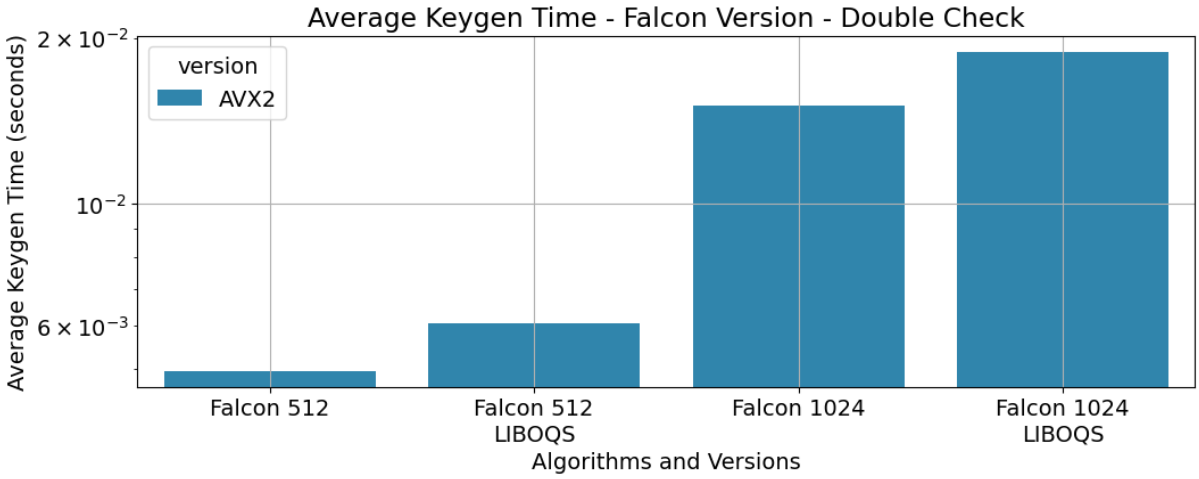

Figura 5.10: Confronto dei tempi di generazione chiavi tra l'implementazione diretta e l'implementazione LibOQS per FALCON

I tempi di *keygen* nel grafico 5.10 rispecchiano maggiormente la differenza di livelli di sicurezza offerti dalle due varianti di FALCON. Ancora una volta *LibOQS* risulta leggermente più lenta, probabilmente per operazioni aggiuntive dovute al *wrapping*.

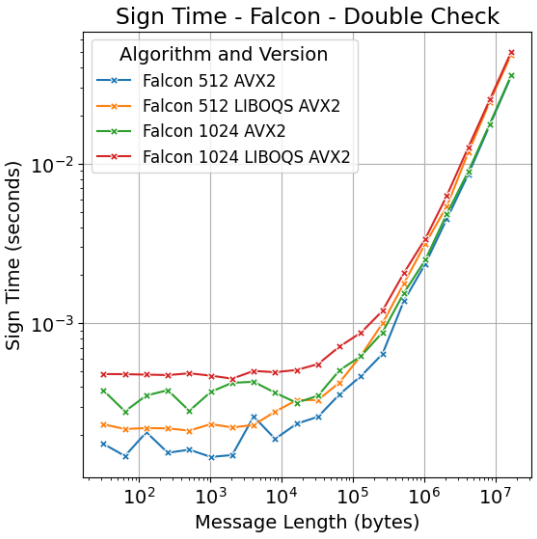

Figura 5.11: Controllo dei tempi di firma con LibOQS per FALCON con messaggi di lunghezza variabile

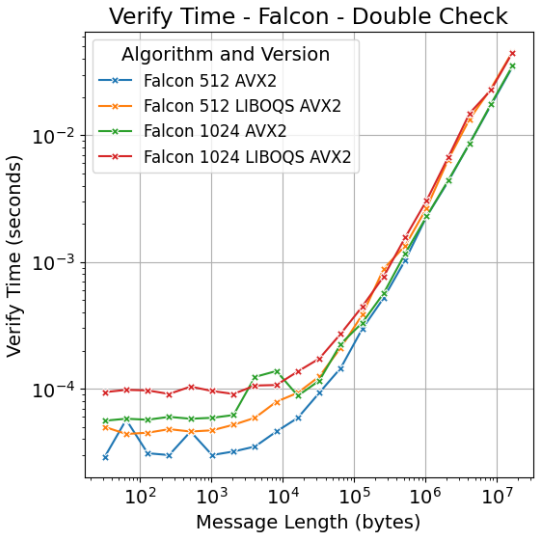

Figura 5.12: Controllo dei tempi di verifica con LibOQS per FALCON con messaggi di lunghezza variabile

Anche FALCON, come Dilithium, presenta nei grafici 5.11 e 5.12 un aumento dei tempi esponenziale quando la dimensione del messaggio da firmare e verificare cresce oltre l'ordine dei KiloBytes.

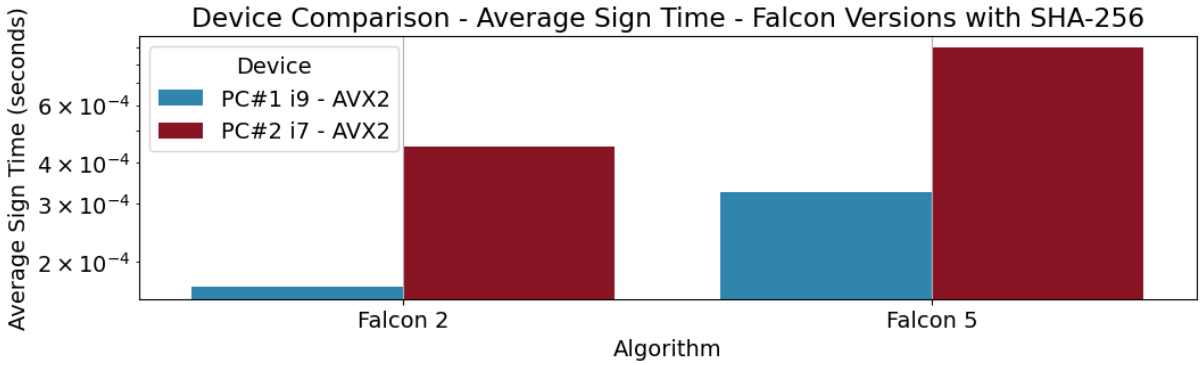

Figura 5.13: Tempi di firma con messaggi di lunghezza fissa su PC#1 e PC#2 per FALCON

Per quanto riguarda i test che includono l'*hashing* del messaggio, il grafico 5.13 mostra che i tempi di firma rimangono ridotti su i dispositivi utilizzati per la fase di test, permettendo di effettuare circa un migliaio di operazioni di firma al secondo, come evidenziato anche dal team di ricerca nella relativa documentazione [23].

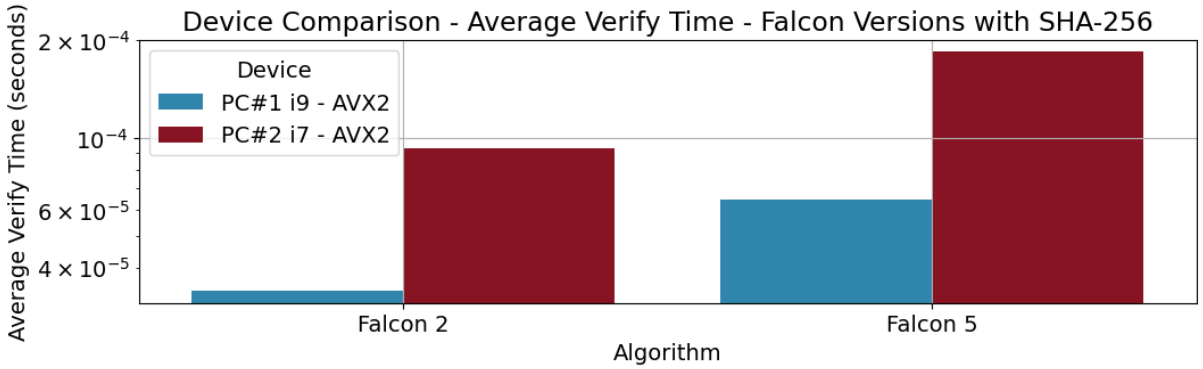

Figura 5.14: Tempi di verifica con messaggi di lunghezza fissa su PC#1 e PC#2 per FALCON

Allo stesso modo, nel grafico 5.14, i tempi di verifica con *hashing* del messaggio mostrano un incremento delle performance di circa un ordine di grandezza rispetto ai tempi di firma.

La differenza tra i risultati di performance dei due dispositivi è piuttosto lineare. Con tutta probabilità tale differenza è dovuta al salto prestazionale tra i due processori (frequenza base, numero di cores, eccetera) piuttosto che a peculiarità dell'algoritmo in esecuzione.

Successivamente vengono riassunte le informazioni nella tabella 5.2. Quest'ultima mette in relazione le versioni *AVX2* e *REF*, di cui non sono stati presentati i grafici.

	FALCON 512	FALCON 1024
Dimensione chiave privata (bytes)	1281	2305
Dimensione chiave pubblica (bytes)	897	1793
Dimensione firma (bytes)	752	1462
Livello di sicurezza (bytes)	1	5
REF		
Tempo medio Key Gen (ns)	4986	14667
Tempo medio firma SHA256 (ns)	201	426
Tempo medio firma SHA512 (ns)	211	390
Tempo medio verifica SHA256 (ns)	32	60
Tempo medio verifica SHA512 (ns)	34	59
AVX2		
Tempo medio Key Gen (ns)	5032	14965
Tempo medio firma SHA256 (ns)	168	330
Tempo medio firma SHA512 (ns)	184	308
Tempo medio verifica SHA256 (ns)	32	59
Tempo medio verifica SHA512 (ns)	33	58

Tabella 5.2: Confronto tra le versioni *REF* e *AVX2* di FALCON. Tempi riferiti all'esecuzione su PC#1

5.3 SPHINCS+

Come anticipato nei capitoli precedenti, le implementazioni di SPHINCS+ si diversificano per la funzione di *hashing* utilizzata. poiché l'implementazione *Haraka* attualmente risulta poco sicura, tra le soluzioni *SHA256* e *SHAKE256* è stata selezionata la prima.

Sulla base di questa, sono poi state considerate le rispettive versioni *REF* e *AVX2* in tutte le varianti richieste dai test, cioè SPHINCS+ 128, SPHINCS+ 192 e SPHINCS+ 256 che offrono rispettivamente i livelli di sicurezza 2, 3 e 5, come CRYSTALS Dilithium.

