UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Triennale in Informatica

Crittografia Post Quantistica: i nuovi standard di firma digitale

Relatore:

Prof.ssa: Anna Bernasconi

Candidato:

Ginevra Maoro

ANNO ACCADEMICO 2023/2024

Introduzione

La crittografia a chiave pubblica ricopre al mondo d'oggi un ruolo di evidente importanza poiché garantisce la protezione di dati sensibili trasmessi ogni secondo sulle reti di comunicazione di tutto il mondo. Con la minaccia dell'avvento dei computer quantistici, che sarebbero in grado di risolvere in tempo polinomiale i problemi su cui si basa attualmente la sicurezza dei crittosistemi a chiave pubblica, è nata la necessità di trovare nuovi sistemi crittografici sicuri nei confronti della computazione quantistica.

All'interno di questa tesi ci concentreremo in particolare su due nuovi schemi di firma digitale, che utilizzano la crittografia a chiave pubblica, sicuri nei confronti della computazione quantistica: Dilithium e SPHINCS+. Essi sono due dei vincitori del programma di standardizzazione del NIST per la crittografia post quantistica nell'ambito della firma digitale. Cercheremo inoltre di dare un' introduzione semplice e comprensibile delle complesse basi matematiche su cui si basano questi due nuovi schemi di firma, per poi descriverne il funzionamento. Infine mostreremo i risultati di alcuni test eseguiti su Dilithium, SPHINCS+ e ECDSA, quest'ultimo è uno degli attuali schemi di firma non resistente alla computazione quantistica. Questi test sono stati eseguiti in modo da poter confrontare tra loro i due nuovi schemi di firma, sicuri nei confronti della computazione quantistica, ma anche per fare un confronto con lo standard attuale per la firma digitale, non sicuro nei confronti della computazione quantistica.

La tesi è organizzata nel seguente modo: nel primo capitolo effettueremo una introduzione ai problemi su cui si basa oggi la sicurezza della crittografia a chiave pubblica, parleremo anche degli algoritmi quantistici che minacciano la sicurezza sia della crittografia a chiave pubblica che a chiave privata. Parleremo del programma di standardizzazione condotto dal NIST a partire dal 2016, per definire nuovi schemi nell'ambito della crittografia a chiave pubblica che siano resistenti, e quindi sicuri, alla computazione quantistica. Questo programma, come discuteremo, non riguarderà la crittografia a chiave privata perché, con qualche accorgimento, vedremo essere resistente nei confronti della computazione quantistica.

Nel secondo capitolo, spigheremo cosa è una firma digitale, quali sono le sue caratteristiche e i requisiti che deve avere. Analizzeremo quindi lo standard attuale di firma digitale, basato sulla crittografia su curve ellittiche e vulnerabile alla computazione quantistica: ECDSA.

Nel terzo capitolo introdurremo la teoria e i problemi dietro la crittografia basata

sui reticoli, che la rendono resistente alla computazione quantistica, su cui si basa uno dei vincitori del processo di standardizzazione del NIST nell'ambito delle firme digitali: CRYSTAL-Dilithium.

Il quarto capitolo introdurrà le firme basate su hash, illustrando alcuni schemi di firma, grazie ai quali è stato possibile elaborare l'altro degli schemi di firma digitale vincitore del processo di standardizzazione del NIST: SPHINCS+.

L'ultimo capitolo è stato riservato alla valutazione sperimentale dei tre schemi di firma approfonditi all'interno di questa tesi, utilizzando librerie che hanno permesso di testarli e raccogliere i risultati per effettuarne il confronto in termini di prestazioni in tempo, dimensioni delle chiavi, firma e livello di sicurezza.

Indice

Capitolo 1

Introduzione alla Crittografia Post Quantistica

1.1 Crittografia al mondo d'oggi

Negli ultimi trenta anni la **crittografia a chiave pubblica**, chiamata anche **crittografia asimmetrica**, è diventata una componente indispensabile nella nostra infrastruttura digitale di comunicazione globale. Le reti di questa infrastruttura supportano una pletora di applicazioni che sono importanti per la nostra economia, la nostra sicurezza e il nostro modo di vivere, come telefoni cellulari, commercio su Internet, reti sociali e cloud computing. In un mondo così connesso, la capacità di individui, aziende e governi di comunicare in modo sicuro è di massima importanza [19].

La crittografia a chiave pubblica è nata dal tentativo di affrontare i problemi della **crittografia simmetrica**, nella quale mittente e destinatario usano la stessa chiave segreta per cifrare e decifrare le informazioni. Il primo problema è quello della distribuzione sicura delle chiavi; il secondo problema è quello di garantire la non ripudiabilità, cioè di accertare l'identità del mittente del messaggio, che vedremo sarà possibile con l'uso delle firme digitali.

Molti dei nostri protocolli di comunicazione a chiave pubblica, che utilizzano due chiavi, una privata e una pubblica, si basano principalmente su tre funzionalità crittografiche fondamentali:

- **Cifratura e Decifrazione**: usate per garantire la riservatezza delle comunicazioni. Il mittente cifra un messaggio con la chiave pubblica del destinatario, e il destinatario decifra il messaggio con la propria chiave privata.

- **Firme digitali**: usate per garantire autenticità, integrità e non ripudiabilità dei dati. Il mittente usa la propria chiave privata nel processo di firma del messaggio, mentre il destinatario usa la chiave pubblica del mittente per verificare la firma sul messaggio.

- **Scambio di chiavi**: due parti cooperano per scambiarsi in modo sicuro una chiave simmetrica per permettere l'uso successivo di sistemi crittografici simmetrici [52][26].

Attualmente, queste funzionalità sono implementate principalmente utilizzando lo scambio di chiavi Diffie-Hellman (DH), e l'RSA (dal nome degli autori Rivest-Shamir-Adleman) un sistema crittografico versatile in quanto fornisce le operazioni necessarie per abilitare la cifratura, la decifrazione, le firme digitali e lo scambio di chiavi all'interno di un framework comune. La sicurezza di questi dipende dalla difficoltà di risolvere certi problemi di Teoria dei Numeri come **il problema della fattorizzazione intera** o **il problema del logaritmo discreto.**[19] [26]. Negli ultimi anni si sono inoltre affermati i sistemi a chiave pubblica basati sulle curve ellittiche, che garantiscono gli stessi livelli di sicurezza dei protocolli DH e RSA, con chiavi più corte.

1.1.1 Problema della fattorizzazione

Definizione 1 (**Fattorizzazione di interi**)**.** *Nella Teoria dei Numeri, per fattorizzazione di interi si intende la scomposizione di un numero intero in un insieme di divisori interi non banali che, moltiplicati fra loro, diano il numero di partenza.*

Il **Teorema Fondamentale dell'Aritmetica** afferma che ogni intero positivo può essere scomposto in un prodotto di **numeri primi**, cioè numeri che non possono essere ulteriormente fattorizzati in altri interi maggiori di 1, e che questo prodotto è **unico** se si trascura l'ordine dei fattori.

La fattorizzazione è un processo algoritmico di successive divisioni per ottenere i singoli fattori, ma quando il numero da fattorizzare è sufficientemente grande, non si conosce alcun algoritmo non quantistico efficiente, cioè di costo in tempo polinomiale nella dimensione del numero da fattorizzare. È proprio la difficoltà nella risoluzione di questo problema che viene sfruttata per garantire la sicurezza del crittosistema RSA.

In particolare nel caso dell'RSA, un messaggio viene cifrato sfruttando la difficoltà di fattorizzare un numero semiprimo, cioè un numero che è prodotto di due interi primi distinti e non noti. Di fatti la chiave pubblica all'interno di questo crittosistema è un numero N ottenuto moltiplicando due numeri primi p e q molto grandi che restano segreti. Se calcolare N da p e q è semplice, l'operazione inversa cioè quella di fattorizzare N per trovare p e q è molto più difficile ed inoltre questa soluzione, come ci dice il Teorema Fondamentale dell'Aritmetica, è unica se p e q sono primi.

1.1.2 Problema del logaritmo discreto

Nell'aritmetica modulare i logaritmi discreti sono il corrispettivo dei logaritmi ordinari. Prima di dare la definizione del problema ricordiamo la definizione di gruppo ciclico:

Definizione 2 (**Gruppo ciclico**). *Un gruppo G è ciclico se esiste un elemento $g \in G$ (detto generatore) tale che G è l'insieme delle potenze di g ad esponente intero $G = \{g^n : n \in \mathbb{Z}\}$.*

Qui abbiamo usato la nozione moltiplicativa, se vogliamo usare quella additiva invece di potenze si parla di multipli e quindi $G = \{ng : n \in \mathbb{Z}\}$. Quest'ultima definizione ci tornerà comoda quando parleremo del protocollo ECDSA di firma digitale che utilizza le curve ellittiche.

Definizione 3 (**Problema del Logaritmo Discreto**). *Sia G un gruppo ciclico e sia $g \in G$ un generatore di G, dato un elemento $h \in G$, trovare un intero k tale che $h = g^k$. L'intero $k \equiv \log_g h$ è il logaritmo discreto.*

Si osservi che il fatto che G sia un gruppo ciclico garantisce che per ogni h esiste un intero k valido. In crittografia il problema del logaritmo discreto sul gruppo moltiplicativo $\mathbb{Z}_p^*$ degli interi minori di p e coprimi con p, con p numero primo, risulta essere molto utile. Ricordiamo che se vogliamo effettuare l'operazione di moltiplicazione all'interno di questo gruppo, o una qualsiasi altra operazione aritmetica, possiamo procedere con l'algoritmo usuale e ridurre il risultato, o i fattori intermedi del calcolo, modulo p non appena possibile .
Consideriamo per esempio il problema del logaritmo discreto su $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$, questo gruppo moltiplicativo ha due generatori 3 e 5. Scegliendo 5 come generatore, possiamo verificare che tutti gli elementi di $\mathbb{Z}_7^*$ si possono esprimere usando potenze intere successive di 5. Risulta infatti $5^0 \mod 7 \equiv 1$, $5^1 \mod 7 \equiv 5$, $5^2 \mod 7 \equiv 4$, $5^3 \mod 7 \equiv 6$, $5^4 \mod 7 \equiv 2$, $5^5 \mod 7 \equiv 3$; si generano così tutti gli elementi di $\mathbb{Z}_7^*$. Quindi, il problema del logaritmo discreto su $\mathbb{Z}_7^*$ con generatore 5 è:

$$\textit{Dato } h \in \mathbb{Z}_7^* \textit{ trovare } k \textit{ tale che } 5^k \mod 7 \equiv h.$$

Possiamo osservare che calcolare $(g^k \mod p)$ è considerato un problema trattabile, per il quale si hanno a disposizione algoritmi polinomiali nella dimensione dell'istanza di input. In particolare viene utilizzato il metodo delle *quadrature successive*, con il quale per calcolare $(g^k \mod p)$ sono richieste un numero di operazioni polinomiali nel numero di cifre binarie dell'esponente k, per un costo complessivo dell'operazione pari a $O((\log k)^3)$. Viceversa, calcolare la funzione inversa, ossia dato h trovare l'elemento k per cui $g^k \equiv h \mod p$, è un problema difficile, per il quale non sono noti algoritmi non quantistici che lo risolvono in tempo polinomiale. Questo è il motivo per cui il problema del logaritmo discreto è alla base di numerosi schemi crittografici come lo scambio di chiavi Diffie-Hellman.

1.1.3 Complessità computazionale

Attualmente in crittografia alcuni problemi computazionali della classe di complessità computazionale NP svolgono un ruolo importante. Prima di definire la classe NP definiamo la classe $NTIME$:

Definizione 4 (**NTIME**). *Se una macchina di Turing non deterministica decide il problema I in tempo f, allora $I \in NTIME(f)$*

Informalmente possiamo definire una Macchina di Turing non deterministica come un modello teorico di calcolo le cui regole di comportamento specificano più di un'azione possibile quando la macchina si trova in uno specifico stato. Cioè, lo stato successivo di una Macchina di Turing non deterministica non è completamente determinato dalla sua azione e dal simbolo corrente che vede, a differenza di una macchina di Turing deterministica.

A questo punto possiamo definire la classe NP come segue

Definizione 5 (**NP**). *NP è la classe dei problemi decidibili in tempo non deterministico polinomiale, definita come:*

$$NP = \bigcup_{k \geq 1} NTIME(n^k)$$

La sicurezza della prima generazione di sistemi crittografici a chiave pubblica di successo, sviluppati negli anni '70, era basata su problemi matematici come la **fattorizzazione in numeri primi** e il **logaritmo discreto** che si ipotizza appartengano alla sottoclasse dei problemi NP-*intermedi* di NP, chiamata classe NPI. Questa sottoclasse è costituita da quei problemi che si ritiene non possano essere risolti in tempo polinomiale su una macchina di Turing deterministica, e quindi non appartengono alla classe P, ma allo stesso tempo non sono difficili come i problemi più difficili in NP, cioè quelli appartenenti alla sottoclasse NPC formata dai problemi NP-completi:

$$NPI = NP \setminus (P \cup NPC)$$

(si veda la Figura 1.1).

Sulla base dei problemi NP-completi si definiscono i problemi NP-hard: nella Teoria della Complessità, i problemi NP-hard da "non determistic polynomial-time hard problem" (in italiano chiamati anche NP-difficili o NP-ardui) sono una classe di problemi che può essere definita informalmente come la classe dei problemi *almeno* difficili come i più difficili problemi della classe di complessità NP. Formalmente:

Definizione 6 (**NP-hard**). *Un problema H è NP-hard se e solo se ogni problema $\Pi \in NP$ è polinomialmente riducibile ad H, ovvero $\forall \Pi \in NP$, $\Pi \leq_T H$.*

Da questa definizione si ricava che i problemi NP-hard sono non meno difficili dei problemi NP-completi.

In seguito alla scoperta dell'algoritmo di Shor, che vedremo tra poco, è diventato chiaro che almeno alcuni problemi NP-intermedi ammettono soluzioni efficienti su computer quantistici [26].

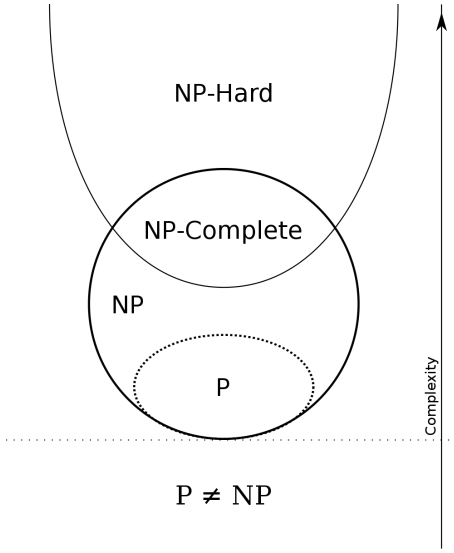


Figura 1.1: Classi di complessità dei problemi NP-completi, NP-hard e NP-intermedi [50].

1.2 La minaccia dei computer quantistici

In un computer quantistico la logica che determina i processi interni è basata sulle leggi della meccanica quantistica. L'informazione al suo interno è rappresentata utilizzando non bit classici, ma *qubit*, ovvero bit quantistici. Un qubit, a differenza di un bit classico, presenta due proprietà fondamentali per la computazione quantistica: *Sovrapposizione* e l'*Entanglement*. Esse permettono a un computer quantistico di ricercare milioni di soluzioni simultaneamente anziché sequenzialmente, portando a una crescita potenzialmente **esponenziale** della potenza computazionale.

1.2.1 Algoritmo di Shor

Nel 1994, l'informatico teorico Peter Shor ha esposto alla comunità scientifica un algoritmo per risolvere il **problema della fattorizzazione** in **tempo polinomiale** su un computer quantistico. Fu riconosciuto come il primo algoritmo quantistico che offriva un'accelerazione *esponenziale* rispetto agli algoritmi classici per risolvere il problema della fattorizzazione.

Il lavoro di Shor ha avuto un enorme impatto nel campo della crittografia perché la disponibilità di un computer quantistico, su cui possa essere eseguito l'algoritmo di Shor, metterebbe fuori gioco tutti i cifrari a **chiave pubblica** oggi più comunemente usati. La più immediata conseguenza si avrebbe sul cifrario RSA, che come abbiamo già detto, fonda la propria sicurezza nell'assunzione che il problema della fattorizzazione dei numeri interi sia un problema computazionalmente impossibile da risolvere in tempo ragionevole per un computer classico. Ricordiamo infatti che in questo cifrario la chiave pubblica contiene un intero ottenuto come prodotto di due primi e la chiave privata si può calcolare solo conoscendo questi due fattori.

Adattamenti dell'algoritmo di Shor permettono di risolvere in tempo polinomiale anche il **problema del logaritmo discreto**, su campi finiti e su curve ellittiche, e quindi di compromettere il protocollo DH per lo scambio di chiavi, e di attaccare i cifrari a chiave pubblica basati sulle curve ellittiche, in quanto entrambi basano la propria sicurezza sull'intrattabilità classica del problema del logaritmo discreto [15] [19].

1.2.2 Algoritmo di Grover

Per quanto riguarda i protocolli di crittografia a **chiave privata** e quelli basati sull'uso di **funzioni hash crittografiche**, a livello quantistico la minaccia è rappresentata dall'algoritmo di Grover, il quale però consente accelerazioni computazionali più modeste.

L'algoritmo di Grover permette di eseguire la ricerca di un elemento in una database non strutturato, fornendo un aumento *quadratico* della velocità per gli algoritmi di ricerca quantistica rispetto agli algoritmi di ricerca sui computer classici. Questo significa che, dato un problema di ricerca, se nel caso classico questo richiede un numero N di accessi, nel caso quantistico, con l'algoritmo di Grover, sono sufficienti circa $\sqrt{N}$ accessi quantistici. Quindi, l'algoritmo consente a un attaccante con un computer quantistico di condurre un attacco di forza bruta e cercare nello spazio delle chiavi di un cifrario simmetrico in modo quadraticamente più veloce rispetto a qualsiasi computer classico.

Per esempio, attualmente una chiave AES (Advanced Encryption Standard) di 128 bit viene considerata sicura nei confronti degli attacchi di forza bruta classici; per difendersi invece da un attacco quantistico che utilizza l'algoritmo di Grover, si può mantenere lo stesso livello di sicurezza, impiegando però una chiave di 256 bit. Questo è considerato sicuro perché, anche con i computer quantistici, eseguire 2^{128} operazioni per violare i cifrari è impraticabile nel prossimo futuro.

Pertanto, gli algoritmi crittografici simmetrici esistenti come AES e SHA-256 (Secure Hash Algorithm) possono essere protetti dalla minaccia degli attacchi quantistici, a patto di utilizzare chiavi e hash sufficientemente lunghi (almeno il doppio rispetto alle lunghezze correnti).

Inoltre, è stato dimostrato che un aumento esponenziale della velocità per gli algoritmi di ricerca è impossibile, suggerendo che gli algoritmi simmetrici e le funzioni hash dovrebbero essere sicuri in un'era quantistica [26] [19]. L'impatto dei computer quantistici su larga scala sugli algoritmi crittografici comuni, sia simmetrici che asimmetrici, è riassunto nella Tabella 1.1.

Algoritmo di crittografia	**Tipo**	**Scopo**	**Mitigazione nei confronti dei computer quantistici**
AES	Chiave simmetrica	Cifratura	È necessaria una maggiore lunghezza della chiave
SHA-2, SHA-3	Hash crittografico	Funzione hash	È necessaria una maggiore lunghezza dell'output
RSA	Chiave pubblica	Firma, stabilimento della chiave condivisa	Non è più sicuro, è necessario passare a un algoritmo post quantistico
ECDSA, ECDH (crittografia su curve ellittiche)	Chiave pubblica	Firma, Scambio delle chiavi pubbliche per generare la chiave condivisa	Non è più sicuro, è necessario passare a un algoritmo post quantistico
DSA (crittografia su campi finiti)	Chiave pubblica	Firma, Scambio delle chiavi pubbliche per generare la chiave condivisa	Non è più sicuro, è necessario passare a un algoritmo post quantistico

Tabella 1.1: Impatto della computazione quantistica sugli algoritmi crittografici comuni [19]

1.3 Conseguenze della computazione quantistica

Abbiamo visto che gli impatti più significativi degli algoritmi quantistici si verificano nel contesto della **crittografia a chiave pubblica**, dove l'algoritmo di Shor offre una soluzione in tempo polinomiale al problema della fattorizzazione e del logaritmo discreto. Pertanto, i crittosistemi asimmetrici basati su questi problemi devono essere sostituiti da nuovi schemi di crittografia sicuri rispetto alla computazione quantistica, basati su problemi NP-completi o problemi NP-hard, cioè problemi matematici per cui non sono noti algoritmi risolutivi classici o quantistici efficienti.

1.3.1 Avvento dei computer quantistici

La scoperta che i computer quantistici potrebbero essere utilizzati per risolvere certi problemi più velocemente dei computer classici, ha suscitato grande interesse nella computazione quantistica, e allo stesso tempo, come già accennato, ha avuto un forte impatto sulla crittografia.

In particolare, le domande da porsi sono:

- Quando saranno costruiti computer quantistici su larga scala?
- C'è un modo per resistere sia a un avversario di calcolo quantistico che classico?

La questione di quando sarà costruito un computer quantistico su larga scala è complicata e controversa. Ma sappiamo che essi consentirebbero nuovi attacchi crittografici che non sarebbero possibili utilizzando computer classici. Inoltre questi attacchi potrebbero verificarsi in futuro su dati che vengono trasmessi o archiviati ora. In letteratura, questi attacchi sono noti come *"raccogli ora, decifra dopo"*. Quindi non è sufficiente aspettare fino a quando i sistemi richiesti saranno disponibili, ma è necessario apportare modifiche ora. Pertanto, indipendentemente dal fatto che possiamo stimare il momento esatto dell'arrivo effettivo dell'era della computazione quantistica, dobbiamo iniziare ora a rivalutare la sicurezza crittografica e a preparare i nostri sistemi di sicurezza delle informazioni per essere in grado di resisterle [19].

1.3.2 Cosa è la Crittografia post quantistica

Nel mondo accademico, questa nuova scienza che comprende gli sforzi per identificare e sviluppare schemi crittografici a chiave pubblica implementabili su elaboratori classici, che possano resistere agli attacchi sia da sistemi di calcolo quantistico che classico, e che possano interagire con i protocolli di comunicazione e le reti esistenti, prende il nome di **"Crittografia Post-Quantistica"** o **"Crittografia Sicura al Quantum"**. Da non confondere con la Crittografia Quantistica, che invece utilizza le proprietà della meccanica quantistica per creare un canale di comunicazione sicuro [15].

Attualmente questo è un settore di ricerca attivo, i cui sforzi hanno portato a progressi nella ricerca, aprendo la strada al dispiegamento di crittosistemi post-quantistici nel mondo reale.

Il National Institute of Standards and Technology (NIST), un'agenzia del Dipartimento del Commercio degli Stati Uniti la cui missione è promuovere l'innovazione americana e la competitività industriale, svolge un ruolo unico nella standardizzazione della crittografia post-quantistica, come parte della sua responsabilità nello sviluppo di norme e linee guida per la protezione dei sistemi di informazione. Inoltre una grande comunità internazionale è emersa per affrontare la questione della sicurezza delle informazioni in un futuro di computazione quantistica, nella speranza che la nostra infrastruttura a chiave pubblica possa rimanere intatta utilizzando nuove primitive resistenti al quantum. Il coinvolgimento della comunità è fondamentale affinché gli standard crittografici del NIST siano approvati dall'industria e da altre organizzazioni di standard in tutto il mondo [19].

Famiglie crittografiche per la crittografia post quantistica

Non esiste una singola alternativa agli algoritmi esistenti basati sulla fattorizzazione intera o sui logaritmi discreti. I crittografi hanno proposto una serie di diverse strutture matematiche, che soddisfano i requisiti di difficoltà necessari, come potenziali

candidati per la migrazione ai crittosistemi post quantistici a chiave asimmetrica. Alcune famiglie ben note includono (si consultino [26, 19, 52] per maggiori dettagli):

1. **Crittografia basata su reticoli**: Questa classe di algoritmi si basa sulla difficoltà di problemi come Shortest Vector Problem (SVP) e il Closest Vector Probelm (CVP) nelle strutture a reticolo (o lattice), di cui parleremo più diffusamente nel Capitolo 3. Non esiste un algoritmo quantistico noto per risolvere SVP o CVP per reticoli di grandi dimensioni in tempo polinomiale. Gli schemi più noti basati su reticoli includono NTRU e Learning with Errors (LWE). Si osservi che attaccare questi schemi risulta computazionalmente pesante, ma la loro adozione risulta relativamente semplice in quanto si hanno a disposizione algoritmi efficienti, altamente parallelizzabili, e che richiedono tempi di esecuzione ridotti utilizzando chiavi di grandezza media.

2. **Crittografia basata su codici**: Questo tipo di crittografia si basa sulla difficoltà di decodificare un codice lineare generale ed ha avuto più successo con gli schemi di cifratura piuttosto che in quelli di firma. Inoltre questi schemi sono piuttosto veloci ma richiedono dimensioni della chiave molto grandi. L'esempio più noto è il crittosistema di McEliece.

3. **Crittografia polinomiale multivariata**: Questi schemi si basano sulla difficoltà di risolvere sistemi di equazioni di variabili multiple su un campo finito. La crittografia multivariata ha storicamente avuto più successo come approccio alle firme digitali dove però richiede dimensione della chiave molto grandi.

4. **Crittografia basata su hash**: Questi sono sistemi crittografici che utilizzano solo funzioni hash crittografiche. Sono spesso utilizzati per le firme digitali; la loro sicurezza, anche nei confronti di attacchi quantistici, è ben compresa e consolidata. Molti tra i più efficienti schemi di firma basati su hash hanno lo svantaggio che il firmatario deve mantenere un registro delle chiavi già utilizzate per firmare i precedenti messaggi. Un altro svantaggio è che essi possono produrre solo un numero limitato di firme. Il numero di firme può essere incrementato ma ciò incrementa la dimensione della firma.

5. **Crittografia basata su isogenie**: Questi sistemi si basano sulla difficoltà di alcuni problemi legati alla struttura algebrica delle curve ellittiche.