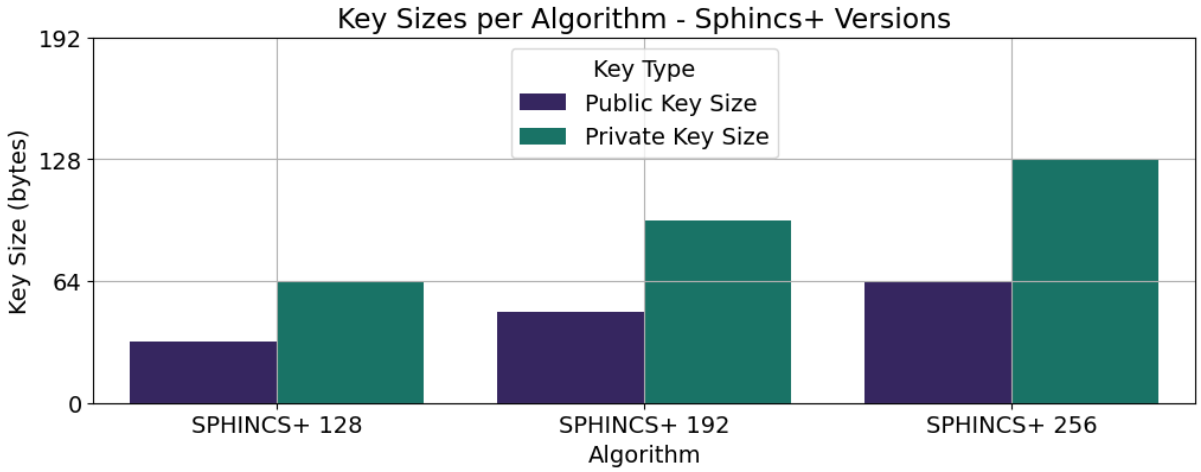

Figura 5.15: Dimensione delle chiavi private e pubbliche per le versioni di SPHINCS+

Il grafico 5.15 mette subito in evidenza una caratteristica fondamentale di SPHINCS+: le chiavi hanno dimensioni ridottissime, pari a qualche decina di bytes. Questo perché SPHINCS+ è un sistema di firma digitale *Hash Based*.

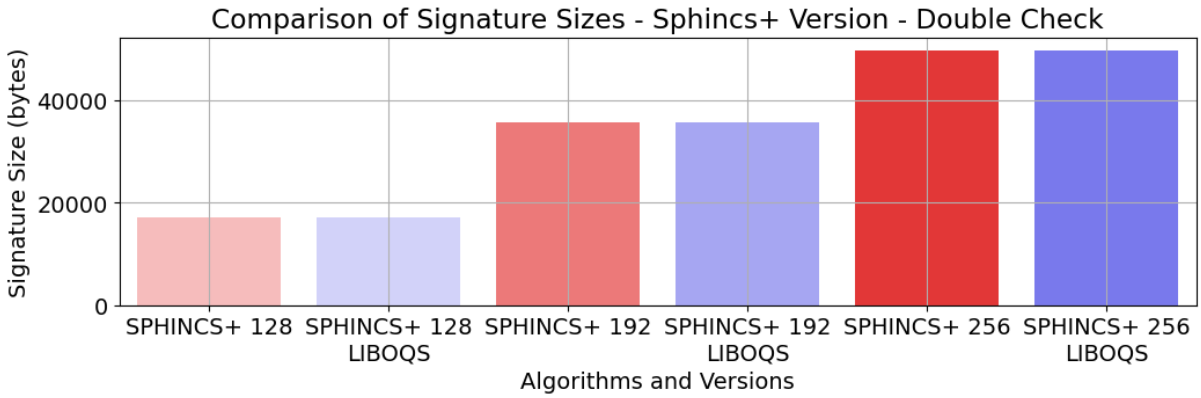

Figura 5.16: Confronto delle dimensioni delle firme tra l'implementazione diretta e l'implementazione LibOQS per SPHINCS+

Al contrario, le dimensioni delle firme sono molto elevate, come è possibile notare dal grafico 5.16. La dimensione delle firme è costante per ciascuna versione dell'algoritmo, dunque sia l'implementazione diretta che l'implementazione con *LibOQS* hanno riportato gli stessi risultati.

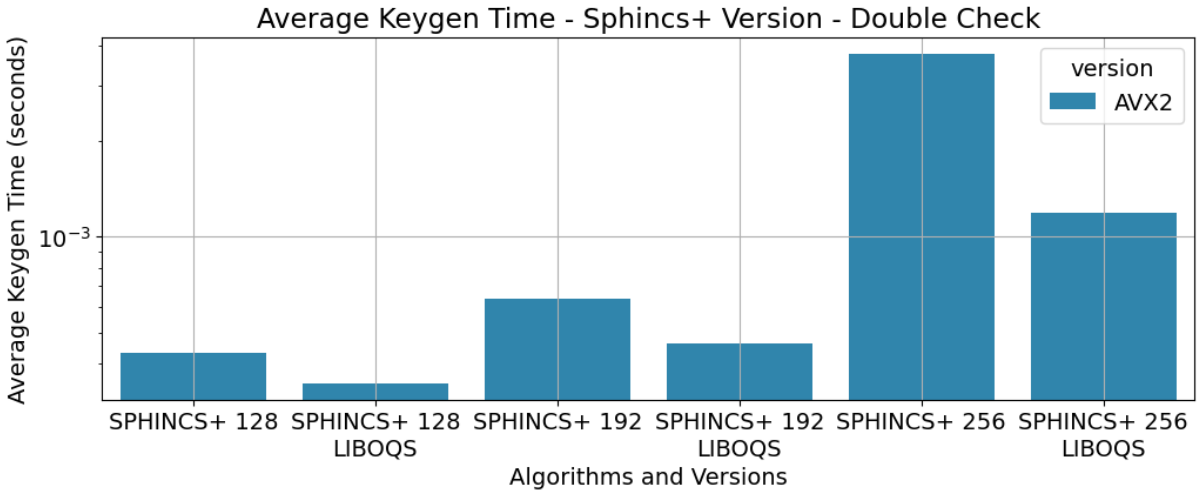

Figura 5.17: Confronto dei tempi di generazione chiavi tra l'implementazione diretta e l'implementazione LibOQS per SPHINCS+

I tempi di *keygen* nel grafico 5.17 rispecchiano la differenza di livello di sicurezza offerta dalle versioni di SPHINCS+ in maniera analoga a quanto visto per i precedenti candidati.

In questo caso l'implementazione della libreria *LibOQS* risulta la più veloce in tutti gli ambiti (sia per la generazione delle chiavi, sia in firma e verifica). Questa peculiarità è probabilmente dovuta ai parametri di compilazione di SPHINCS+ introdotti da *LibOQS*, tali da aumentare il numero di ottimizzazioni rispetto all'implementazione diretta offerta. In ogni caso i risultati rimangono sullo stesso ordine di grandezza, per cui possono essere considerati validi.

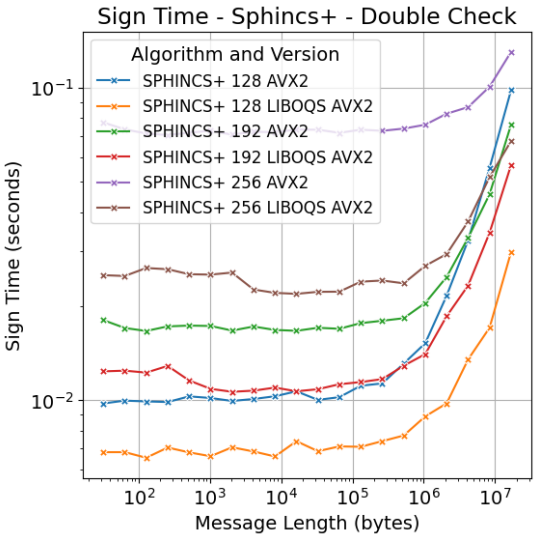

Figura 5.18: Controllo dei tempi di firma con LibOQS per SPHINCS+ con messaggi di lunghezza variabile

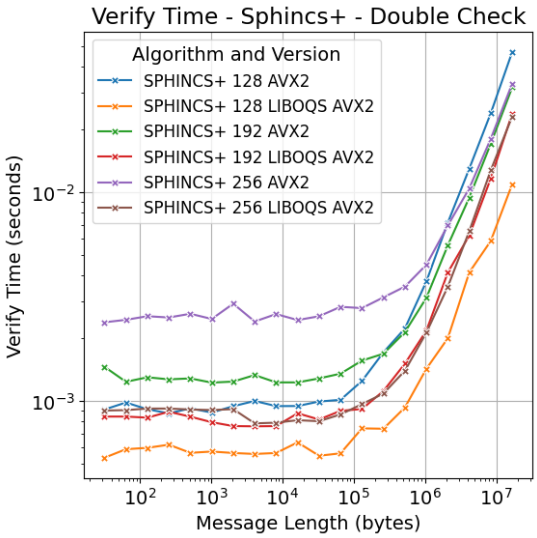

Figura 5.19: Controllo dei tempi di verifica con LibOQS per SPHINCS+ con messaggi di lunghezza variabile

I grafici 5.18 e 5.19 rivelano un comportamento unico di SPHINCS+, dovuto alla natura degli algoritmi *Hash Based*. L'aumento esponenziale dei tempi di firma e verifica viene assunto quando il messaggio (elaborato senza tecniche di *hashing*) raggiunge dimensioni nell'ordine dei MegaBytes. Prima di tale soglia i tempi si mantengono piuttosto costanti e, in generale, sono molto più elevati rispetto agli altri candidati.

Questi aspetti sono dovuti all'alta presenza di operazioni di *hashing* all'interno degli algoritmi di SPHINCS+: essi *appiattiscono* le differenze di tempi dovute alla dimensione dell'input e rallentano i tempi di esecuzione poiché sono operazioni onerose.

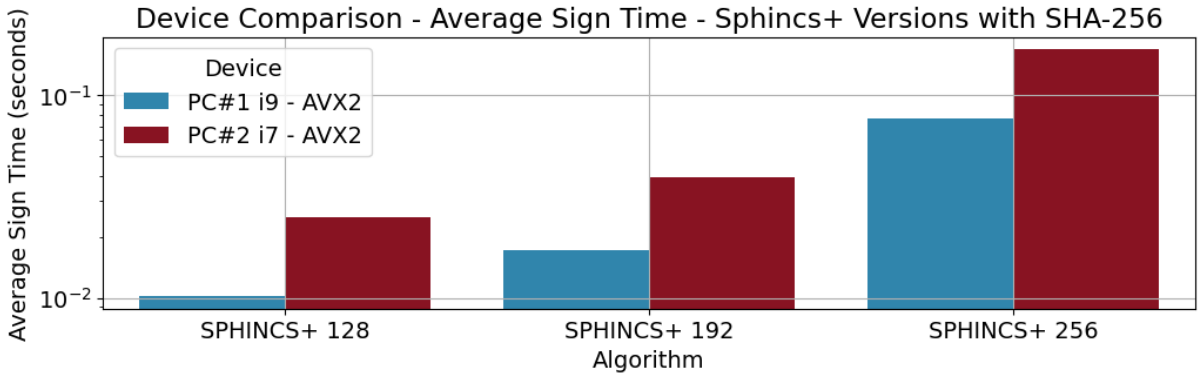

Figura 5.20: Tempi di firma con messaggi di lunghezza fissa su PC#1 e PC#2 per SPHINCS+

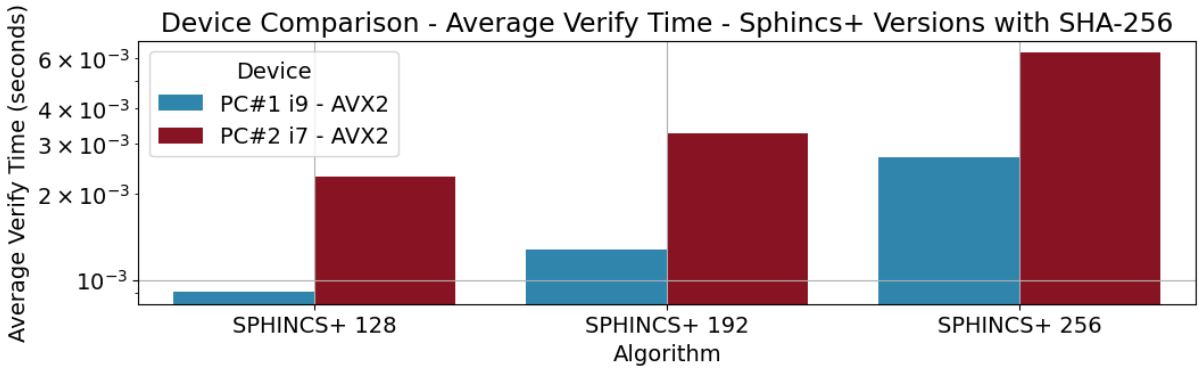

Figura 5.21: Tempi di verifica con messaggi di lunghezza fissa su PC#1 e PC#2 per SPHINCS+

I test che includono l'*hashing* del messaggio con *SHA-256* e *SHA-512*, evidenziati nei grafici 5.20 e 5.21, mostrano tempistiche che difficilmente permettono di effettuare numeri elevati di firme e verifiche al secondo. Le tempistiche aumentano in maniera esponenziale al livello di sicurezza offerto.