1.3.3 Programma di standardizzazione del NIST

Riconoscendo l'impatto potenziale del calcolo quantistico sui sistemi crittografici attuali, il National Institute of Standards and Technology (NIST) americano, nel dicembre del 2016, ha avviato un processo di standardizzazione per definire ed analizzare nuovi schemi nell'ambito della crittografia a chiave pubblica che siano resistenti alla computazione quantistica. In particolare, le categorie di tecniche crittografiche oggetto della standardizzazione sono la cifratura a chiave pubblica, il

Key-Encapsulation Mechanism (KEM) e le firme digitali. Al processo, aperto e trasparente, sono stati invitati a partecipare, per ideare e poi esaminare i metodi crittografici proposti, esperti di crittografia di tutto il mondo e gli stakeholder del dominio della sicurezza [26].

Nel novembre 2017, erano stati sottomessi un totale di 82 candidati per essere considerati dal NIST nel processo di standardizzazione, tra di loro 69 soddisfacevano i criteri minimi e i requisiti per la sottomissione al processo e sono stati quindi accettati come partecipanti al 1° round, conclusosi il 30 gennaio 2019.

In seguito alla scrematura del 1° round sono rimasti in gioco 26 candidati, 17 dei quali erano algoritmi di cifratura pubblica e di key-encapsulation, i restanti 9 invece erano per la firma digitale, tutti quanti sono quindi passati al 2° round per ulteriori analisi [7].

Il 2° round si è concluso nel Luglio del 2020 con la scelta dei 15 algoritmi che avrebbero partecipato al 3° round. Tra essi vi erano gli algoritmi più promettenti che si pensava sarebbero stati considerati per la standardizzazione alla fine del 3° round: Classic McEliece, CRYSTALS-Kyber, NTRU e SABER come algoritmi di cifratura pubblica e key-encapsulation, insieme a CRYSTALS-Dilithium, FALCON e Rainbow per la firma digitale. I rimanenti 8 algoritmi passati al 3° round, tra cui SPHINCS+, erano considerati candidati alternativi: nonostante fossero in fase di considerazione per la standardizzazione, era considerato improbabile che ciò sarebbe avvenuto alla fine del 3° round [8].

Durante il 3° round, all'inizio del 2022, fu realizzato un attacco allo schema di firma digitale Rainbow [16], basato su crittografia multivariata, il quale era considerato tra i più promettenti per diventare uno standard.

Il 3° round si è concluso nel Luglio 2022 quando, dopo 6 anni di competizione, è stato annunciato il gruppo di vincitori che comprende CRYSTALS-Kyber, CRYSTALS-Dilithium, FALCON e Sphincs+. Invece sono stati scelti come candidati del 4° round, attualmente in corso per ulteriori valutazioni, i seguenti algoritmi: BIKE, Classic McEliece, HQC, e SIKE [9]. Nell'agosto del 2022 SIKE è stato attaccato, ed è stato quindi scartato dal 4° round.

I tre algoritmi ancora in considerazione sono progettati per il KEM e non utilizzano reticoli strutturati o funzioni hash nei loro approcci. Invece nel gruppo dei vincitori tre sono basati su reticoli e uno, SPHINCS+, è basato su hash.

Come algoritmo di KEM, il NIST ha selezionato CRYSTALS-Kyber, tra i suoi vantaggi ci sono chiavi di crittografia relativamente piccole che due parti possono scambiarsi facilmente, oltre alla sua velocità di funzionamento.

Per le firme digitali, sono stati selezionati i tre algoritmi CRYSTALS-Dilithium, FALCON e SPHINCS+. I revisori hanno notato l'alta efficienza dei primi due e il NIST raccomanda CRYSTALS-Dilithium come algoritmo principale. FALCON è invece raccomandato per le applicazioni che richiedono firme digitali più piccole di quelle fornite da Dilithium. Il terzo, SPHINCS+, produce firme di dimensioni maggiori, ed è più lento degli altri due, ma è prezioso come backup per un moti-

vo principale: si basa su un approccio matematico diverso rispetto alle altre due selezioni del NIST [38].

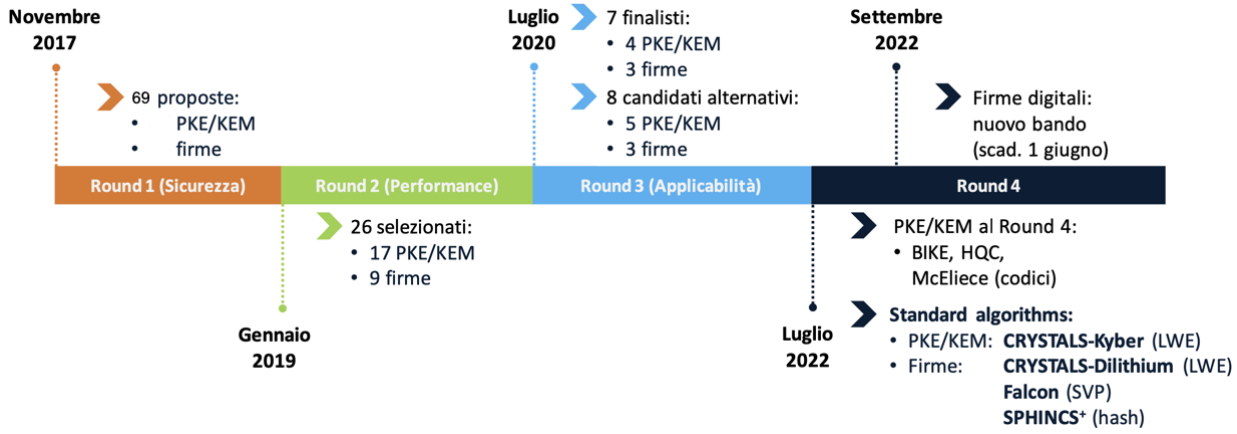


Figura 1.2: Time-line del programma di standardizzazione del NIST

Infine nell'agosto 2023, il NIST ha pubblicato tre bozze di standard per schemi di KEM e di firma digitale, che includono gli algoritmi sopra citati, richiedendo commenti su di essi da parte della comunità:

- FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard [41]
- FIPS 204, Module-Lattice-Based Digital Signature Standard [40]
- FIPS 205, Stateless Hash-Based Digital Signature Standard [42]

Il periodo abilitato ai commenti è stato chiuso nel novembre del 2023.

All'interno di questa tesi ci concentreremo sugli schemi di crittografia per la firma digitale. Nel prossimo capitolo descriveremo il protocollo di firma digitale e il suo funzionamento generale, introducendo uno degli standard attualmente usati il quale però non è resistente alla computazione quantistica: ECDSA. In secondo luogo passeremo a un'analisi degli schemi post-quantistici di firma digitale selezionati dal NIST, in particolare CRYSTALS-Dilithium e SPHINCS+.

Capitolo 2

Protocolli di Firma digitale e lo standard ECDSA

In origine i metodi crittografici sono stati sviluppati per garantire la *confidenzialità* delle comunicazioni, in particolare tra coppie di persone in ambienti ristretti. Con la diffusione delle reti, e in particolare di Internet, il problema si è però enormemente esteso ed ha assunto connotazioni nuove. Sono così emerse tre funzionalità importanti che sono oggi richieste ai protocolli crittografici a seconda dell'uso e del livello di protezione desiderato: **l'integrità**, **l'autenticità** e la **non ripudiabilità**; funzionalità che è possibile garantire con le firme digitali [15].

2.1 La firma digitale

La firma digitale utilizza la **crittografia asimmetrica**, caratterizzata dall'uso di una chiave pubblica e una privata, correlate tra loro. In generale, nel processo di firma, il mittente utilizzerà la propria chiave privata per firmare i dati che vuole inviare al destinatario, a sua volta, il destinatario utilizzerà la chiave pubblica del mittente per verificare la firma sui dati ricevuti.

La firma digitale consente di dimostrare l'**integrità** e l'**autenticità** di un messaggio, oltre ad offrire un'ulteriore funzionalità che è quella della **non ripudiabilità**:

1. **Integrità** dei dati: verifica che il messaggio non sia stato corrotto successivamente al momento in cui è stato firmato. Se un messaggio firmato viene corrotto, la firma digitale ad esso associata non è più valida.

2. **Non ripudiabilità**: poiché il messaggio è firmato utilizzando la chiave privata del mittente, il mittente non può disconoscere un documento da lui firmato e il destinatario può essere sicuro dell'origine del messaggio (solo il mittente con la propria chiave privata avrebbe potuto firmare il messaggio).

3. **Autenticità**: il destinatario può verificare l'identità del mittente; poiché il messaggio è firmato utilizzando la chiave privata del mittente, possiamo essere sicuri dell'autenticità del messaggio.

Inoltre per la sicurezza di una firma digitale, nei confronti di alcune tipologie di attacchi, si richiedono in generale i seguenti requisiti [52]:

- La firma deve essere una funzione che dipende dal messaggio da firmare.
- La firma deve utilizzare alcune informazioni conosciute solo dal firmatario (la sua chiave privata) per prevenire falsificazione e ripudiabilità.
- Deve essere semplice produrre la firma digitale, ovvero si devono avere algoritmi efficienti.
- Deve essere semplice verificare la firma digitale.
- Deve essere computazionalmente intrattabile falsificare una firma digitale, questo viene garantito usando problemi matematici difficili, grazie ai quali è fondamentalmente difficile risalire alla chiave privata del firmatario a partire dalle informazioni pubbliche

2.1.1 Schema generale di firma digitale

Si supponga che Bob voglia mandare un messaggio ad Alice. In questo schema considereremo che il messaggio venga trasmesso in chiaro. Prima della trasmissione, Bob applica al suo messaggio una funzione hash sicura, come per esempio SHA-512, producendo il digest del messaggio. Data l'importanza di questo concetto ai fini della nostra trattazione, riportiamo di seguito la definizione di funzione hash crittograficamente sicura.

Una funzione hash H accetta in input un blocco di dati M di lunghezza variabile e produce un risultato di dimensione fissa $h = H(M)$. Una "buona" funzione hash, ha la caratteristica di produrre output distribuiti uniformemente e apparentemente casuali. In particolare una funzione hash crittograficamente sicura deve soddisfare tutti i requisiti della Tabella 2.1

Tornando al nostro schema di firma, una volta ottenuto il digest, esso, insieme alla chiave privata di Bob, vengono forniti in input a un algoritmo di generazione di firma digitale (ad esempio l'algoritmo di decifrazione di un cifrario a chiave pubblica) ottenendo la firma digitale. Adesso Bob trasmette il messaggio in chiaro con allegata la firma digitale ad Alice.

Quando Alice riceve il messaggio insieme alla firma digitale, applicherà la stessa funzione hash usata da Bob al messaggio in chiaro ricevuto per ottenere un digest. Successivamente utilizza la firma digitale ricevuta e la chiave pubblica di Bob come input per un algoritmo di verifica della firma (ad esempio l'algoritmo di cifratura dello stesso cifrario a chiave pubblica usato per generare la firma), ottenendo così

un digest. Se quest'ultimo digest è identico a quello ottenuto applicando la funzione hash al messaggio in chiaro ricevuto, allora la firma è valida e il messaggio autentico.

Lo schema di firma descritto è riassunto nella Figura 2.1.

Requisito	**Descrizione**
Input di dimensione variabile	H può essere applicata a un blocco di dati di qualsiasi dimensione
Output di dimensione fissa	H produce un output di dimensione fissa
Efficienza	$H(x)$ è relativamente facile da calcolare per ogni dato x, rendendo pratiche le implementazioni sia hardware che software
Resistenza alla preimmagine	Per ogni dato valore hash h, è computazionalmente non realizzabile trovare un y tale che $H(y) = h$
Resistenza alla seconda preimmagine	Per ogni dato blocco x, è computazionalmente non realizzabile trovare un $y \neq x$ con $H(y) = H(x)$
Resistenza alle collisioni	È computazionalmente non realizzabile trovare una qualsiasi coppia (x, y) con $x \neq y$, tale che $H(x) = H(y)$.
Pseudo-casualità	L'output di H deve essere pseudo-casuale

Tabella 2.1: Requisiti per una funzione hash crittograficamente sicura H [52].

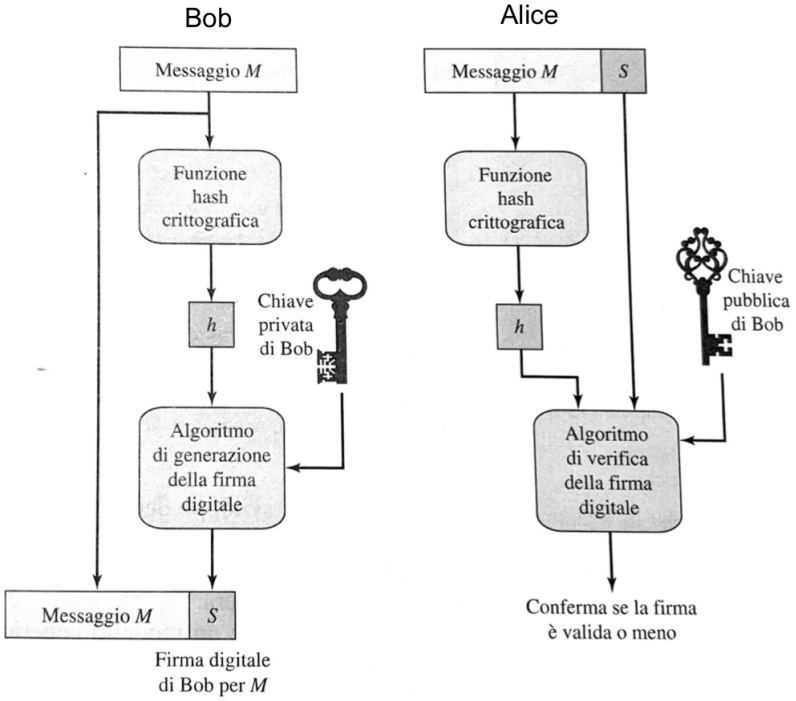


Figura 2.1: Modello generico di costruzione di una firma digitale [52].