Di seguito è presente un riassunto in formato tabellare dei dati precedenti. La tabella 5.3 mette in relazione le versioni *AVX2* e *REF* di SPHINCS+, di cui non sono stati presentati i grafici.

SPHINCS+ Versione:	S+ 128	S+ 192	S+ 256
Dimensione chiave privata (bytes)	64	96	128
Dimensione chiave pubblica (bytes)	32	48	64
Dimensione firma (bytes)	17088	16224	29792
Livello di sicurezza	2	3	5
REF			
Tempo medio Key Gen (ns)	1961	2967	10366
Tempo medio firma SHA256 (ns)	46825	75216	202425
Tempo medio firma SHA512 (ns)	46302	76296	199154
Tempo medio verifica SHA256 (ns)	2861	4310	5883
Tempo medio verifica SHA512 (ns)	2888	4382	5786
AVX2			
Tempo medio Key Gen (ns)	437	644	3955
Tempo medio firma SHA256 (ns)	10105	17201	76303
Tempo medio firma SHA512 (ns)	10119	17284	74597
Tempo medio verifica SHA256 (ns)	921	1272	2714
Tempo medio verifica SHA512 (ns)	897	1280	2688

Tabella 5.3: Confronto tra versioni *REF* e *AVX2* di SPHINCS+. Tempi riferiti all'esecuzione su PC#1

5.4 RSA

Per poter comprendere meglio la bontà delle prestazioni degli algoritmi PQC, sono stati eseguiti dei test simili anche su una delle tecniche attualmente utilizzate nei processi di firma digitale, ovvero RSA.

Sono state considerate tutte le versioni di RSA associate ai vari livelli di sicurezza. Nella tabella riassuntiva finale è stata aggiunta una riga per specificare i vari *Modulus* di ciascuna versione, cioè la stringa da cui vengono generate le chiavi private e pubbliche.

Solitamente le versioni di RSA non vengono riconosciute in funzione del livello di sicurezza o della dimensione delle chiavi, bensì in funzione del *Modulus*, dunque quando si parla di RSA 1024 si sta identificando l'algoritmo RSA che utilizza chiavi generate a partire da un *Modulus* di 1024 bits.

L'inclusione delle versioni meno sicure di RSA nei test è motivata dal loro grande utilizzo, dunque gli obiettivi dei nuovi algoritmi PQC in termini di performance dovrebbero essere tarati proprio sui risultati di queste versioni di RSA, sebbene poco sicure.

I test su RSA sono stati implementati utilizzando direttamente le librerie *OpenSSL* installate nel sistema operativo, dunque a differenza degli altri algoritmi non è stato utiliz-

zato e compilato alcun codice sorgente, escluso quello di implementazione dei *performance test*.

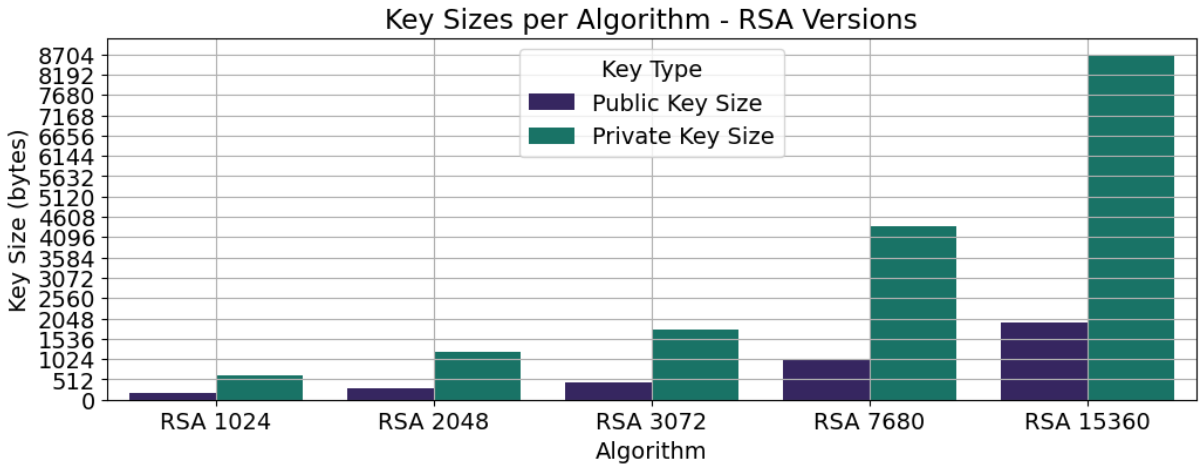

Figura 5.22: Dimensione delle chiavi private e pubbliche di RSA in funzione dei *Modulus*

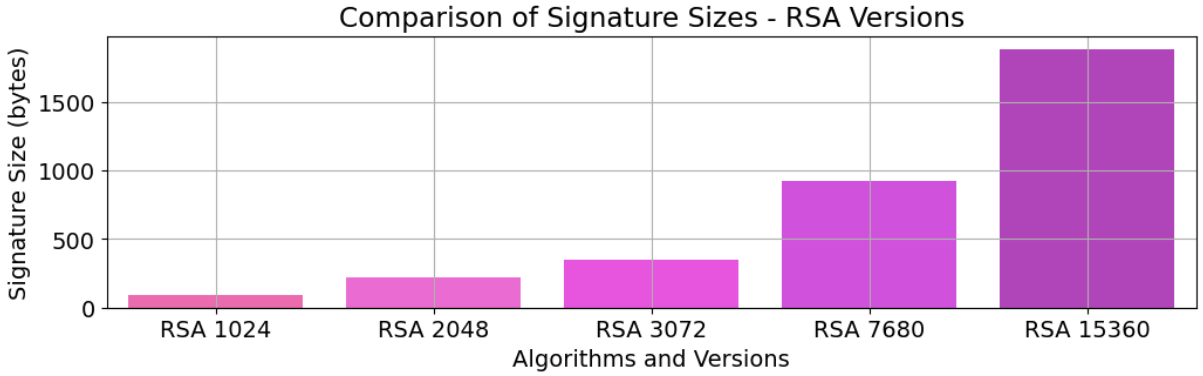

Figura 5.23: Dimensione delle firme di RSA in funzione dei *Modulus*

Per quanto riguarda la dimensione delle chiavi, entrambe contengono il *Modulus* tuttavia:

1. la chiave pubblica contiene pochi altri dati, per cui la sua dimensione finale (in Bytes) è poco maggiore a quella del *Modulus* (espresso in bits).

2. la chiave privata contiene molti altri parametri, per cui genericamente possiede una dimensione circa 4 volte maggiore rispetto al *Modulus* (espresso in bits) [34].

Nei grafici 5.22 e 5.23 le dimensioni delle chiavi e delle firme crescono vertiginosamente all'aumentare della dimensione del *Modulus*. Considerando il caso in cui il *Modulus* è pari a 15360 bits, vuol dire che esso rappresenta un numero intero positivo dal valore compreso tra 2^{15359} e 2^{15360}, valore non rappresentabile da nessun tipo di hardware tradizionale. Per tale motivo è ragionevole che i tempi di generazione delle chiavi siano così elevati.

Al di là delle vulnerabilità di sicurezza che lo coinvolgono, è difficile immaginare un ampio utilizzo di RSA nei prossimi decenni se l'aumento del livello di sicurezza introduce calcoli così complessi da rendere i tempi di esecuzione proibitivi.

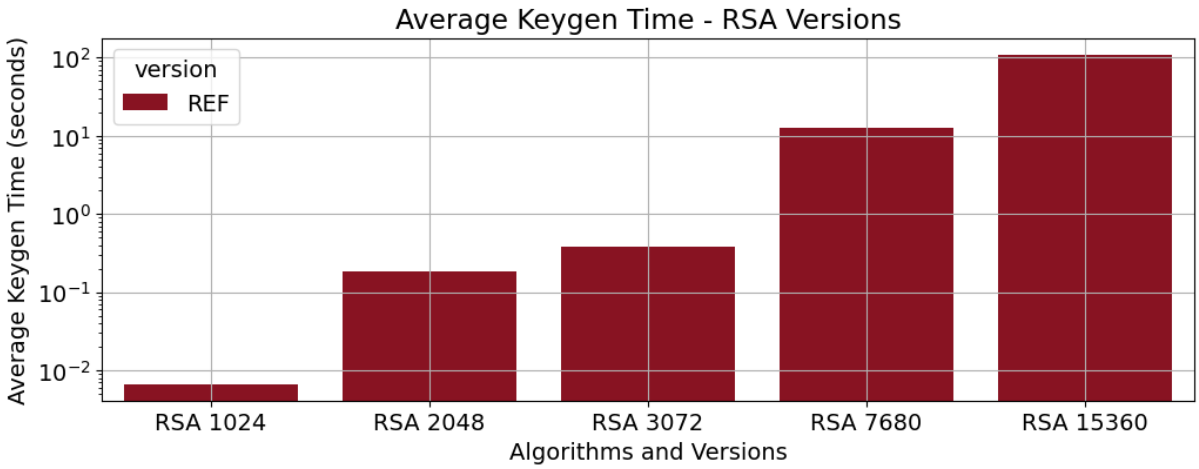

Figura 5.24: Tempo di generazione delle chiavi di RSA in funzione dei *Modulus*. Tempi misurati su PC#1

.

Il grafico 5.24 dimostra che anche i tempi di generazione della chiavi esplodono in maniera esponenziale, richiedendo decine di secondi per le versioni di RSA più sicure (misurati sull'hardware PC#1).

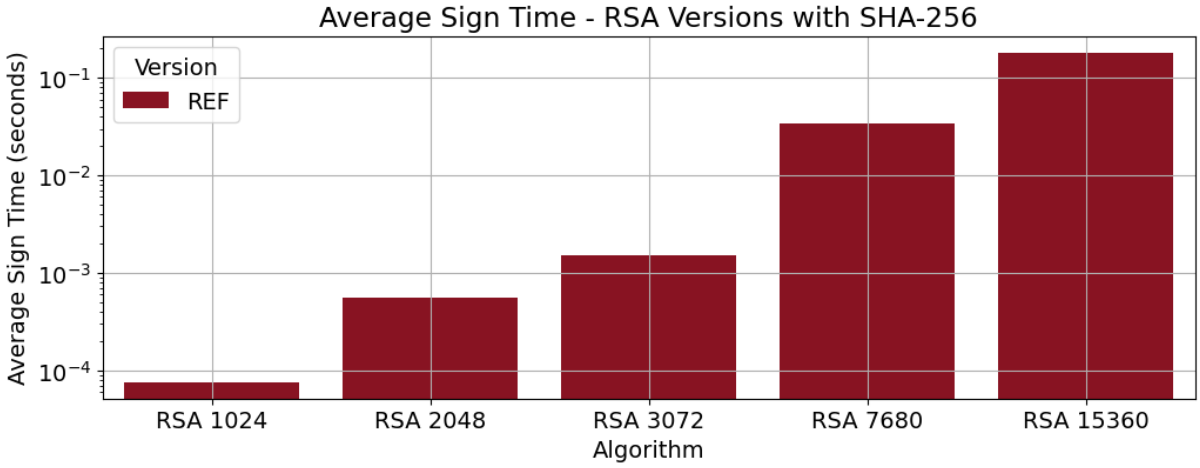

Figura 5.25: Tempi di firma di RSA eseguito su messaggi di lunghezza fissa 32 byte

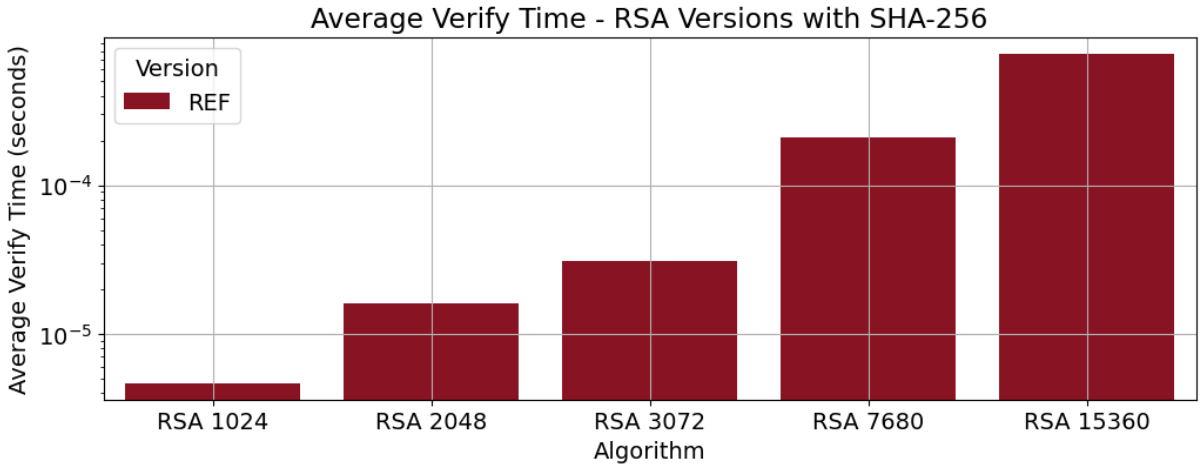

Figura 5.26: Tempi di verifica di RSA eseguito su messaggi di lunghezza fissa 32 byte

Allo stesso modo nei grafici 5.25 e 5.26 si notano gli stessi comportamenti nel caso di firma di un messaggio (che ha subito una fase di *hashing* con *SHA-256*) e nel caso di verifica della firma di un messaggio. I tempi aumentano esponenzialmente ma rimangono accettabili per eseguire più firme e verifiche in un breve tempo.

RSA con Modulus:	**1024**	**2048**	**3072**	**7680**	**15360**
Dimensione chiave privata (bytes)	610	1193	1767	4364	8684
Dimensione chiave pubblica (bytes)	162	294	422	998	1958
Dimensione firma (bytes)	96	224	352	928	1856
Livello di sicurezza	1	2	3	4	5
Modulus (bits)	1024	2048	3072	7680	15360
Tempo medio Key Gen (ms)	6.7	186.2	377.5	12749.6	109706.9
Tempo medio firma SHA256 (ns)	71	531	1451	35299	181973
Tempo medio firma SHA512 (ns)	81	545	1595	36041	182464
Tempo medio verifica SHA256 (ns)	4.7	15	30	209	766
Tempo medio verifica SHA512 (ns)	5.1	18	39	201	768

Tabella 5.4: Confronto tra diverse configurazioni di RSA. Tempi riferiti all'esecuzione su PC#1

5.5 Confronti tra algoritmi

Quest'ultima sezione ha l'obiettivo di confrontare direttamente gli algoritmi tra loro, al fine di evidenziare aspetti non rilevabili analizzando i candidati singolarmente. Per aggiungere un ulteriore metro di paragone sarà presente anche RSA (ove significativo).

Ciascun algoritmo verrà confrontato con gli altri candidati in funzione dei livelli di sicurezza. I grafici infatti conterranno solo versioni di algoritmi con livello di sicurezza pari o similare. Nelle colonne della tabella 5.5 sono evidenziati i confronti eseguiti.

Livello di sicurezza	Livello 1/2	Livello 3	Livello 5
CRYSTALS Dilithium	Dilithium 2	Dilithium 3	Dilithium 5
FALCON	Falcon 512	-	Falcon 1024
SPHINCS+	SPHINCS+ 128	SPHINCS+ 192	SPHINCS+ 256
RSA	RSA 2048	RSA 3072	RSA 15360

Tabella 5.5: Ogni colonna contiene gli algoritmi che hanno un livello di sicurezza confrontabile

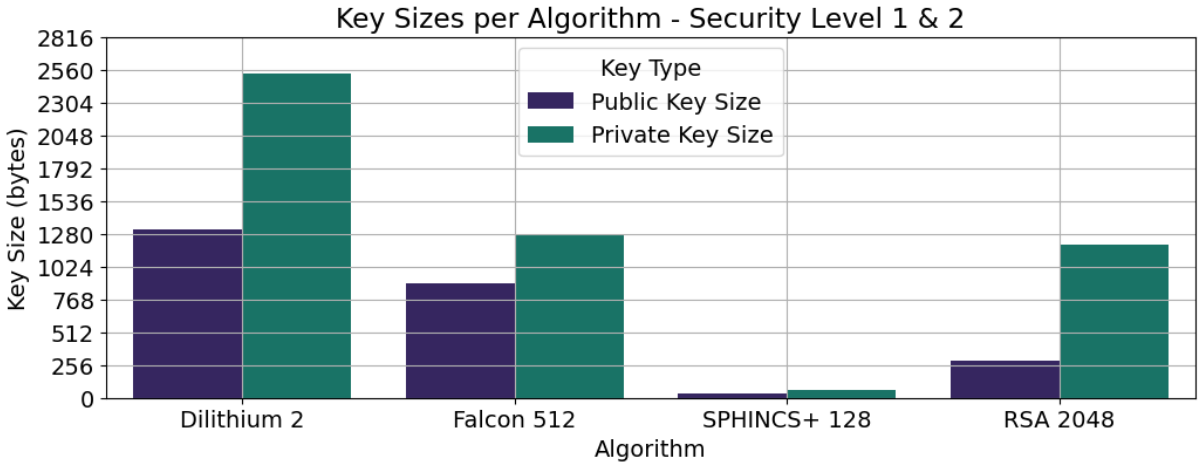

Figura 5.27: Dimensione delle chiavi private e pubbliche per gli algoritmi che offrono livello di sicurezza 1 o 2: CRYSTALS Dilithium, FALCON, SPHINCS+ e RSA

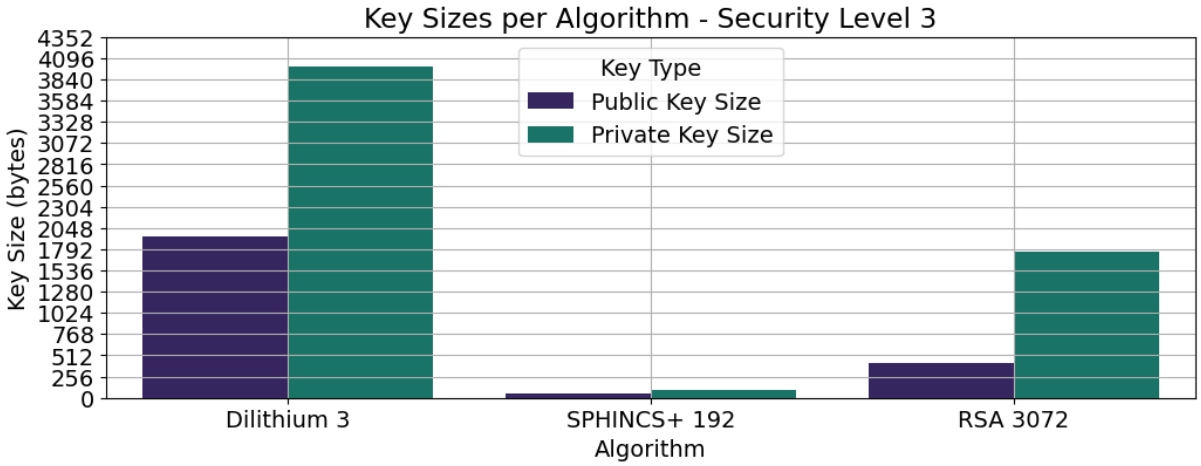

Figura 5.28: Dimensione delle chiavi private e pubbliche per gli algoritmi che offrono livello di sicurezza 3: CRYSTALS Dilithium, SPHINCS+ e RSA

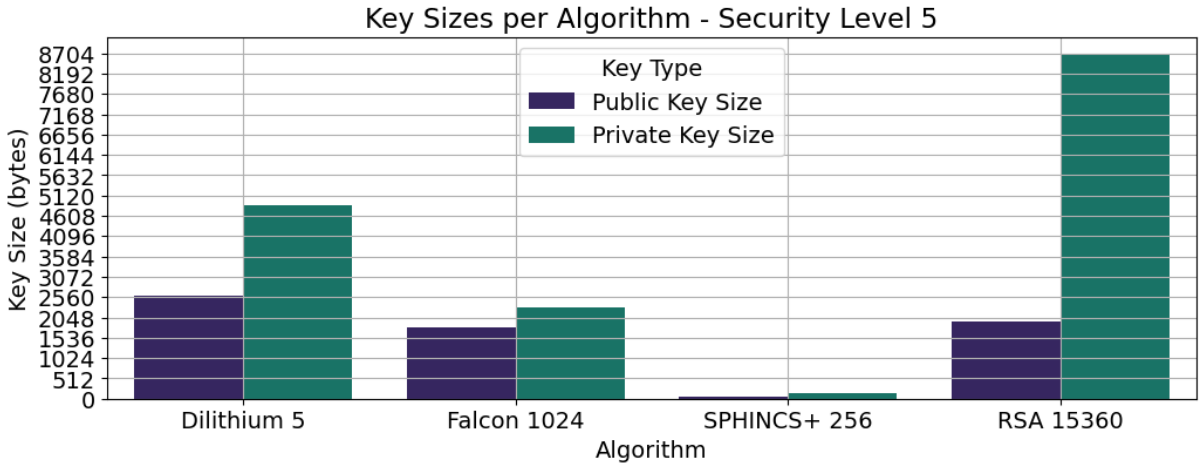

Figura 5.29: Dimensione delle chiavi private e pubbliche per gli algoritmi che offrono livello di sicurezza 5: CRYSTALS Dilithium, FALCON, SPHINCS+ e RSA

I tre grafici precedenti (5.27 5.28 5.29) si concentrano sul confrontare le dimensione delle chiavi, ciascuno a parità del livello di sicurezza trattato.

Uno degli aspetti non misurati direttamente ma che è possibile dedurre da questi grafici è la minimizzazione dei tempi di trasmissione: per minimizzare la trasmissione, a parità di bitrate, è necessario inviare un numero inferiore di pacchetti ethernet, la cui dimensione è di circa 1500 bytes.