2.2 Crittografia su curve ellittiche

Andiamo adesso a definire alcuni concetti chiave della crittografia su curve ellittiche, sulla quale si basa lo schema di firma digitale ECDSA che analizzeremo successivamente.

La crittografia su curve ellittiche (ECC: Elliptic Curve Cryptography) è una tipologia di crittografia a **chiave pubblica** basata sulle curve ellittiche definite su campi finiti, la cui sicurezza si basa sulla difficoltà del **problema del Logaritmo Discreto su curva ellittica** (ECDLP). L'utilizzo di questo metodo crittografico è stato proposto da Neal Koblitz e Victor S. Miller nel 1985 [48]. Essa è nata per risolvere i problemi del carico computazionale che si porta dietro la crittografia a chiave pubblica di "1° generazione" (RSA, El Gamal, DH), nella quale negli ultimi anni, per garantirne un utilizzo sicuro, vi è stato un progressivo aumento della lunghezza della chiave che ha comportato un carico di elaborazione più pesante. La principale attrattiva dell'ECC è che offre la stessa sicurezza dei sistemi a chiave pubblica di 1° generazione con l'utilizzo di chiavi più piccole, riducendo così il carico computazionale. Diamo adesso alcune definizioni per comprenderla al meglio.

Ricordiamo anzitutto che un campo F è un insieme non vuoto dotato di due operazioni binarie interne, chiamate addizione e moltiplicazione, che soddisfano le proprietà associativa, commutativa, di esistenza dell'elemento neutro e di esistenza dell'inverso di ciascun elemento (ad eccezione dello zero per la moltiplicazione).

Definizione 7 (**Curva Ellittica**). *Una curva ellittica E con coefficienti a e b nel campo F è definita come l'insieme dei punti $(x, y) \in F^2$ che soddisfano l'equazione algebrica $y^2 + axy + by = x^3 + cx^2 + dx + e$, dove $a, b, c, d, e \in F$. Per i nostri scopi, ci limiteremo alle equazioni nella forma normale di Weierstrass $y^2 = x^3 + ax + b$:*

$$E(a, b) = \{(x, y) \in F^2 \mid y^2 = x^3 + ax + b\} \cup \{O\}$$

dove O è chiamato Punto all'infinito. (Figura 2.2a)

In particolare in crittografia si fa riferimento alle curve ellittiche su campi finiti, cioè curve le cui variabili e coefficienti sono tutti circoscritti agli elementi di un campo finito. Noi in particolare ci interesseremo delle "curve a valori primi" sul campo finito F_p, con p numero primo. Oltre alle curve prime, la crittografia su curve ellittiche utilizza un'altra famiglia di curve, chiamate "curve ellittiche binarie". Si tratta delle curve i cui coefficienti e le cui variabili assumono valore nel campo $GF(2^m)$, costituito da 2^m elementi che si possono pensare come tutti gli interi binari di m cifre e su cui si può operare mediante l'aritmetica polinomiale modulare.

Definizione 8 (**Curva Ellittica su un campo finito primo**).
Si definisce una curva ellittica su un campo finito primo, una curva E_p, i cui coefficienti a e b appartengono a F_p, ed è definita da tutti quei punti $(x, y) \in$

F_p^2 che soddisfano l'equazione $y^2 \mod p = x^3 + ax + b \mod p$ *insieme al punto all'infinito O:*

$$E_p(a,b) = \{(x,y) \in F_p^2 \mid y^2 \mod p = x^3 + ax + b \mod p\} \cup \{O\}$$

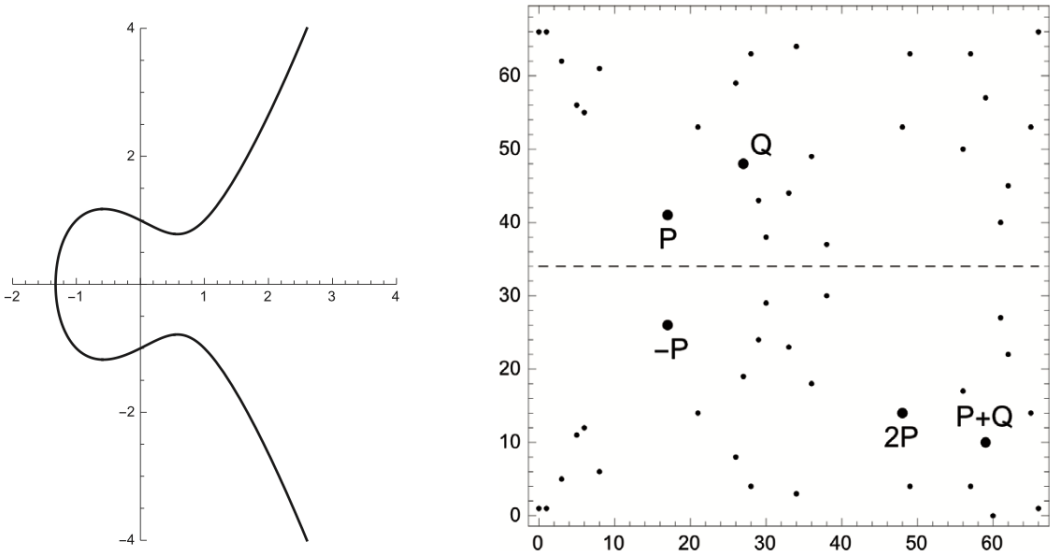
(Figura 2.2b)

Per poter definire protocolli crittografici basati su curve ellittiche, è necessario utilizzare curve ellittiche che definiscano un insieme di punti con la struttura di **gruppo finito abeliano**, in modo che al suo interno possa essere definita una operazione, con particolari proprietà, tra i punti dell'insieme. Si può dimostrare che un gruppo finito può essere definito sull'insieme $E_p(a,b)$ se a e b soddisfano la seguente condizione:

$$4a^3 + 27b^2 \mod p \neq 0 \mod p$$

la quale garantisce che l'equazione della curva abbia radici distinte.

L'operazione che viene definita in questo gruppo abeliano è l'addizione, la cui definizione non verrà trattata in questo contesto, ma diremo solo che una delle sue proprietà è che O funge da elemento neutro.



(a) Curva $y^2 = x^3 - x + 1$ (b) Curva ellittica prima $E_{67}(-1,1)$

Figura 2.2

Come già anticipato, la sicurezza dell'ECC si basa sulla difficoltà del problema del Logaritmo Discreto su curva ellittica. Prima di darne la definizione definiamo il concetto di ordine di un punto della curva ellittica: l'ordine n di un punto $P \in E_q(a,b)$ è il più piccolo numero intero tale che $nP = O$.

Adesso possiamo dare la seguente definizione

Definizione 9 (**Problema del Logaritmo Discreto su curva ellittica**). *Data una curva ellittica $E_q(a,b)$ definita sul campo finito Fq, un punto $P \in E_q(a,b)$ di ordine n e un punto $Q = kP$, dove $k \in [0, \ldots n-1]$, trovare k. L'intero k è chiamato logaritmo discreto di Q in base P e si indica con $k = \log_P Q$.*

L'adottabilità della crittografia a curva ellittica dipende dalla facilità di calcolare la moltiplicazione di un punto P per uno scalare k, ovvero calcolare $Q = kP$, utilizzando l'algoritmo polinomiale dei raddoppia successivi. La sicurezza di questo schema crittografico è invece legata all'incapacità di calcolare k, dati i punti P e Q, problema per il quale sono attualmente noti solo algoritmi di costo esponenziale. La dimensione della curva ellittica, misurata dal numero totale di coppie intere discrete (punti) che soddisfano l'equazione della curva, determina la difficoltà del problema [48].

I parametri delle curve ellittiche per crittosistemi devono essere scelti accuratamente in modo da resistere agli attacchi noti al problema ECDLP. L'attacco più ovvio consiste nella ricerca esaustiva, con la quale viene calcolata la sequenza $(P,\ 2P,\ 3P,\ ...,\ (n-1)\,P)$ fino a che non si ottiene Q. Il costo computazionale in tempo è di n passi nel caso peggiore e $n/2$ passi nel caso medio. Si capisce bene che per contrastare la ricerca esaustiva è sufficiente considerare valori di n molto grandi [23].

Osserviamo infine che il Problema del Logaritmo discreto è considerato estremamente difficile e, a parità di dimensione del campo, molto più difficile dei tradizionali problemi della fattorizzazione degli interi e del logaritmo discreto nell'algebra modulare. Infatti per questi problemi esiste un algoritmo subesponenziale chiamato *index calculus* per calcolare il logaritmo discreto, che può essere utilizzato per attaccare sia il protocollo DH che il cifrario RSA. L'algoritmo *index calculus*, per la cui descrizione rimandiamo alla letteratura specialistica [4], sfrutta una struttura algebrica dei campi finiti che non è presente sulle curve ellittiche. Ad oggi, nessuno è stato capace di progettare algoritmi di tipo *index calculus* per risolvere il problema del logaritmo discreto per le curve ellittiche: quindi i protocolli per tali curve sembrano invulnerabili a questo tipo di attacchi. Al momento il migliore attacco noto al problema del logaritmo discreto per le curve ellittiche, detto *Pollard* ρ, richiede in media $O(2^{b/2})$ operazioni per chiavi di b bit ed è dunque pienamente esponenziale. Tutto ciò ha delle implicazioni evidenti sulla sicurezza: a parità di lunghezza delle chiavi, i protocolli basati sulle curve ellittiche (ECC) sono molto più sicuri del cifrario RSA e del protocollo DH, considerati equivalenti tra di loro dal punto di vista della sicurezza in quanto entrambi soggetti agli attacchi basati sull'index calculus.

Questo significa che per garantire lo stesso livello di sicurezza la crittografia su curve ellittiche richiede chiavi pubbliche di lunghezza minore e quindi più facilmente utilizzabili. In proposito il NIST ha pubblicato nel seguente documento [14] la Tabella 2.2 di equivalenza fra i livelli di sicurezza degli algoritmi simmetrici rispetto agli algoritmi a chiave pubblica, dove di particolare interesse è il confronto tra RSA ed ECC. La tabella riporta la dimensione in bit delle chiavi che garantiscono livelli di sicurezza equivalenti nei tre diversi sistemi, dove due sistemi si considerano di

sicurezza equivalente se è richiesto lo stesso costo computazionale per forzarli [15]. Ad esempio, il lavoro necessario per forzare un cifrario simmetrico come AES con chiave di 128 bit è pari a quello necessario per forzare un cifrario RSA a 3072 bit, mentre per garantire un'analoga sicurezza con un cifrario basato sull'ECC sono sufficienti 256 bit.

Algoritmi a chiave simmetrica (TDEA, AES) (bit della chiave)	RSA e DH (bit del modulo)	ECC (bit dell'ordine)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Tabella 2.2: Confronto del livello di sicurezza tra algoritmi simmetrici algoritmi asimmetrici [14]

2.3 ECDSA: Elliptic Curve Digital Signature Algorithm

In crittografia, l'Elliptic Curve Digital Signature Algorithm (ECDSA) è uno schema di firma digitale crittograficamente sicuro nei confronti della computazione classica ma non di quella quantistica, che offre una variante del Digital Signature Algorithm (DSA), usando la crittografia ellittica. Il DSA è lo standard del NIST per la firma digitale che basa la sua difficoltà nel calcolare il logaritmo discreto su campi finiti. L'ECDSA si basa invece sulla matematica dei gruppi ciclici delle curve ellittiche su campi finiti e sulla difficoltà del **problema del logaritmo discreto su curva ellittica**. Ricordiamo che un gruppo ciclico è un gruppo abeliano.

L'ECDSA fu proposto la prima volta nel 1992 da Scott Vanstone, nel 1998 è diventato uno standard ISO (ISO 14888), nel 1999 è stato accettato come standard ANSI (ANSI X9.62) mentre nel 2000 è diventato uno standard IEEE (IEEE P1363 2). Infine è stato inserito nella versione del 2009 del documento FISP 186 (Federal Information Processing Standard) del NIST.
Attualmente l'ECDSA è ampiamente utilizzato in molte applicazioni di sicurezza, ad esempio, è lo schema di firma digitale scelto per Bitcoin nel quale si fa uso della curva "secp256k1" [49].

Come tutti gli schemi di firma digitale esso è costituito da 3 fasi: **generazione delle chiavi**, **generazione della firma** e **verifica della firma**.