Le chiavi pubbliche sono spesso inviate assieme ai messaggi firmati, quindi sono contenute nelle comunicazioni. Tra i sistemi di firma digitale presenti nei grafici, gli unici che hanno la possibilità di mantenere una dimensione delle chiave pubblica limitata (tale da non richiedere più di un pacchetto) sono SPHINCS+, RSA e FALCON.

Per SPHINCS+ questo risultato è dettato dalla natura dell'algoritmo (*Hash Based*) mentre per FALCON è un risultato degno di nota. Raramente gli algoritmi *Lattice Based* riescono a ridurre le dimensioni delle chiavi in tal modo.

Un'altra cosa che si può comprendere è che RSA è completamente fuori scala se consideriamo la dimensione della chiave privata nella sua implementazione più sicura: questo rende le nuove proposte PQC un'ottima soluzione da standardizzare.

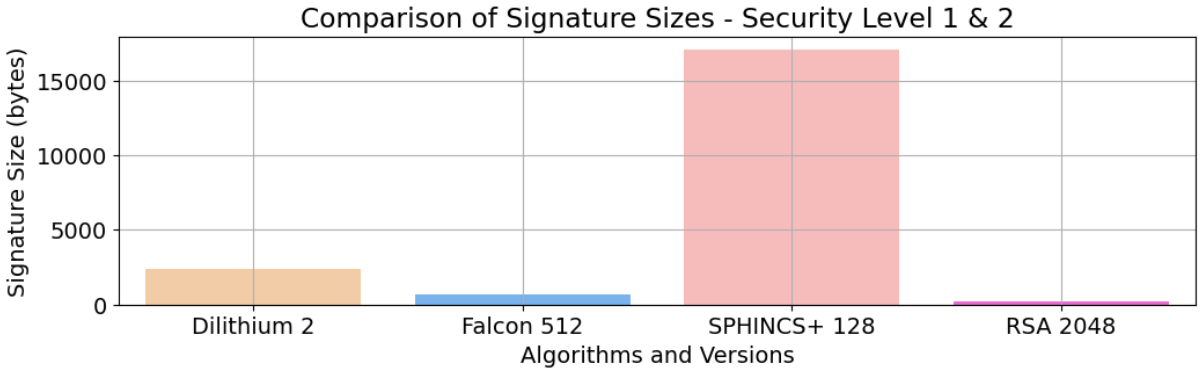

Figura 5.30: Dimensione della firma per gli algoritmi che offrono livello di sicurezza 1 e 2: CRYSTALS Dilithium, FALCON, SPHINCS+ e RSA

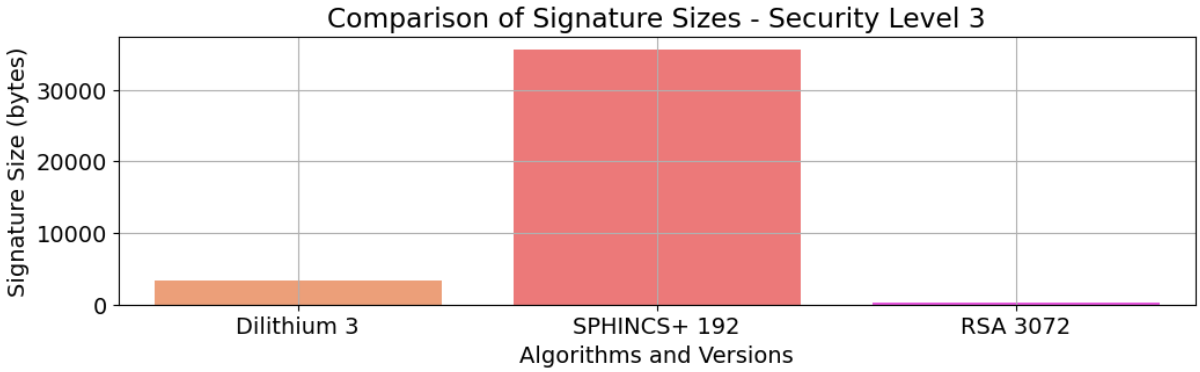

Figura 5.31: Dimensione della firma per gli algoritmi che offrono livello di sicurezza 3: CRYSTALS Dilithium, SPHINCS+ e RSA

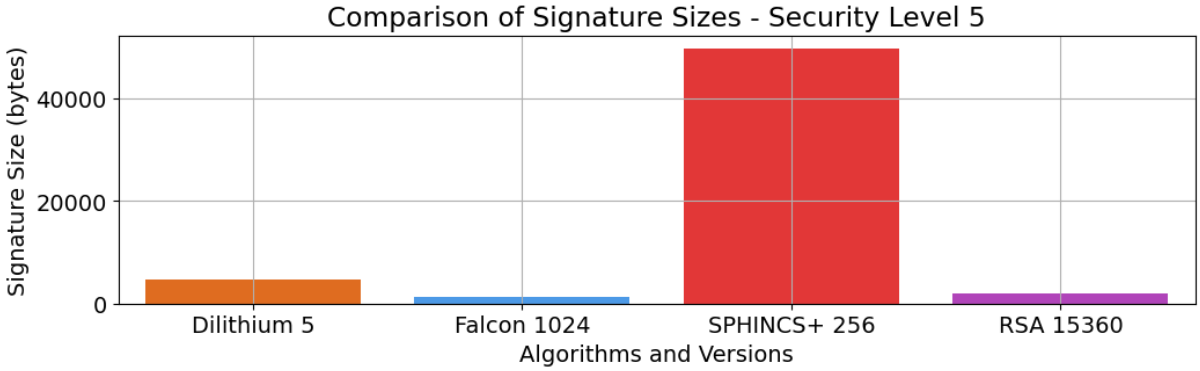

Figura 5.32: Dimensione della firma per gli algoritmi che offrono livello di sicurezza 5: CRYSTALS Dilithium, FALCON, SPHINCS+ e RSA

Se consideriamo le dimensioni delle firme (trattate dai grafici 5.30 5.31 5.32) otteniamo altri aspetti di riflessione importanti: tra gli algoritmi PQC, FALCON sembra l'unico in grado di mantenere ridotta la trasmissione di una comunicazione firmata poiché è l'algoritmo che produce le firme di lunghezza inferiore.

L'obiettivo di *compattezza* che il team di FALCON si è autoimposto ha molto valore se paragonato con le soluzioni attualmente in uso o candidate alla standardizzazione. SPHINCS+, sempre per la propria natura, produce delle firme di dimensioni significative, per cui richiederà grandi tempi di trasmissione rispetto agli altri algoritmi considerati.

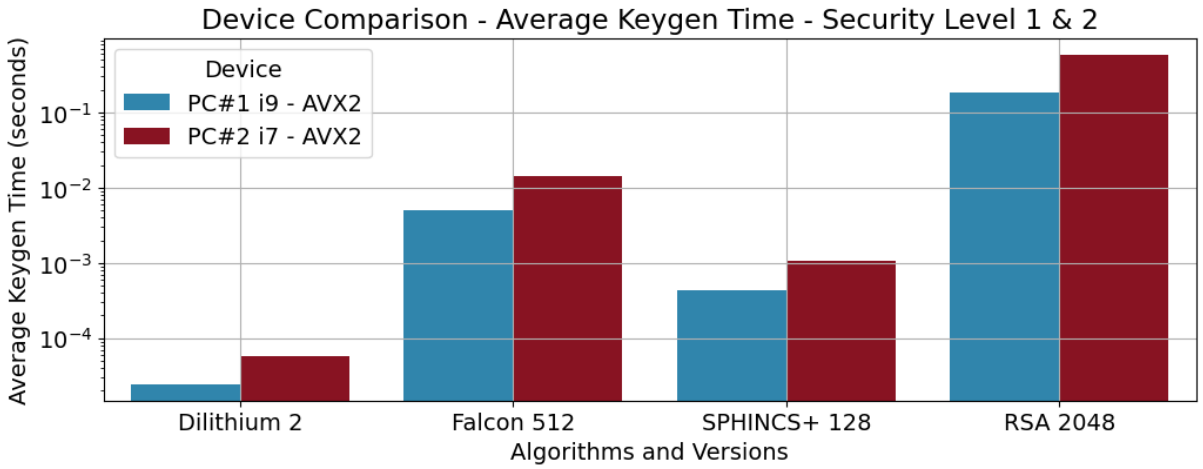

Figura 5.33: Tempi di generazione delle chiavi per gli algoritmi che offrono livello di sicurezza 1 e 2: CRYSTALS Dilithium, FALCON, SPHINCS+ e RSA

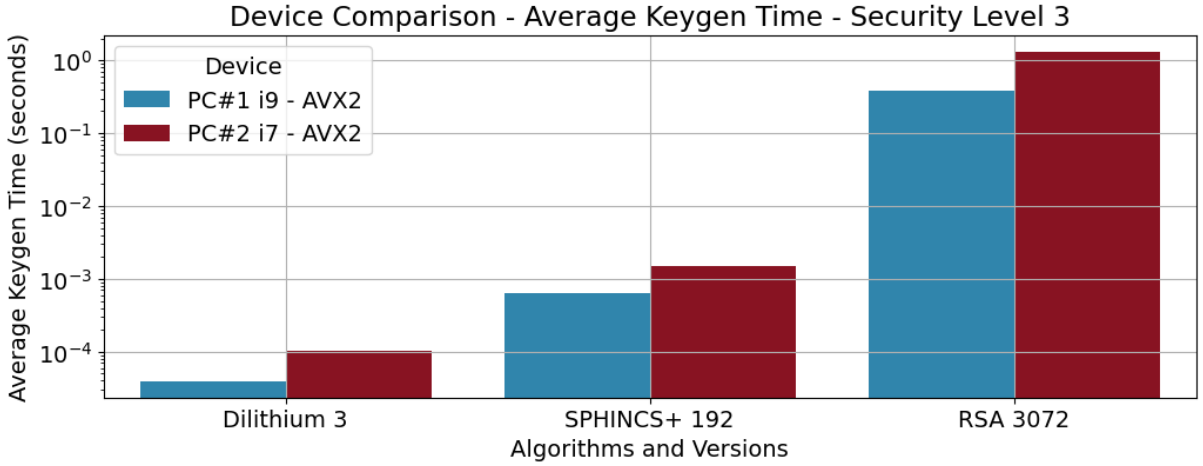

Figura 5.34: Tempi di generazione delle chiavi per gli algoritmi che offrono livello di sicurezza 3: CRYSTALS Dilithium, SPHINCS+ e RSA

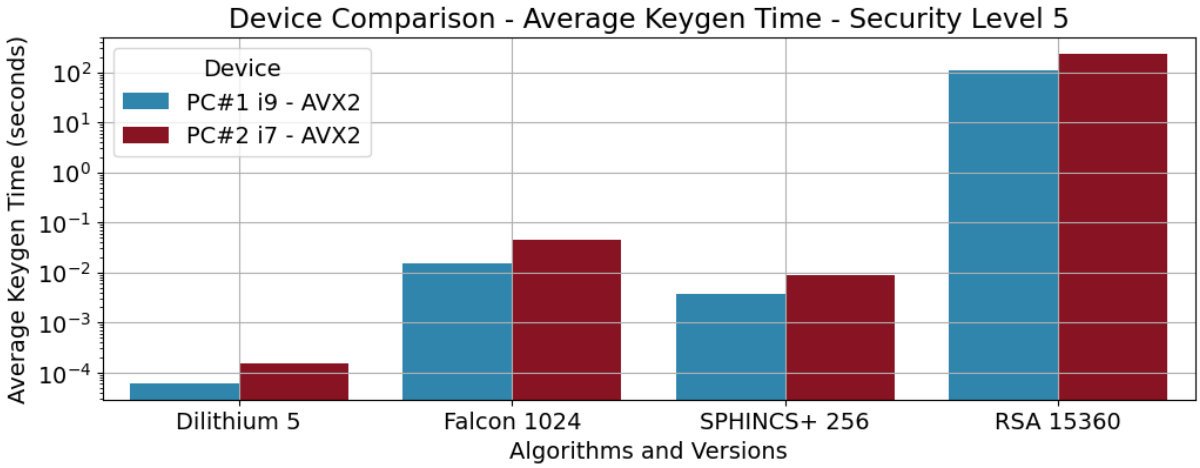

Figura 5.35: Tempi di generazione delle chiavi per gli algoritmi che offrono livello di sicurezza 5: CRYSTALS Dilithium, FALCON, SPHINCS+ e RSA

Per l'analisi dei tempi di generazione delle chiavi sono stati sviluppati i grafici 5.33 5.34 5.35. Tra loro sono molto simili infatti, tra i vari livelli di sicurezza, si può notare che le differenze di prestazioni tra gli algoritmi sono mantenute in proporzione.