2.3.1 Parametri globali del dominio

Prima di iniziare la fase di generazione delle chiavi devono essere scelti i **parametri globali** dello schema di firma:

- Si sceglie una curva ellittica $E_p(a, b)$ definita su un un campo finito Fp con p primo, di equazione $y^2 = x^3 + ax + b$, caratterizzata dai parametri a e b.
- Si sceglie un punto base $G = (x_g, y_g) \in E_p(a, b)$, cioè un generatore della curva ellittica, di ordine n primo e grande, cioè n è il più piccolo numero intero tale che $nG = O$. Per la sicurezza dello schema di firma, ANSI X9.62 ha posto $n > 2^{160}$.

Nel caso della curva "secp256k1" usata da Bitcoin i parametri sono i seguenti:

- coefficiente $a = 0$.
- coefficiente $b = 7$.
- numero primo $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
- punto base $G = (G_x, G_y)$= (0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798, 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8).
- ordine n del punto base G n=0xfffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141

2.3.2 Generazione delle chiavi

Il firmatario, chiamiamolo Alice, genera una coppia di chiavi, una privata SK_A e una pubblica PK_A, nel seguente modo:

1. Seleziona un intero d in maniera casuale dall'intervallo $d \in [1, n - 1]$
2. Calcola $Q = dG$, che sarà un punto in $E_p(a, b)$, utilizzando l'algoritmo dei raddoppi successivi.
3. La chiave pubblica è Q, quella privata è d, quindi $(PK_A, SK_A) = (Q, d)$.

2.3.3 Generazione della firma

Il firmatario Alice, avendo i parametri pubblici dello schema di firma e la propria chiave privata, può adesso generare una firma per il suo messaggio m; andrà ad eseguire i seguenti passaggi:

1. Seleziona un intero casuale $k \in [1, n - 1]$

2. Calcola il punto $P = (x, y) = kG$ e pone $r = x \mod n$. Se $r = 0$, riparte dal punto 1

3. Calcola $k^{-1} \mod n$

4. Calcola $e = H(m)$, dove H è una funzione hash crittografica (SHA-2 o SHA-3)

5. Calcola $s = k^{-1}(e + dr) \mod n$, dove d è la sua chiave privata. Se $s = 0$, riparte dal punto 1.

6. La firma di Alice per il messaggio m è la coppia di interi (r, s)

2.3.4 Verifica della firma

Il destinatario, chiamiamolo Bob, conosce la chiave pubblica Q di Alice e i parametri globali del sistema: la curva ellittica $E_p(a, b)$, il punto base G e l'ordine n di esso. Bob riceve il messaggio in chiaro m e la firma (r, s) di Alice, e andrà a verificare la firma eseguendo i seguenti passaggi:

1. Verifica che r ed s siano interi compresi in $[1, n-1]$, in caso contrario la firma non è valida.

2. Calcola $e = H(m)$, dove H è la stessa funzione hash usata nel processo di generazione della firma.

3. Calcola $w = s^{-1} \mod n$

4. Calcola $u_1 = ew \mod n$ e $u_2 = rw \mod n$

5. Calcola il punto $X = (x_1, y_1) = u_1G + u_2Q$

6. Se $X = O$, rifiuta la firma, altrimenti calcola $v = x_1 \mod n$

7. Accetta la firma di Alice se e solo se $v = r$

Dimostrazione di correttezza della verifica della firma

Se (r, s) è la firma legittima del messaggio m ricevuto da Alice, dalla definizione del parametro s risulta:

$$s = k^{-1}(e + dr) \mod n$$

Quindi

$$\begin{aligned} k &= s^{-1}(e + dr) \mod n \\ &= (s^{-1}e + s^{-1}dr) \mod n \\ &= (we + wdr) \mod n \qquad (\text{perché } w = s^{-1} \mod n) \\ &= (u_1 + u_2 d) \mod n \qquad (\text{perché } u_1 = ew \mod n \quad \text{e} \quad u_2 = rw \mod n). \end{aligned}$$

Adesso si consideri che

$$X = u_1G + u_2Q = u_1G + u_2dG = (u_1 + u_2d)G = kG \tag{2.1}$$

Al passo 6 della verifica avevamo che

$$v = x_1 \mod n$$

dove x_1 è l'ascissa del punto $X = (x_1, y_1) = u_1G + u_2Q$. Al passo 2 della generazione della firma avevamo invece posto che

$$r = x \mod n$$

dove x è l'ascissa del punto $P = (x, y) = kG$. La correttezza segue dal fatto che dal passaggio 2.1 possiamo dire che

$$X = kG$$

ma allo stesso tempo sappiamo che

$$P = (x, y) = kG$$

quindi

$$P = X$$

$$(x, y) = (x_1, y_1)$$

$$x = x_1$$

da cui

$$\mathbf{r{=}v}$$

(perché $v = x_1 \mod n$ e $r = x \mod n$)[52][49].

2.3.5 Sicurezza

Si può notare che è necessario sia k (il valore casuale) che d (la chiave privata) per calcolare il valore s della firma, ma sono necessari solo r e Q (chiave pubblica) per verificare la firma. Poiché $r = x \mod n$ dove x è l'ascissa del punto $P = (x, y) = kG$ e $Q = d \cdot G$ e a causa della difficoltà del problema del logaritmo discreto su curve ellittiche, per il quale non esiste un algoritmo risolutivo che impieghi tempo polinomiale, è computazionalmente improponibile calcolare d conoscendo Q o k conoscendo r. Questo rende sicuro l'algoritmo ECDSA: non c'è modo di trovare la chiave privata d e l'intero casuale k ed è impraticabile per qualcuno falsificare una firma senza accesso alla chiave privata d del firmatario.

Accorgimenti cruciali per la sicurezza

Come stabiliscono gli standard, è cruciale che l'intero k debba essere generato casualmente ed essere unico per ciascuna firma, in caso contrario l'equazione al passo 5 di generazione della firma può essere risolta ricavando la chiave privata d. Infatti, presi due messaggi noti diversi m e m', firmati con lo *stesso* k segreto, dato che il valore della firma r dipende solo da k, esso sarà uguale per le due firme sui due messaggi m e m':

- firma su $m : (r, s)$
- firma su $m' : (r, s')$

Siano $e = H(m)$ ed $e' = H(m')$. Risulta

$$s = k^{-1}(e + dr)$$

$$s' = k^{-1}(e' + dr)$$

da cui si ottiene:

$$\begin{aligned}(s - s') &= k^{-1}(e + dr) - k^{-1}(e' + dr) \\ &= k^{-1}(e + dr - (e' + dr)) \\ &= k^{-1}(e + dr - e' - dr) \\ &= k^{-1}(e - e')\end{aligned}$$

Da quest'ultima equazione il crittoanalista può trovare

$$k = \frac{e - e'}{s - s'}$$

e ricordando che $s = k^{-1}(e + dr)$, il crittoanalista può trovare la chiave privata d:

$$d = \frac{sk - e}{r}\,.$$

Tutte le precedenti operazioni sono svolte modulo n.

Questa implementazione errata è stata sfruttata, per esempio, per estrarre la firma digitale usata nella console PlayStation3 [49].

In modo analogo al problema del logaritmo discreto intero, l'ECDLP risulta difficile sui computer classici ma ha una soluzione efficiente sui computer quantistici ancora una volta tramite l'algoritmo di Shor. Rimandiamo alla letteratura recente [29] per una discussione relativa alla generalizzazione dell'algoritmo di Shor al caso ECDLP.

Capitolo 3

Crittografia sui lattici e firma Dilithium

Come anticipato nella sezione 1.3.3, uno dei primi quattro vincitori del processo di standardizzazione della crittografia post quantistica del NIST è CRYSTALS-Dilithium. Si tratta di un algoritmo di firma digitale che basa la sua sicurezza sulla difficoltà di alcuni problemi definiti su reticoli e ritenuti intrattabili anche per un computer quantistico, quindi prima di introdurre l'algoritmo di Dilithium, richiameremo alcuni concetti teorici fondamentali della crittografia basata sui reticoli.

3.1 Crittografia basata sui reticoli

In semplici termini matematici, un reticolo può essere pensato come una raccolta di punti regolarmente distanziati che si ripetono a intervalli fissi. Per descrivere come questi punti sono posizionati l'uno rispetto all'altro, usiamo i vettori. La disposizione specifica di questi vettori, che definisce il layout del reticolo, è indicata come base ed ogni punto all'interno del reticolo può essere espresso come combinazione lineare, a coefficienti interi, dei vettori della base. È importante notare che i reticoli non sono limitati a due o tre dimensioni ma possono estendersi a dimensioni più elevate e, in campi come la crittografia, possono coinvolgere 1000 o più dimensioni [26].

Definizione 10 (**Reticolo**)**.**
Un reticolo, in inglese lattice, può essere definito nel seguente modo:

$$\mathcal{L} = \left\{ \sum_{i=1}^{n} x_i b_i | x_i \in \mathbb{Z}, b_i \in \mathbb{R}^m \right\}$$

dove b_i sono vettori linearmente indipendenti di dimensione m che vanno a formare la base del reticolo; m è la dimensione del reticolo ed n il rango del reticolo.

Si dice che un reticolo è di rango completo quando $m = n$. La base di un reticolo non è unica, anzi l'esistenza di molteplici basi per lo stesso reticolo è importante

per lo sviluppo di algoritmi crittografici [52]. Essendoci infinite basi che possono descrivere un reticolo esiste una nozione di base favorevole e base sfavorevole:

- **base favorevole**: composta da vettori corti e il più possibile ortogonali, i problemi sui lattici sono più facili da risolvere avendo a disposizione una base favorevole.
- **base sfavorevole**: composta da vettori lunghi e vicini tra loro, i problemi sui lattici sono più difficili da risolvere su una base sfavorevole.

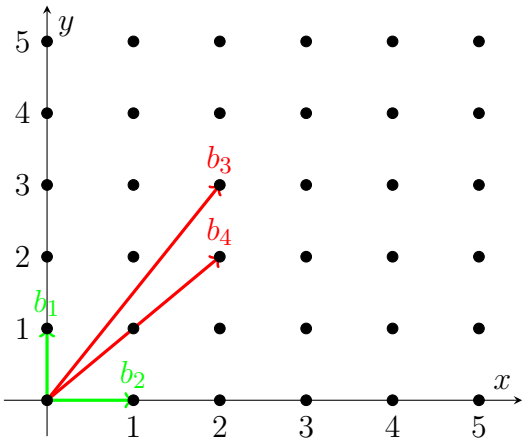


Figura 3.1: Base favorevole (b_1, b_2) e base sfavorevole (b_3, b_4)

Come detto in precedenza, sui reticoli è possibile definire alcuni problemi computazionali sulla cui difficoltà di risoluzione si basa la sicurezza degli schemi di crittografia basata sui lattici.

3.1.1 Shortest Vector Problem (SVP)

Questo problema consiste nel trovare il vettore non nullo più corto all'interno del reticolo [52]. Sia $\lambda(\mathcal{L})$ la lunghezza del vettore non nullo più corto nel reticolo $\mathcal{L}$, cioè

$$\lambda(\mathcal{L}) = \min_{v \in \mathcal{L} \setminus \{0\}} \|v\|_N \,,$$

dove la norma N di solito è la norma Euclidea.

Definizione 11 (Shortest Vector Problem (SVP)).
Data una base arbitraria $B = \{b_1, ..., b_k\}$ di un reticolo $\mathcal{L} = \mathcal{L}(B)$, trovare il vettore del reticolo più corto non nullo, cioè, un $v \in \mathcal{L}$ per cui

$$||v|| = \lambda(\mathcal{L}) = \min_{v' \in \mathcal{L} \setminus \{0\}} \|v'\|_N$$

Particolarmente importanti per la crittografia dei reticoli sono i problemi di approssimazione, che sono parametrizzati da un fattore di approssimazione $\gamma \geq 1$ che

è tipicamente considerato come una funzione della dimensione del reticolo n, cioè $\gamma = \gamma(n)$. La versione approssimata di SVP è la seguente (si noti che impostando $\gamma(n) = 1$ abbiamo il problema definito sopra)

Definizione 12 (**Approximate Shortest Vector Problem** (SVP_γ))**.** *Data una base arbitraria $B = \{b_1, ..., b_k\}$ di un reticolo $\mathcal{L} = \mathcal{L}(B)$ di dimensione n, trovare un vettore del reticolo non nullo $v \in \mathcal{L}(B)$ per il quale vale*

$$||v|| \leq \gamma(n) \cdot \lambda(\mathcal{L})$$

In poche parole, questo problema si traduce nel trovare un vettore che non sia "troppo più lungo" del vettore più corto [43].

Difficoltà nel Caso medio e nel Caso peggiore

Sebbene ci siano molti problemi noti NP-hard, non tutti questi problemi sono adatti come base per la sicurezza crittografica. In questo contesto, la nozione di difficoltà nel caso medio è utile per la crittografia. Un problema è **difficile nel caso medio** se la maggior parte delle istanze del problema estratte casualmente da una certa distribuzione sono difficili, mentre un problema è **difficile nel caso peggiore** se è difficile solo in alcune istanze isolate: quelle del caso peggiore. I crittografi che si occupano di crittografia post quantistica cercano quindi problemi matematici che soddisfano l'ipotesi di difficoltà nel caso medio e impiegano strumenti teorici come le riduzioni dal caso peggiore al caso medio per identificare protocolli adatti la cui sicurezza ed efficienza possono essere garantite [26].

3.1.2 Algoritmi e complessità

I problemi sui reticoli SVP e SVP_γ sono stati intensamente studiati e nonostante esistano diversi algoritmi di risoluzione, essi diventano computazionalmente difficili da risolvere in reticoli di grandi dimensioni. Tra gli algoritmi di risoluzione di questi due problemi possiamo citare LLL [31] (Lenstra, Lenstra, Lovasz), proposto nel 1982. Questo, però, è in grado di risolvere in tempo polinomiale solamente la versione approssimata di SVP per valori di γ di ordine $\gamma = 2^{\Theta(n \log \log n / \log n)}$, troppo elevati per poter essere utilizzati in un attacco reale ai danni degli schemi basati sui reticoli.

Dal lato della complessità, molti problemi sui reticoli sono noti per essere NP-hard. Infatti nel caso specifico di SVP è stato il matematico ungherese Miklós Ajtai a dimostrare che SVP è NP-hard nel peggiore dei casi, per alcuni reticoli casuali [6], implicando che è un problema in grado di dare protezione anche nei confronti di attacchi quantistici. Il lavoro di Ajtai [5] è stato molto importante perché ha fornito inoltre la prima riduzioni dal caso peggiore al caso medio per un problema sui reticoli, fornendo una riduzione da SVP_γ al problema SIS (Short Integer Solutions), problema che analizzeremo nella Sezione 3.1.3.

Ajtai [5] introducendo il problema SIS, ha dimostrato che risolvere SIS al caso medio è almeno difficile quanto risolvere vari problemi di reticolo nel caso peggiore.

Con la riduzione dal caso peggiore al caso medio è possibile definire alcune costruzioni che risultano difficili a meno che *tutte* le istanze di alcuni problemi sui reticoli, ad esempio SVP e SVP_γ, siano facili da risolvere. Questo risultato, rafforza la fiducia nel design di schemi crittografici basati sui reticoli. Infatti, la sicurezza di questi, per opportuni parametri, è garantita a meno che non si possa dimostrare che è *sempre* possibile risolvere facilmente un'istanza di SVP_γ, cosa che non pare realistica considerati i costi computazionali degli algoritmi attualmente utilizzati per trovare soluzioni a questi problemi. Inoltre, nonostante i numerosi tentativi compiuti a partire dagli anni '90, non sono stati proposti algoritmi quantistici che risolvono SVP_γ con prestazioni nettamente migliori degli algoritmi classici sopra citati. Questo fatto supporta la convinzione diffusa nella comunità matematica che i problemi sui reticoli possano costituire la base adatta su cui sviluppare schemi crittografici resistenti ad attacchi di un computer quantistico e a cui affidare il futuro della crittografia a chiave pubblica [22].

Successivamente al lavoro di Ajtai, Regev[46] nel 2005 ha introdotto un altro importante problema sui reticoli: Learning With Errors. Regev ha dimostrato la difficoltà del problema LWE descrivendo una riduzione da SVP_γ a LWE. I protocolli crittografici che si basano quindi su SIS o LWE godono quindi della proprietà di essere dimostrabilmente sicuri come un problema del caso peggiore (SVP_γ) che si sospetta fortemente essere estremamente difficile. Tuttavia le applicazioni crittografiche di SIS e LWE sono inefficienti a causa della dimensione della chiave pubblica associata, che tipicamente è una matrice A. Per aggirare questa inefficienza, si usano varianti del LWE e di SIS, come Ring-SIS, Ring-LWE, Module-SIS, Module-LWE le quali hanno chiavi più compatte e consentono implementazioni di protocolli crittografici computazionalmente più efficienti, a parità di sicurezza. Allo stesso modo di SIS e LWE, questi problemi ammettono riduzioni da problemi sui reticolo nel caso peggiore [33][44][34].

Introduciamo nel seguito i problemi SIS, LWE e le loro varianti accennate nelle righe precedenti.

3.1.3 Short Integer Solutions (SIS)

Il problema Short Integer Solution è un problema difficile nel caso medio che viene utilizzato nelle costruzioni di crittografia basate sui reticoli. Nel 1996 Ajtai [5] ha dimostrato che il problema SIS è sicuro al caso medio se SVP_γ (dove $\gamma = n^c$ per qualche costante $c > 0$) è difficile nel peggiore dei casi. Il problema SIS e le sue varianti sono utilizzati in diversi schemi di crittografia post-quantistica tra cui CRYSTALS-Dilithium e Falcon [51].

Definizione 13 (**Short Integer Solutions (SIS)**).
Sia $A \in \mathbb{Z}_q^{n \times m}$ una matrice di dimensione $n \times m$ i cui elementi sono interi modulo q, cioè appartengono a $\mathbb{Z}_q$. Trovare un vettore non nullo $x \in \mathbb{Z}^m$, di norma $||x|| \leq \beta$ tale che

$$Ax = 0 \in \mathbb{Z}_q^n$$

Al fine di garantire che $Ax = 0$ abbia una soluzione breve e non banale, richiediamo: $\beta < q$, $\beta \geq \sqrt{n \log q}$ e $m \geq n \log q$ [51].

3.1.4 Ring Short Integer Solutions (R-SIS)

Una variante di SIS è il Ring-SIS introdotto in [44, 33].

Definizione 14 (Ring-SIS).
Sia $R_q = \mathbb{Z}_q[X]/(X^n+1)$ un anello polinomiale di grado n intero, i cui elementi sono polinomi di grado inferiore a n con coefficienti appartenenti a $\mathbb{Z}_q$ e con relazione $X^n = -1$. Dotiamo inoltre R_q con una certa norma $||\cdot||$. Presi m elementi indipendenti uniformemente casuali $a_i \in R_q$ (a_i è un polinomio in R_q) si definisce un vettore $\vec{a} := (a_1, \ldots, a_m) \in R_q^m$. Trovare un vettore non nullo $\vec{z} := (z_1, \ldots, z_m) \in R^m$ tale che:

$$\vec{a}^T \cdot \vec{z} = \sum_{i=1}^{m} a_i \cdot z_i = 0 \mod q \quad e \quad 0 < \|\vec{z}\| \leq \beta$$

Il vantaggio principale di Ring-SIS rispetto a SIS è la sua relativa compattezza ed efficienza: il numero m di elementi $a_1 \in R_q$ richiesti per garantire l'esistenza di una soluzione sufficientemente breve è solo $m \approx \log q$, piuttosto che $m \approx n \log q$ nel caso di SIS. In breve, in Ring-SIS ogni elemento casuale $a_i \in R_q$ corrisponde a n vettori correlati (non indipendenti) $a_i \in \mathbb{Z}_q^n$ in SIS, dove. Allo stesso modo, ogni elemento dell'anello $z_i \in R$ di una soluzione Ring-SIS rappresenta un blocco corrispondente di n interi in una soluzione SIS. Questa regola empirica può essere formalizzata trattando Ring-SIS come un caso speciale di SIS. La struttura più ricca di R_q porta a un aumento dell'efficienza, ma anche a più forti ipotesi di difficoltà del caso peggiore. Per maggiori dettagli consultare [43].

Matrice nega-circolante

La matrice *nega-circolante* del polinomio b è definita così:

$$\text{Per} \quad b = \sum_{i=0}^{n-1} b_i x^i \in R_q, \quad \text{Rot}(b) := \begin{bmatrix} b_0 & -b_{n-1} & \ldots & -b_1 \\ b_1 & b_0 & \ldots & -b_2 \\ \vdots & \vdots & \ddots & \\ b_{n-1} & b_{n-2} & & b_0 \end{bmatrix}$$

Quando $n = 2^k$, nell'anello polinomiale $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ la moltiplicazione $\vec{a}^T \cdot \vec{z}$ può essere calcolata in maniera più semplice calcolando la matrice nega-circolante di tutti i polinomi a_i nel vettore $\vec{a}$, moltiplicando ogni $\text{Rot}(a_i)$ per il vettore che contiene i coefficienti di tutti gli z_i, e infine sommando tutti i risultati. In questa configurazione, R-SIS è una variante di SIS in cui A è costituita da blocchi di matrici nega-circolanti: $A = [Rot(a_1)|...|Rot(a_m)]$ [51][28].

3.1.5 Module Short Integer Solutions (M-SIS)

Il problema Module-SIS generalizza sia SIS che Ring-SIS.

Definizione 15 (**Module-SIS**). *Dati m vettori di polinomi $a_1, \ldots, a_m \in R_q^d$ indipendenti, scelti da una distribuzione uniforme, e la corrispondente matrice $A := (a_1, \ldots, a_m) \in R_q^{d \times m}$, trovare $\vec{z} := (z_1, \ldots, z_m) \in R^m$ tale che:*

$$A^T \cdot \vec{z} = \sum_{i=1}^{m} a_i \cdot z_i = 0 \mod q \quad e \quad 0 < \|\vec{z}\| \leq \beta$$

Sebbene M-SIS sia una variante meno compatta di SIS rispetto a R-SIS, il problema M-SIS è almeno asintoticamente difficile quanto R-SIS e quindi fornisce un limite più stretto sull'ipotesi di difficoltà di SIS. Questo rende l'assunzione della difficoltà di M-SIS una supposizione di base più sicura, ma meno efficiente, rispetto a R-SIS [51]. Come R-SIS, M-SIS può essere interpretato in termini di matrici. Facendo riferimento alla precedente definizione di matrice nega-circolante, M-SIS consiste nel prendere una matrice SIS, chiamiamola A, nella forma:

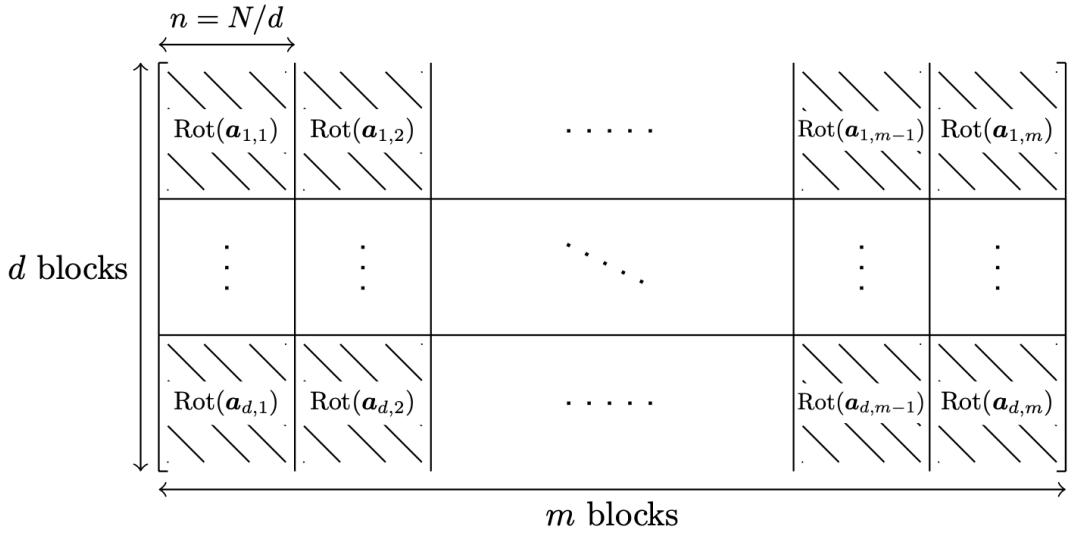


Figura 3.2: Matrice A in Module-SIS [28]

dove ad ogni polinomio $a_{j,i}$ in ogni vettore a_i che compone la matrice A è stata applicata la rotazione $Rot(a_{j,i})$ generando una matrice nega-circolante, dove $j \in \{1, \ldots, d\}$ e $i \in \{1, \ldots, m\}$. Nella matrice A avremo quindi d righe di matrici nega-circolanti e m colonne di matrici nega-circolanti. A questo punto per effettuare la moltiplicazione $\vec{a}^T \cdot \vec{z}$, si moltiplica la matrice A per il vettore z che contiene i coefficienti di tutti i polinomi z_i. Vedremo successivamente un esempio di come questa rotazione semplifichi effettivamente la moltiplicazione $\vec{a}^T \cdot \vec{z}$ applicandola nel problema LWE.

3.1.6 Learning With Errors (LWE)

Il problema Learning With Errors è stato introdotto dall'informatico e matematico teorico Oded Regev [46] nel 2005 ed ha avuto fin da subito un impatto sulla crittografia, soprattutto quella post quantistica.

Il LWE si basa sull'idea di rappresentare un segreto all'interno di un sistema di equazioni lineari random e nasconderlo introducendo un errore (o rumore) all'interno del sistema, in quanto l'introduzione del rumore rende difficile risolvere il sistema e quindi trovare il segreto.

Come vedremo dalla sua definizione, il problema LWE è molto simile al problema SIS. I due problemi sono infatti sintatticamente molto simili, e possono essere visti come duali l'uno dell'altro.

LWE è parametrizzato da interi positivi n e q e da una distribuzione di errore χ su $\mathbb{Z}$, dove χ è solitamente una distribuzione Gaussiana discreta dalla quale si campiona l'errore.