Come già anticipato, il NIST considera il tempo di generazioni chiavi come un aspetto secondario. Effettivamente questo ragionamento è intuitivo poiché questa procedura viene eseguita molte meno volte rispetto alle procedura di firma e verifica. In ogni caso, quando si parla di tempistiche, CRYSTALS Dilithium si mantiene tra i più veloci.

Per i tempi di firma e verifica verranno mostrati solo i grafici in cui i messaggi sono stati pre-elaborati con *hashing* tramite *SHA-256*. Ciò perché *SHA-256* è attualmente il più utilizzato nelle firme *AdES* (standard europeo presentato in introduzione) [35]. Indipendentemente dall'input, *SHA-256* produrrà sempre una stringa di output la cui dimensione è 32 bytes, che verrà poi firmata.

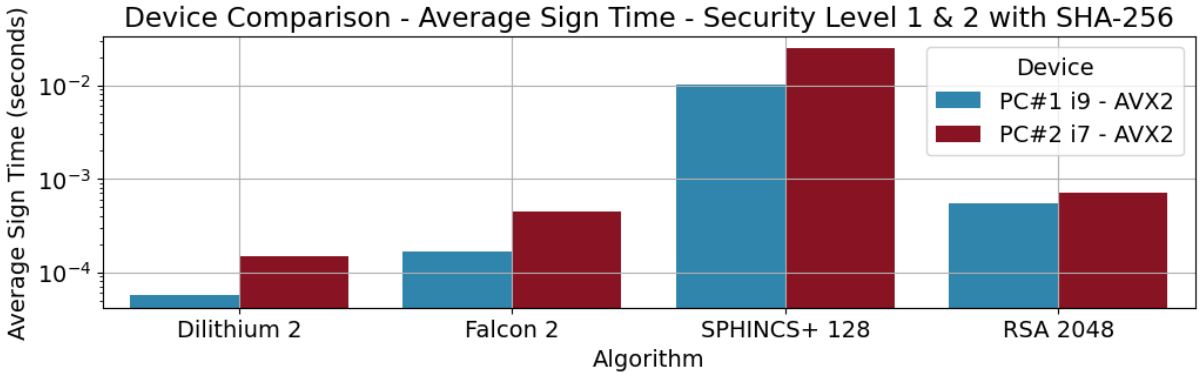

Figura 5.36: Tempi di firma con messaggio di lunghezza fissa su PC#1 e PC#2 per gli algoritmi che offrono livello di sicurezza 1 e 2

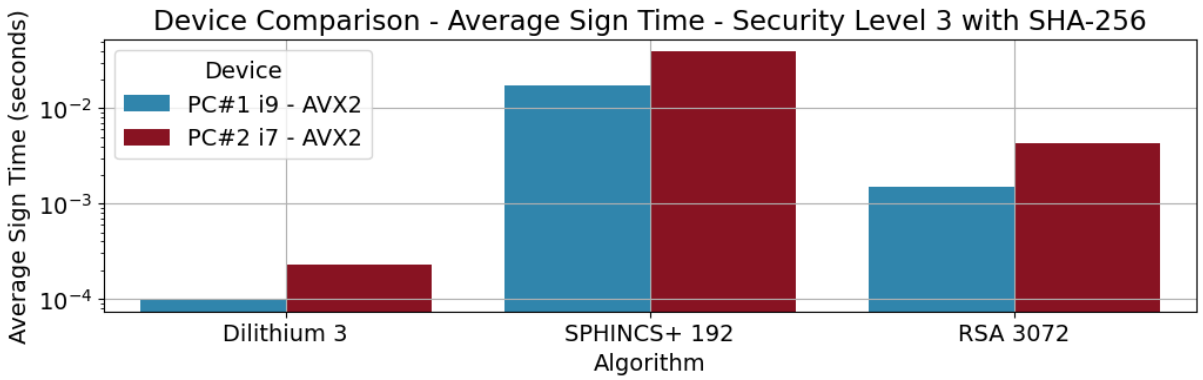

Figura 5.37: Tempi di firma con messaggio di lunghezza fissa su PC#1 e PC#2 per gli algoritmi che offrono livello di sicurezza 3

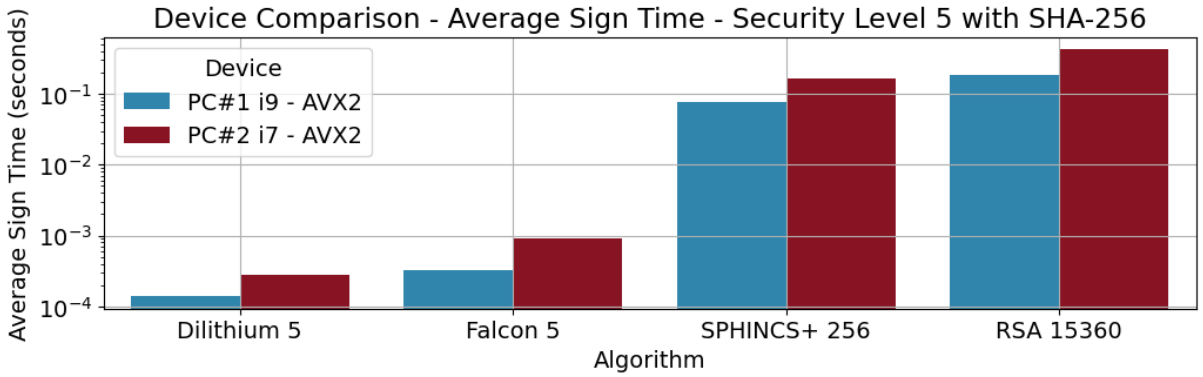

Figura 5.38: Tempi di firma con messaggio di lunghezza fissa su PC#1 e PC#2 per gli algoritmi che offrono livello di sicurezza 5

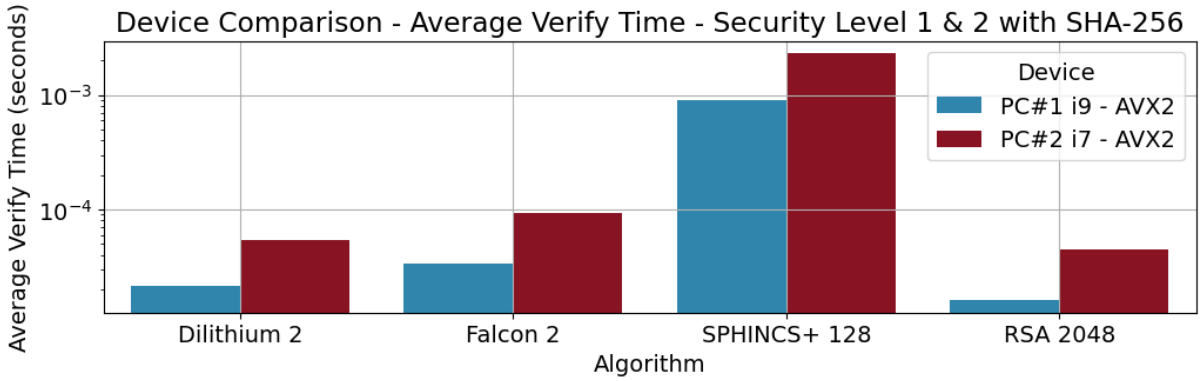

Figura 5.39: Tempi di verifica con messaggio di lunghezza fissa su PC#1 e PC#2 per gli algoritmi che offrono livello di sicurezza 1 e 2

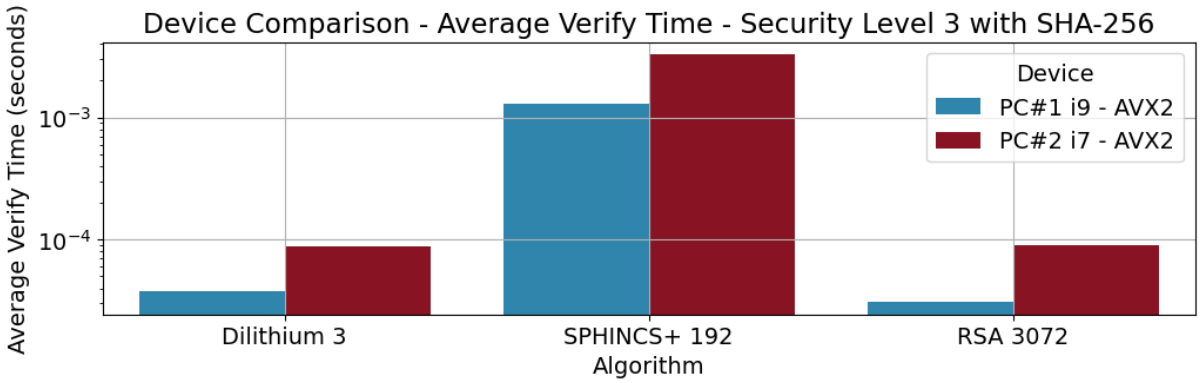

Figura 5.40: Tempi di verifica con messaggio di lunghezza fissa su PC#1 e PC#2 per gli algoritmi che offrono livello di sicurezza 3

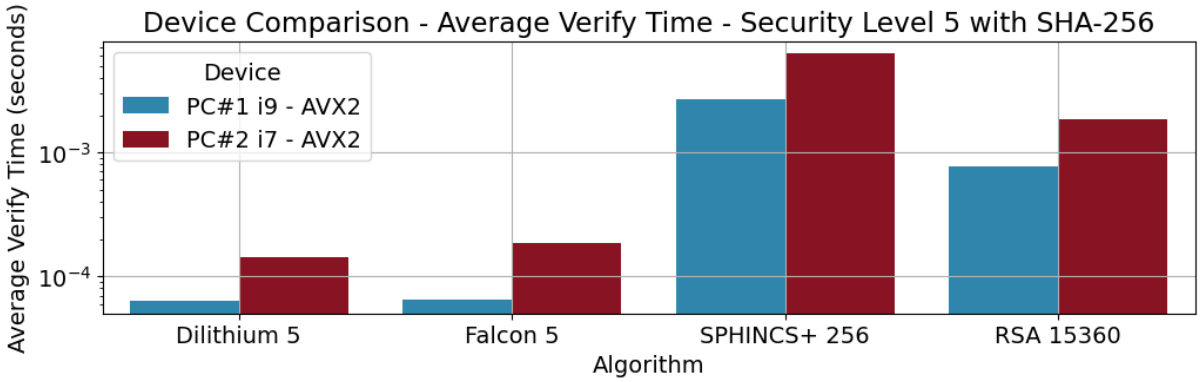

Figura 5.41: Tempi di verifica con messaggio di lunghezza fissa su PC#1 e PC#2 per gli algoritmi che offrono livello di sicurezza 5

I grafici dei tempi di firma (5.36, 5.37, 5.38) e dei tempi di verifica (5.39, 5.40, 5.41) dimostrano che CRYSTALS Dilithium è il candidato con i tempi di firma minori e un'ottima soluzione anche nei tempi di verifica della firma.