Definizione 16 (**Learning With Error (LWE)**).
Per un vettore $s \in \mathbb{Z}_q^n$ chiamato segreto, consideriamo m vettori $a_1, \ldots, a_m \in \mathbb{Z}_q^n$ campionati secondo la distribuzione uniforme e m valori $e_1, \ldots, e_m \in \mathbb{Z}_q$ campionati dalla distribuzione χ, ottenendo il seguente sistema, dove $b_i \in \mathbb{Z}_q$:

$$\begin{cases} \langle a_1, s\rangle + e_1 = b_1 \pmod q \\ \langle a_2, s\rangle + e_2 = b_2 \pmod q \\ \vdots \\ \langle a_m, s\rangle + e_m = b_m \pmod q \end{cases} \tag{3.1}$$

Il sistema può essere riscritto nella seguente forma matriciale dove ogni vettore a_i rappresenta una riga della matrice

$$\begin{pmatrix} a_{1,1} & \ldots & a_{1,n} \\ a_{2,1} & \ldots & a_{2,n} \\ \vdots & \vdots & \vdots \\ a_{m,1} & \ldots & a_{m,n} \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \pmod q$$

che in forma più compatta diventa $As + e = b \pmod q$. In questo scenario LWE restituirà m coppie (a_i, b_i) con $i \in \{1, \ldots, m\}$ che possono essere viste in maniera equivalente alla coppia (A, b).

Vi sono due versioni del problema LWE:

- **Ricerca**: Nella versione di ricerca del LWE, dati m campioni

$$(a_i, b_i) = (a_i, \langle a_i, s\rangle + e_i) \in Z_q^n \times Z_q \,,$$

dove a_i è campionato secondo la distribuzione uniforme ed e_i, da cui dipende b, è campionato secondo la distribuzione χ, si richiede di trovare s.

- **Decisione**: La versione decisionale del LWE, consiste nell'essere in grado di distinguere se i campioni (a_i, b_i) sono tratti dalla distribuzione del problema o da campioni casuali.

È stato dimostrato che la versione di ricerca del LWE è equivalente a quella decisionale. In entrambe osserviamo che trovare il segreto s senza l'errore e_i è molto facile, lo si può fare applicando l'algoritmo di eliminazione Gaussiana. Introducendo l'errore e_i invece questi sistemi diventano difficili da risolvere, ed è proprio su tale complessità che si basano i sistemi crittografici a chiave pubblica costruiti a partire dal LWE, dove la chiave pubblica è rappresentata da (A, b) e quella privata da s.

Questi schemi basati su LWE per essere sicuri richiedono però chiavi pubbliche di grandi dimensioni, dell'ordine di $n \times m$, cioè la dimensione della matrice A. Quindi al crescere dei parametri di sicurezza n e m, la dimensione di A, membro della chiave pubblica, ha una crescita quadratica. È desiderabile però che la matrice abbia una dimensione che sia al più lineare rispetto ai parametri di sicurezza. Per raggiungere questo obbiettivo si cerca di dare più struttura alla matrice A, in modo che possa essere rappresentata da un'informazione limitata e quindi sia necessario meno spazio per memorizzarla. A questo scopo sono state introdotte due varianti del LWE: Ring-LWE e Module-LWE.

3.1.7 Ring Learning With Errors (R-LWE)

Ring-LWE è una variante del problema LWE introdotto in [34].

Definizione 17 (**Ring-LWE**). *Sia $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ un anello polinomiale di grado n potenza di 2, i cui elementi sono polinomi di grado inferiore a n con coefficienti appartenenti a $\mathbb{Z}_q$ e con relazione $x^n = -1$. Prendiamo un polinomio $s \in R_q$, chiamato segreto, e consideriamo un polinomio $a \in R_q$ campionato secondo la distribuzione uniforme ed un polinomio $e \in R_q$ campionato secondo la distribuzione χ. In questo modo otteniamo la coppia $(a, b = s \cdot a + e \mod q)$*

Adesso per costruire la matrice A, possiamo prendere il polinomio a e costruire la sua matrice nega-circolante.

Vediamo un esempio. Supponiamo che il polinomio a sia

$$a(x) = 4 + 3x + 2x^2 + 12x^3 \in R_{13} = \mathbb{Z}_{13}[x]/(x^4 + 1)\,.$$

$Rot(a)$ sarà la matrice $A \in \mathbb{Z}_{13}^{4\times 4}$:

$$A = \begin{bmatrix} 4 & 1 & 11 & 10 \\ 3 & 4 & 1 & 11 \\ 2 & 3 & 4 & 1 \\ 12 & 2 & 3 & 4 \end{bmatrix}$$

Si noti che adesso per memorizzare A, è necessario avere solo gli n coefficienti del polinomio, da cui è possibile costruire tutta la matrice $n \times n$. Quindi la matrice A può essere memorizzata in spazio $O(n)$ anziché $O(n^2)$.

Il problema Ring-LWE diventa quindi $As + e = b$ dove A è la matrice nega-circolante del polinomio a, s è il vettore che contiene i coefficienti del polinomio segreto ed e è il vettore che contiene i coefficienti del polinomio che rappresenta l'errore e.

Per esempio, se scegliamo

$$s(x) = 6 + 9x + 11x^2 + 11x^3$$
$$e(x) = 0 - 1x + 1x^2 + 1x^3\,,$$

otteniamo:

$$\begin{bmatrix} 4 & 1 & 11 & 10 \\ 3 & 4 & 1 & 11 \\ 2 & 3 & 4 & 1 \\ 12 & 2 & 3 & 4 \end{bmatrix} \begin{pmatrix} 6 \\ 9 \\ 11 \\ 11 \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \\ 4 \\ 12 \end{pmatrix} \pmod{13}$$

La coppia (a, b) si riduce quindi ad essere

$$(a(x) = 4 + 3x + 2x^2 + 12x^3,\ b(x) = 4 + 3x + 4x^2 + 12x^3)$$

e ricavare s, in seguito all'introduzione dell'errore e sarà difficile.

Per estendere il problema ad m polinomi $a_1, \ldots, a_m \in R_q$ e m polinomi $e_1, \ldots, e_m \in R_q$ per gli errori, occorre considerare la matrice A data da $A = [Rot(a_1)|...|Rot(a_m)]$, da moltiplicare per il vettore s che contiene i coefficienti del polinomio s. Il risultato di questo prodotto andrà poi sommato al vettore e che contiene tutti i coefficienti degli m polinomi errore $e_1, \ldots, e_m$. Per ulteriori approfondimenti consultare [43] [28].

3.1.8 Module Learning With Errors (M-LWE)

Il problema Module-LWE generalizza sia LWE che Ring-LWE, ed è stato formalizzato in [18]. Per semplicità definiremo un'istanza del Module-LWE e ne mostreremo un esempio.

Module-LWE

Sia $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ l'anello dei polinomi in x, con relazione $x^n = -1$ e con coefficienti in $\mathbb{Z}_q$. Un'istanza del Module-LWE è data da $(A, b) \in R^{d\times m} \times R_q^d$, dove A è quindi una matrice di polinomi con d righe e m colonne. Avremo quindi $a_{i,j}(x) \in R_q$, $s_j(x) \in R_q$ e $e_i(x) \in R_q$ per ogni $i \in (1, \ldots, d)$ e per ogni $j \in (1, \ldots, m)$.

Diamo adesso un esempio in cui $n = 2$. Il grado dei polinomi sarà quindi pari a 1, le righe della matrice sono $d = 2$ e le colonne $m = 3$. L'equazione $As + e = b$ diventa:

$$\begin{pmatrix} a(x) & c(x) & d(x) \\ f(x) & g(x) & h(x) \end{pmatrix} \begin{pmatrix} s_1(x) \\ s_2(x) \\ s_3(x) \end{pmatrix} + \begin{pmatrix} e_1(x) \\ e_2(x) \end{pmatrix} = \begin{pmatrix} b_1(x) \\ b_2(x) \end{pmatrix} \in R_q^2\,.$$

Tutti i polinomi sono di grado 1, e sono quindi descritti da due coefficienti (a_0 e a_1, c_0 e c_1, etc.). Per ogni polinomio in A si può calcolare la matrice nega-circolante, ottenendo

$$\left(\begin{array}{c|c|c} \mathrm{Rot}(a(x)) & \mathrm{Rot}(c(x)) & \mathrm{Rot}(d(x)) \\ \hline \mathrm{Rot}(f(x)) & \mathrm{Rot}(g(x)) & \mathrm{Rot}(h(x)) \end{array}\right)$$

IL sistema si trasforma così nel sistema seguente, dove i coefficienti in s ed e sono i coefficienti di tutti i polinomi $s_j(x)$ e $e_i(x)$ e i coefficienti in b genereranno i polinomi $b_i(x)$

$$\left(\begin{array}{cc|cc|cc} a_0 & -a_1 & c_0 & -c_1 & d_0 & -d_1 \\ a_1 & a_0 & c_1 & c_0 & d_1 & d_0 \\ \hline f_0 & -f_1 & g_0 & -g_1 & h_0 & -h_1 \\ f_1 & f_0 & g_1 & g_0 & h_1 & h_0 \end{array}\right) \cdot \left(\begin{array}{c} s_{1,0} \\ s_{1,1} \\ \hline s_{2,0} \\ s_{2,1} \\ \hline s_{3,0} \\ s_{3,1} \end{array}\right) + \left(\begin{array}{c} e_{1,0} \\ e_{1,1} \\ \hline e_{2,0} \\ e_{2,1} \end{array}\right) = \left(\begin{array}{c} b_{1,0} \\ b_{1,1} \\ \hline b_{2,0} \\ b_{2,1} \end{array}\right) \pmod q$$

3.2 Dilithium

CRYSTALS-Dilithium è un nuovo algoritmo per la generazione di firme digitali che può resistere ai futuri attacchi informatici da parte dei computer quantistici. Come annunciato nel luglio 2022 [38], CRYSTALS-Dilithium è stato uno dei primi quattro vincitori nel progetto di standardizzazione della crittografia post-quantistica (PQC) gestito dal NIST.

Come membro della Cryptographic Suite for Algebraic Lattices (CRYSTALS), Dilithium è un algoritmo basato su reticoli che fornisce non solo sicurezza basata sulla difficoltà dei problemi su reticoli, ma anche una competitiva compressione della chiave pubblica e molteplici implementazioni efficienti. Questo algoritmo è inoltre raccomandato come scelta primaria tra gli altri due vincitori della competizione nell'ambito della firma digitale: FALCON e SPHINCS+. Infine lo standard di firma Module-Lattice-Based del NIST [40] sarà basato sulla versione 3.1 della specifica di Dilithium.

Lo schema di firma di Dilithium si basa su una tecnica chiamata **Fiat-Shamir with Aborts** [32]. Questa tecnica, proposta da Vadim Lyubashevsky nel 2009, ha aperto la porta a schemi di firma basati su reticoli che possono produrre firme digitali dell'ordine di $5 \cdot 10^4$ bit, con una notevole riduzione in spazio se confrontati con gli schemi precedenti basati sui reticoli che producevano firme dell'ordine di milioni di bit di lunghezza. Considerando che le firme digitali devono essere trasferite sulla rete insieme ai messaggi, ciò implica un vantaggio significativo rispetto alle tecniche precedenti. Per la sua sicurezza Dilithium invece si basa sulla difficoltà dei problemi **Module-LWE** e **Module-SIS**.

Per semplicità e per dare un'idea di come funziona lo schema di Dilithium andremo ad analizzare una versione semplificata e meno efficiente rispetto all'effettivo schema di firma, quest'ultimo è consultabile all'interno della documentazione ufficiale [12].

Livelli di sicurezza del NIST

Nella discussione dello schema di firma di Dilithium parleremo di "livelli di sicurezza". In particolare il NIST basa la sua classificazione degli schemi crittografici a chiave pubblica sugli intervalli di sicurezza offerti dagli standard di **crittografia simmetrica** già esistenti, sui quali il NIST si aspetta una significativa resistenza nei confronti della criptoanalisi quantistica. In particolare, il NIST definisce una categoria separata per ciascuno dei seguenti requisiti di sicurezza (elencati in ordine di forza crescente), definendo così i seguenti livelli di sicurezza [39]:

- **Livello 1**: qualsiasi attacco che infrange la definizione di sicurezza pertinente deve richiedere risorse computazionali paragonabili o superiori a quelle richieste per la ricerca di chiavi su un cifrario a blocchi con una **chiave a 128 bit** (ad es. AES128)
- **Livello 2**: qualsiasi attacco che infrange la definizione di sicurezza pertinente deve richiedere risorse computazionali paragonabili o superiori a quelle richieste per la ricerca di collisioni su una **funzione hash a 256 bit** (ad es. SHA256/ SHA3-256)
- **Livello 3**: Qualsiasi attacco che infrange la definizione di sicurezza pertinente deve richiedere risorse computazionali paragonabili o superiori a quelle richieste per la ricerca di chiavi su un cifrario a blocchi con una **chiave a 192 bit** (ad es. AES192)
- **Livello 4**: Qualsiasi attacco che infrange la definizione di sicurezza pertinente deve richiedere risorse computazionali comparabili o superiori a quelle richieste per la ricerca di collisione su una **funzione hash a 384 bit** (ad es. SHA384/ SHA3-384)
- **Livello 5**: Qualsiasi attacco che infrange la definizione di sicurezza pertinente deve richiedere risorse computazionali comparabili o superiori a quelle richieste per la ricerca di chiavi su un cifrario a blocchi con una **chiave a 256 bit** (ad es. AES 256)

Notazioni

Prima di spiegare la versione semplificata di Dilithium Figura 3.3, fissiamo alcune notazioni:

- q è un numero primo, in particolare per garantire la sicurezza si pone $q = 2^{23} - 2^{13} + 1 = 8380417$.
- $\mathbb{Z}_q$ indica l'insieme $\{0, \ldots, q-1\}$.
- $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ indica l'insieme dei polinomi in X con coefficienti in $\mathbb{Z}_q$ con relazione $X^n = -1$, per garantire la sicurezza si pone $n = 256$.