CRYSTALS Dilithium ha delle prestazioni superiori (in termini di complessità temporale) a FALCON principalmente per due aspetti:

1. FALCON include molte operazioni che coinvolgono numeri *floating point*, per cui l'esecuzione in queste fasi risulta più onerosa.

2. CRYSTALS Dilithium non riduce al minimo le dimensioni delle chiavi e delle firme, cosicché non ci siano fasi di compressione e espansione dei dati aggiuntive prima della vera e propria elaborazione.

Ovviamente non c'è una scelta giusta o errata nelle implementazioni di FALCON e CRYSTALS Dilithium, tuttavia è interessante rilevare come le differenze concettuali e implementative si riflettano effettivamente nei risultati.

SPHINCS+ purtroppo è svantaggiato in questo tipo di competizioni: essendo un algoritmo *Hash Based* le procedure di firma e verifica coinvolgono spesso fasi di *hashing* che sono lente per loro natura. Per questo SPHINCS+, in tutte le sue versioni, risulta sempre almeno un ordine di grandezza più lento rispetto ai candidati appartenenti allo stesso livello di sicurezza.

Come per il confronto tra FALCON e CRYSTALS Dilithium, anche SPHINCS+ ha dei vantaggi: innanzitutto una forte dimostrazione teorica della sua sicurezza (poiché si basa su assunzioni matematiche *semplici* rispetto ai *Lattice Based*), alta compatibilità con diversi tipi di hardware e un utilizzo di memoria ridotto. Questo lo rende adatto ad essere eseguito nei dispositivi con processori ARM oppure nei dispositivi IoT (*Internet of Things*) che mancano della compatibilità con le ottimizzazioni AVX2. Difatti SPHINCS+ possiede molte versioni rispetto agli altri candidati: ciascuna variante porta al suo interno tecniche diverse per eseguire gli stessi procedimenti oppure ottimizzazioni diverse.

La considerazione fondamentale che è possibile estrapolare dai precedenti grafici riguarda i tempi di esecuzione dei nuovi algoritmi PQC: essi sono conformi alla complessità temporale dei sistemi di firma digitale attualmente utilizzati. I tempi di RSA per le operazioni di firma e verifica sono pienamente compatibili con i tempi di questi candidati. Quest'ultimo aspetto dimostra ancora una volta che i candidati considerati sono maturi e pronti per la standardizzazione, come definito dal NIST nell'ultimo report [10].

6

Conclusioni

Nel capitolo precedente, attraverso grafici e considerazioni, sono stati evidenziati i punti di forza e gli aspetti da migliorare per ciascuno dei candidati. In generale, tutte e tre si sono dimostrate ottime soluzioni poiché offrono elevate prestazioni e sono mature dal punto di vista della sicurezza, almeno per quanto riguarda le conoscenze attuali.

Il NIST ha indicato che i sistemi più adatti per essere standardizzati e sostituire quelli attuali sono i *Lattice Based*. CRYSTALS Dilithium e FALCON offrono dimensioni delle chiavi e delle firme in linea con i sistemi attuali, oltre a tempi di esecuzione molto rapidi [10].

Tra i due, il NIST mostra una preferenza per CRYSTALS Dilithium poiché, come rilevato da altre fonti e dai test sulle prestazioni, sembra essere meno vulnerabile agli attacchi *side channel*, diventati oggetto di numerosi studi negli ultimi anni. Inoltre, CRYSTALS Dilithium è considerato più *semplice*: a differenza di FALCON, CRYSTALS Dilithium non utilizza i calcoli in virgola mobile nel suo *core*. Queste operazioni, oltre ad essere più onerose in termini di risorse, introducono un livello di imprecisione che, negli hardware più limitati, possono portare a fallimenti involontari nelle operazioni di firma e verifica [10]. D'altra parte, FALCON si distingue per la riduzione della dimensione dei messaggi, un aspetto importante quando si tratta di trasmissione di comunicazioni tramite le reti internet.

Per l'insieme di questi aspetti, sono di recente pubblicazione i primi standard del NIST per alcuni algoritmi di crittografia e altri di firma digitale a chiave asimmetrica: tra questi ci sono CRYSTALS Dilithium e SPHINCS+. FALCON necessiterà di maggiore tempo per la standardizzazione [12].

È importante sottolineare che, al termine del secondo *round*, SPHINCS+ era considerato tra i finalisti alternativi per la firma digitale. Il suo successo, che lo ha portato a diventare uno degli algoritmi in fase di standardizzazione, è dovuto a due motivazioni principali:

1. Il NIST ha espresso la volontà di standardizzare algoritmi di firma digitale basati su problemi e teorie quanto più diverse possibile.

2. Altri algoritmi candidati finalisti hanno subito delle rivalutazioni di sicurezza, tra cui *Rainbow* [13].

Il primo punto è fondamentale: se il NIST decidesse di proseguire solo con algoritmi *Lattice Based*, esisterebbe il rischio che in futuro venga individuata una vulnerabilità comune a tutti questi sistemi, compromettendo tutte le comunicazioni basate su tali tecnologie.

La scelta di includere anche SPHINCS+ tra i finalisti del terzo *round* porta quindi ad una maggiore diversificazione e sicurezza nelle future comunicazioni, dato che si tratta di un algoritmo *Hash Based*.

Per questo motivo il *NIST Post-Quantum Cryptography Standardization Process* [9] non si fermerà al terzo *round* ma proseguirà nella ricerca di altri algoritmi di firma digitale da standardizzare, ponendo maggiore attenzione agli algoritmi più unici, come quelli *Isogeny Based* o *Code Based*.

Un altro aspetto che il NIST cercherà nei futuri candidati è la capacità di operare anche a livelli di sicurezza più bassi. Attualmente, molte delle operazioni di crittografia e di firma digitale vengono eseguite con tecnologie che privilegiano l'efficienza in termini di tempi di esecuzione e risorse, come RSA 1024 o RSA 2048. Non tutti i settori sono pronti ad abbandonare le prestazioni offerte da questi sistemi per garantire una maggiore sicurezza.

Per proseguire con il processo di standardizzazione, il NIST è entrato in contatto diretto con i team di ricerca finalisti, al fine di sviluppare sistemi di firma digitale con un'interfaccia pubblica comune (API, *Application Programming Interface*) il più compatibile possibile con i protocolli attuali. Tra gli obiettivi c'è anche la ricerca di più *parameter sets* ottimizzati che permettano agli algoritmi di raggiungere i livelli di sicurezza più richiesti mantenendo prestazioni ottimali [10].

6.1 Altre metriche di performance e limitazioni

I *performance test* effettuati per questo progetto offrono una visione limitata, seppur sufficiente, delle caratteristiche da valutare per un algoritmo di firma digitale quantum resistant. Diversi aspetti non sono stati trattati per insufficienza di risorse, difficoltà tecniche o mancanza di fondamenti teorici.

I test di dimensione degli output (chiavi e firme) e dei tempi di esecuzione sono stati eseguiti in ambienti *general purpose*, simili a quelli in cui questi sistemi verranno effettivamente utilizzati, ma utilizzando poca varietà di hardware. Tuttavia, gli stessi test

hanno dimostrato che le proporzioni dei risultati restano consistenti su hardware differenti, permettendo di prevedere le prestazioni confrontandoli con quelli menzionati nella ricerca.

Tramite l'utilizzo di Windows Subsystem for Linux (WSL) è stato fissato ad un valore unico il tasso della frequenza di clock dell'ambiente Ubuntu, per rendere i tempi dei vari dispositivi maggiormente comparabili: il numero di istruzioni eseguite al secondo per l'ambiente virtualizzato era fisso. Purtroppo sono presenti fonti di errore imprevedibili dovute ad operazioni in background che, messe in esecuzione automaticamente nel sistema operativo primario o virtualizzato, possono aver influenzato i risultati finali. Per ridurre l'effetto di questa eventualità sono stati eseguiti più test ripetuti i cui risultati sono stati mediati.

Uno degli aspetti non considerati sono le esecuzioni su CPU con architettura ARM oppure su dispositivi di tipo IoT: in generale l'esecuzione di test su questi dispositivi risulta molto complessa poiché richiede una fase di compilazione ad hoc. Tuttavia, è proprio su questi dispositivi che è possibile individuare più facilmente alcune vulnerabilità ai *side channel attacks*, come per FALCON, in cui è stata sfruttata l'analisi delle frequenze elettromagnetiche su Cortex-M4 [24].

Una metrica rilevante per valutare ciascun candidato, menzionata in alcuni report del NIST, è il tempo di trasmissione dell'output del processo di firma. Esso non è stato misurato direttamente nei *performance test*, tuttavia l'analisi delle dimensioni delle chiavi e delle firme ha comunque permesso di sviluppare delle considerazioni. È evidente che, a parità di bitrate di trasmissione o ricezione, gli algoritmi con i tempi (o costi) di trasmissione inferiori sono quelli con l'output più compatto, come FALCON.

Infine, un ultimo aspetto non considerato nella ricerca ma individuabile nei report del NIST è l'utilizzo di memoria RAM (*Random Access Memory*) e ROM (*Read Only Memory*) necessaria ai vari algoritmi per il loro utilizzo. I test del NIST evidenziano che i migliori risultati sono raggiunti da CRYSTALS Dilithium, che può essere eseguito su sistemi con circa 9 KiB di RAM e 8 KiB di ROM. Questo tipo di analisi sono piuttosto rilevanti per comprendere in maniera semplice le compatibilità dei vari candidati con gli hardware più limitati [10].

Le tecnologie che fanno affidamento ad hardware *special purpose* o IoT sono molto diffuse, quindi, per la standardizzazione di FALCON e SPHINCS+, è necessario trovare modi per ridurre le risorse richieste, mantenendo comunque l'operatività delle funzioni.

6.2 Il futuro processo di migrazione

Nel contesto della crittografia post-quantum (PQC), è fondamentale prendere misure preventive per proteggere i dati che richiedono una sicurezza a lungo termine. Tali dati devono essere messi al sicuro fin da ora per evitare la minaccia degli attacchi di *Retrospective Decryption*, nei quali un avversario può raccogliere informazioni crittografate oggi per decrittarle in futuro utilizzando computer quantistici. Sono due i principali approcci per garantire la sicurezza: il primo consiste nell'implementare sistemi ibridi che combinano tecnologie crittografiche pre-quantum con schemi post-quantum. Questa strategia può essere adottata anche prima che gli algoritmi post-quantum siano completamente standardizzati, offrendo un livello di protezione aggiuntivo [7]. Il secondo approccio prevede il rafforzamento dei sistemi di crittografia attualmente in uso, aumentando la complessità degli schemi basati su chiave simmetrica, che risultano meno suscettibili agli attacchi dei computer quantistici. Questa tecnica di mitigazione rappresenta una soluzione praticabile nel breve termine.

La continuità del servizio è un aspetto critico durante il processo di migrazione verso la crittografia post-quantum. I sistemi ibridi, che applicano in serie tecnologie diverse, non solo aumentano la sicurezza dei sistemi attuali ma agevolano anche la transizione verso nuovi standard. La loro implementazione potrebbe infatti favorire il processo di standardizzazione e l'adozione diffusa dei sistemi PQC, offrendo una piattaforma di prova per valutare la robustezza degli algoritmi post-quantum in ambienti reali.

In generale, l'applicazione di due schemi crittografici in maniera combinata risulta più sicura rispetto all'uso indipendente di ciascuno schema. Questo principio rende i sistemi ibridi una valida opzione per il futuro, poiché aumentano la resilienza contro possibili compromissioni. Tuttavia, una delle principali sfide su cui la ricerca si sta concentrando riguarda lo sviluppo di *combiners* efficienti e sicuri, capaci di integrare diverse tecnologie crittografiche senza compromettere le prestazioni o la sicurezza.

L'adozione di nuovi sistemi crittografici potrebbe non essere sempre fattibile, specialmente per tecnologie meno accessibili e più particolari, come quelle impiegate nei satelliti [5]. In aggiunta, la natura non sempre open-source delle ricerche sui computer quantistici comporta che alcuni gruppi di scienziati potrebbero aver ottenuto progressi avanzati in questo campo, di cui non si sarà a conoscenza fino alla pubblicazione dei risultati.

Un ulteriore aspetto critico della migrazione è la necessità di una transizione coordinata. Poiché molti attori (o *parties*) devono utilizzare gli stessi sistemi crittografici per garantire la compatibilità e la sicurezza, il passaggio ai nuovi schemi PQC richiede una migrazione simultanea. Nonostante queste sfide, sono già in corso numerosi test per valutare l'implementazione pratica della crittografia post-quantum. Un esempio significativo è il progetto *Google New Hope*, che utilizza schemi PQC per proteggere parte delle co-

municazioni del noto browser Google Chrome, nello specifico verso i siti che supportano questi tipo di tecnologia (principalmente i servizi Google) [36].

L'obiettivo finale di questa transizione verso la crittografia post-quantum rimane quello di bilanciare praticità, sicurezza, prestazioni e costi. La minimizzazione dei costi legati alla rete, alle risorse e al consumo energetico è cruciale per una migrazione efficace e sostenibile. I test di sostituzione degli schemi attuali con protocolli PQC stanno mostrando risultati promettenti, suggerendo che la transizione verso un ambiente crittografico più sicuro e post-quantum potrebbe essere realizzata con successo entro la fine del decennio [10].

7

Bibliografia

[1] EUR-Lex, *Regulation (EU) 2024/1183 of the European Parliament and of the Council of 11 April 2024 amending Regulation (EU) No 910/2014 as regards establishing the European Digital Identity Framework*, https://eur-lex.europa.eu/eli/reg/2024/1183/oj, OJ L 1183, 30 April 2024, 2024.

[2] ETSI, «Electronic Signatures and Infrastructures (ESI); Procedures for Creation and Validation of AdES Digital Signatures; Part 1: Creation and Validation,» European Telecommunications Standards Institute, rapp. tecn. ETSI EN 319 102-1 V1.4.1, 2024. indirizzo: https://www.etsi.org/deliver/etsi_en/319100_319199/31910201/01.04.01_60/en_31910201v010401p.pdf.

[3] ETSI, «ETSI TS 102 918 V1.3.1 (2013-06): Electronic Signatures and Infrastructures (ESI); Associated Signature Containers (ASiC),» ETSI, rapp. tecn. 102918, ver. 1.3.1, 2013. indirizzo: https://www.etsi.org/deliver/etsi_ts/102900_102999/102918/01.03.01_60/ts_102918v010301p.pdf.

[4] ETSI, «ETSI TS 319 142-1 V1.2.1 (2024-01): Electronic Signatures and Infrastructures (ESI); PAdES digital signatures,» ETSI, rapp. tecn. 319142-1, ver. 1.3.1, 2024. indirizzo: https://www.etsi.org/deliver/etsi_en/319100_319199/31914201/01.02.01_60/en_31914201v010201p.pdf.

[5] D. J. Bernstein e T. Lange, «Post-quantum cryptography,» *Nature*, vol. 549, n. 7671, pp. 188–194, 2017. DOI: 10.1038/nature23461.

[6] S. Mitra, B. Jana, S. Bhattacharya, P. Pal e J. Poray, «Quantum cryptography: Overview, security issues and future challenges,» in *2017 4th International Conference on Opto-Electronics and Applied Optics (Optronix)*, IEEE, 2017, pp. 1–7.

[7] European Union Agency for Cybersecurity (ENISA), «Post-Quantum Cryptography: Current State and Quantum Mitigation,» European Union Agency for Cybersecurity (ENISA), rapp. tecn., 2021, Accessed: 2024-07-27. indirizzo: https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation.

[8] National Institute of Standards and Tecnology, *About NIST*, https://www.nist.gov/about-nist, Accessed: 2024-07-10, 2009.

[9] National Institute of Standards and Technology, *NIST Post-Quantum Cryptography Standardization Process*, https://csrc.nist.gov/Projects/Post-Quantum-Cryptography, Accessed: 2024-07-27, 2013.

[10] G. Alagic, G. Alagic, D. Apon et al., «Status report on the third round of the NIST post-quantum cryptography standardization process,» National Institute of Standards e Technology, rapp. tecn., 2022. DOI: 10.6028/NIST.IR.8413. indirizzo: https://www.nist.gov/publications/status-report-third-round-nist-post-quantum-cryptography-standardization-process.

[11] S. Suhail, R. Hussain, A. Khan e C. S. Hong, «On the role of hash-based signatures in quantum-safe internet of things: Current solutions and future directions,» *IEEE Internet of Things Journal*, vol. 8, n. 1, pp. 1–17, 2020.

[12] National Institute of Standards and Technology. «NIST Releases First 3 Finalized Post-Quantum Encryption Standards.» Accessed: 2024-09-05. (2024), indirizzo: https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards.

[13] Rainbow PQC research team, *PQC Rainbow*, https://www.pqcrainbow.org/, Accessed: 2024-07-24, 2020.

[14] Sphincs Plus PQC research team, *Sphincs Plus: Stateless hash-based signatures*, https://sphincs.org/, Accessed: 2024-07-24, 2020.

[15] N. I. of Standards e Technology, «Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process,» National Institute of Standards e Technology, rapp. tecn., 2016, Accessed: 2024-08-16. indirizzo: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[16] National Institute of Standards and Technology, *NIST PQC API resources*, https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/example-files/api-notes.pdf, Accessed: 2024-07-27, 2016.

[17] E. Barker e Q. Dang, «Recommendation for Key Management: Part 1 - General,» National Institute of Standards e Technology, rapp. tecn. NIST SP 800-57 Part 1 Rev. 5, 2020, Accessed: 2024-08-16. DOI: 10.6028/NIST.SP.800-57pt1r5. indirizzo: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf.

[18] CRYSTALS Dilithium PQC research team, *CRYSTALS Dilithium: Cryptographic Suite for Algebraic Lattices*, https://pq-crystals.org/dilithium/, Accessed: 2024-07-24, 2020.

[19] CRYSTALS PQC, *CRYSTALS: Cryptographic Suite for Algebraic Lattices*, https://pq-crystals.org/, Accessed: 2024-07-24, 2020.

[20] L. Ducas, E. Kiltz, T. Lepoint et al., «Crystals-dilithium: A lattice-based digital signature scheme,» *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018. DOI: 10.13154/tches.v2018.i1.238-268.

[21] L. Ducas, E. Kiltz, T. Lepoint et al., «Crystals-dilithium: Algorithm Specifications and Supporting Documentation,» CRYSTALS-Dilithium Research Team, rapp. tecn., 2021. indirizzo: https://pq-crystals.org/dilithium/resources.shtml.

[22] FALCON PQC research team, *FALCON: Fast Fourier Lattuce-based Compact Signatures over NTRU*, https://falcon-sign.info/, Accessed: 2024-07-24, 2020.

[23] P.-A. Fouque, J. Hoffstein, P. Kirchner et al., «Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU,» FALCON Research Team, rapp. tecn., 2020. indirizzo: https://falcon-sign.info/falcon.pdf.

[24] E. Karabulut e A. Aysu, «Falcon down: Breaking falcon post-quantum signature scheme through side-channel attacks,» in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2021, pp. 691–696. DOI: 10.1109/DAC18074.2021.9586131.

[25] J.-P. Aumasson, D. J. Bernstein, W. Beullens et al., *SPHINCS+ Modifications for Round 3 Submission to the NIST Post-Quantum Project*, Included in the third submission package to NIST, 2020.

[26] D. J. Bernstein, D. Hopwood, A. Hülsing et al., «SPHINCS+: A Stateless Hash-Based Signature Scheme,» in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*, New York, NY, USA: Association for Computing Machinery, 2019, pp. 2129–2146. DOI: 10.1145/3319535.3363229. indirizzo: \url{https://sphincs.org/data/sphincs+-paper.pdf}.

[27] D. J. Bernstein, D. Hopwood, A. Hülsing et al., *SPHINCS+ (Version 3.1): Submission to the NIST Post-Quantum Cryptography Project*, https://sphincs.org/data/sphincs+-r3.1-specification.pdf, Accessed: 2024-08-18, 2023.

[28] S. Sun, R. Zhang e H. Ma, «Efficient parallelism of post-quantum signature scheme SPHINCS,» *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, n. 11, pp. 2542–2555, 2020. DOI: 10.1109/TPDS.2020.2995562.

[29] CRYSTALS-Dilithium Team, *CRYSTALS-Dilithium*, https://github.com/pq-crystals/dilithium, Accessed: 2024-07-10, 2023.

[30] CRYSTALS-FALCON Team, *CRYSTALS-FALCON*, https://falcon-sign.info/impl/falcon.h.html, Accessed: 2024-07-10, 2023.

[31] SPHINCS+ Team, *SPHINCS+*, https://github.com/sphincs/sphincsplus, Accessed: 2024-07-10, 2023.

[32] Tomas Lovato, *GitHub - Analisi di algoritmi di firma digitale post-quantum*, https://github.com/LovatoTomas/Analisi_di_algoritmi_di_firma_digitale_post-quantum, 2024.

[33] O. Q. S. project, *liboqs: C library for quantum-safe cryptography*, https://github.com/open-quantum-safe/liboqs, Accessed: 2024-08-20, 2024.

[34] Foonly, *RSA Public Key and Private Key Lengths*, Security Stack Exchange, Accessed: 2024-08-02, 2015. indirizzo: https://security.stackexchange.com/questions/90169/rsa-public-key-and-private-key-lengths.

[35] ETSI, «ETSI TS 119 312 V1.4.3 (2022-01): Electronic Signatures and Infrastructures (ESI); Cryptographic Suites,» ETSI, rapp. tecn. 119312, ver. 1.4.3, 2022. indirizzo: https://www.etsi.org/deliver/etsi_ts/119300_119399/119312/01.04.03_60/ts_119312v010403p.pdf.

[36] Wikipedia contributors. «NewHope.» Accessed: 2024-09-03. (2024), indirizzo: https://en.wikipedia.org/wiki/NewHope.