- S_η indica un sottoinsieme di R_q con coefficienti limitati da η.
- B_τ indica il sottoinsieme di R_q composto da polinomi con τ coefficienti uguali a $\pm$ 1 ed il resto a 0.
- H è una funzione hash con spazio di arrivo B_τ, in particolare è una funzione hash XOF (eXtendable Output Function) cioè una funzione hash crittografica che produce output di lunghezza arbitraria; Dilithium utilizza SHAKE-123 o SHAKE-256.

Come tutti gli schemi di firma si divide in tre fasi: Generazione delle chiavi, Firma e Verifica. D'ora in poi, all'interno dello schema di firma Dilithium, per identificare vettori, matrici utilizzeremo caratteri in grassetto, per gli scalari verranno utilizzati i caratteri in corsivo.

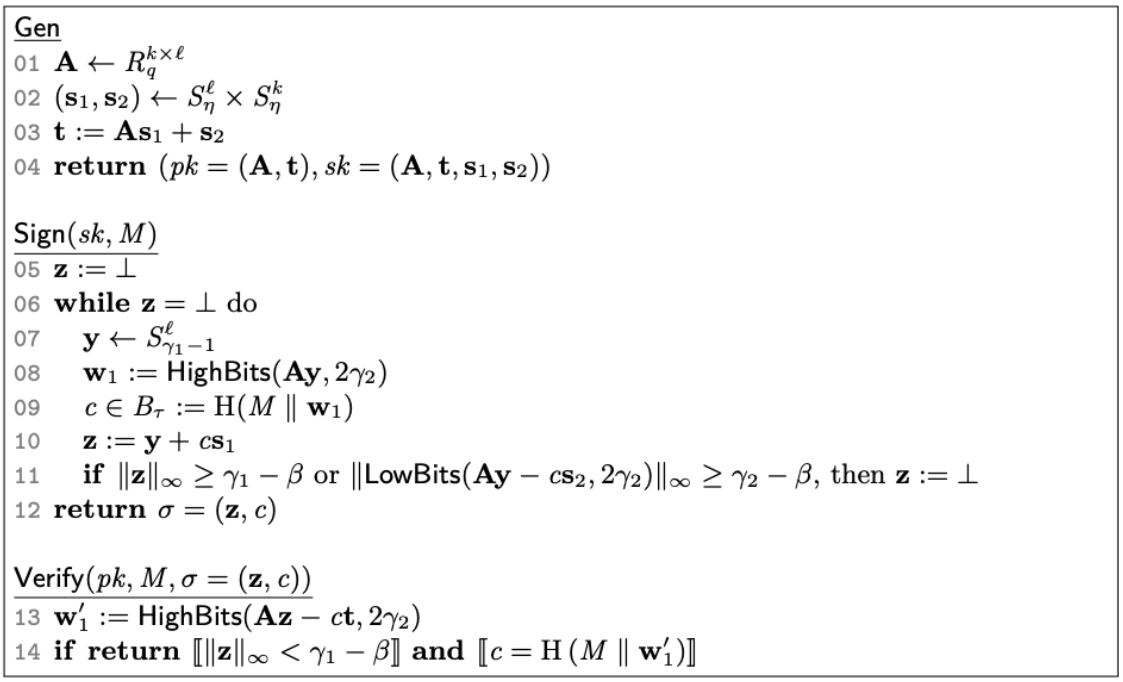


Figura 3.3: Modello dello schema di firma senza compressione della chiave pubblica [12]

.

Generazione delle chiavi

L'algoritmo di generazione delle chiavi campiona uniformemente una matrice $\mathbf{A}$ di dimensione $k \times l$, i cui elementi sono polinomi in $R_q = \mathbb{Z}_q[X]/(X^n + 1)$. Dopodiché campiona uniformemente anche i vettori $\mathbf{s_1}$ e $\mathbf{s_2}$ che comporranno la chiave privata; essendo campionati da S_η gli elementi di $\mathbf{s_1}$ e $\mathbf{s_2}$ saranno polinomi i cui coefficienti saranno limitati da η. Infine utilizzando i vettori $\mathbf{s_1}$, $\mathbf{s_2}$ e la matrice $\mathbf{A}$, viene calcolato il vettore $\mathbf{t} = \mathbf{A}\mathbf{s_1} + \mathbf{s_2}$ che insieme alla matrice $\mathbf{A}$ costituisce la chiave

pubblica. In questa fase vengono quindi generate la chiave pubblica $pk = (\mathbf{A}, \mathbf{t})$ e quella privata $sk = (\mathbf{A}, \mathbf{t}, \mathbf{s_1}, \mathbf{s_2})$.
Come si può osservare la generazione delle chiavi consiste nella creazione di un'istanza del Module-LWE. Infatti per risalire ai segreti $\mathbf{s_1}$ e $\mathbf{s_2}$ della chiave privata sk, data la chiave pubblica $(\mathbf{A}, \mathbf{t})$, un attaccante dovrebbe essere in grado di risolvere il problema Module-LWE, che abbiamo detto essere difficile e intrattabile, fornendo così la garanzia di sicurezza allo schema.

Firma

La fase di firma di Dilithium, come già anticipato utilizza la tecnica Fiat-Shamir with Aborts. Come prima cosa viene generato il vettore $\mathbf{y}$ di polinomi con coefficienti $< \gamma_1$. γ_1 è settato strategicamente in modo che sia grande abbastanza per garantire che la firma non riveli la chiave privata, e abbastanza piccolo in modo che la firma non possa essere facilmente falsificata. A seconda del livello di sicurezza che la firma Dilithium deve realizzare γ_1 assumerà valori differenti. Per un livello di sicurezza 2 avremo $\gamma_1 = 2^{17}$, per un livello di sicurezza 3 e 5 $\gamma_1 = 2^{19}$.

Il firmatario si calcola adesso il vettore di polinomi risultato del prodotto $\mathbf{Ay}$, ogni coefficiente w di questo vettore può essere scomposto come $w = w_1 \cdot 2\gamma_2 + w_0$ dove $|w_0| \leq \gamma_2$, γ_2 è quindi un limite superiore per il valore assoluto dello scalare w_0, il quale rappresenta la parte meno significativa del coefficiente w. γ_2, come γ_1, assumerà valori diversi a seconda del livello di sicurezza richiesto dallo schema di firma Dilithium, per un approfondimento su questi valori si consiglia di consultare la documentazione ufficiale [12]. La funzione HighBits preleva, da ogni coefficiente w del vettore risultante dal prodotto $\mathbf{Ay}$, le componenti w_1 e le inserisce in un vettore $\mathbf{w}_1$, dopodiché la challenge c è creata applicando la funzione hash H al messaggio M concatenato con $\mathbf{w}_1$.

La challenge c in particolare sarà un polinomio in B_τ, cioè un polinomio con τ coefficienti $\pm$ 1 e il resto 0. Adesso la potenziale firma viene costruita come $\mathbf{z} = \mathbf{y} + c\mathbf{s_1}$; nel prodotto $c\mathbf{s_1}$, la challenge c è un polinomio e va a moltiplicare ogni polinomio del vettore $\mathbf{s_1}$ si noti inoltre che $\mathbf{s_1}$ è parte della chiave privata.

Prima di rilasciare $\mathbf{z}$ come parte della firma si controlla che $\mathbf{s_1}$ non sia ricavabile da $\mathbf{z}$, rendendo così lo schema insicuro. Questo può succedere per scelte sfortunate del vettore $\mathbf{y}$. Per evitare di rilasciare $\mathbf{z}$ nel caso in cui si possa ricavare $\mathbf{s_1}$ da esso, si applica quello che è chiamato il "*rejection sampling*", procedura con la quale si controlla che alcuni parametri non escano da un regime di sicurezza; per completare lo schema Fiat-Shamir with Aborts è necessario effettuare questo controllo, se il controllo non è superato con successo si ripete dall'inizio tutta la procedura di firma. I controlli effettuati sono i seguenti

$$||\mathbf{z}||_\infty \geq \gamma_1 - \beta \quad \text{e} \quad ||\text{LowBits}(\mathbf{Ay} - c\mathbf{s_2}, 2\gamma_2)||_\infty \geq \gamma_2 - \beta\,,$$

se una delle due disequazioni è valida si ripete lo schema di firma.

In particolare il parametro β è settato per essere il massimo coefficiente possibile del prodotto $c\mathbf{s_i}$, dove $\mathbf{s_i}$ è un polinomio in $\mathbf{s_1}$. Poiché c ha τ coefficienti uguali a ± 1 e il coefficiente massimo in $\mathbf{s_i}$ è η, si può facilmente comprendere che $\beta \leq \tau \cdot \eta$. Ricordando che $\|\mathbf{z}\|_\infty := \max_i |z_i|$ se in $\mathbf{z}$ c'è un coefficiente maggiore di $\gamma_1 - \beta$, si scarta la firma $\mathbf{z}$ e si ripete la procedura di firma.

Nel secondo controllo si utilizza la funzione LowBits, la quale restituirà, al contrario di HighBits, un vettore $\mathbf{w_0}$ che conterrà tutte le componenti w_0 dei coefficienti w nel vettore di polinomi risultante dell'operazione $\mathbf{Ay} - c\mathbf{s_2}$.

Se applicando la funzione LowBits a $\mathbf{Ay} - c\mathbf{s_2}$ viene restituito un vettore $\mathbf{w_0}$ che ha un coefficiente maggiore di $\gamma_2 - \beta$ (poiché si applica la norma infinito a questo vettore) allora $\mathbf{z}$ viene scartato e si ripete la procedura di firma. Questi due controlli costituiscono quello che avevamo chiamato *rejection sampling*, che evita di far rilasciare $\mathbf{z}$ nel caso in cui da esso sia possibile ricavare il segreto $\mathbf{s_1}$.

Il primo controllo viene fatto per la sicurezza, il secondo sia per la correttezza che la sicurezza. Alla fine della procedura di firma si ottiene la firma come $\sigma = (\mathbf{z}, c)$.

Verifica della firma

Colui che riceve il messaggio M e la firma σ deve compiere alcune operazioni per verificarla. Come prima cosa, calcola $\mathbf{w_1'}$ applicando la funzione HighBits al vettore ottenuto da $\mathbf{Az} - c\mathbf{t}$ e considererà la firma valida se tutti i coefficienti di $\mathbf{z}$ sono più piccoli di $\gamma_1 - \beta$ e se $c = H(M||\mathbf{w_1})$ è uguale a c', dove $c' = H(M||\mathbf{w_1'})$

Dimostriamo adesso che la verifica effettuata in questo modo è corretta, in particolare che la firma è verificata se

$$c = c'$$

ricordando che $c = H(M||\mathbf{w_1})$ e $c' = H(M||\mathbf{w_1'})$ l'equazione sovrastante diventa

$$H(M||\mathbf{w_1}) = H(M||\mathbf{w_1'})$$

che è come dimostrare che

$$\mathbf{w_1} = \mathbf{w_1'}$$

cioè

$$\text{HighBits}(\mathbf{Ay}, 2\gamma_2) = \text{HighBits}(\mathbf{Az} - c\mathbf{t}, 2\gamma_2)$$

si noti che $\mathbf{z}$ e $\mathbf{t}$ sono definiti come segue

$$\mathbf{z} = \mathbf{y} + c\mathbf{s_1} \quad \text{e} \quad \mathbf{t} = \mathbf{As_1} + \mathbf{s_2}$$

quindi

$$\mathbf{Az} - c\mathbf{t}$$

diventa

$$\mathbf{A}(\mathbf{y} + c\mathbf{s_1}) - c(\mathbf{As_1} + \mathbf{s_2})$$

$$\mathbf{Ay}+\mathbf{A}c\mathbf{s_1}-\mathbf{A}c\mathbf{s_1} - c\mathbf{s_2}$$

$$\mathbf{Ay} - c\mathbf{s_2}$$

quindi $\mathbf{Az} - c\mathbf{t} = \mathbf{Ay} - c\mathbf{s_2}$ e possiamo quindi scrivere:

$$\text{HighBits}(\mathbf{Ay}, 2\gamma_2) = \text{HighBits}(\mathbf{Ay} - c\mathbf{s_2}, 2\gamma_2) \tag{3.2}$$

Quest'ultima equazione è vera perché una firma valida avrà

$$||\text{LowBits}(\mathbf{Ay} - c\mathbf{s_2}, 2\gamma_2)||_\infty \leq \gamma_2 - \beta,$$

condizione che ricordiamo aver stabilito nella fase di firma. E poiché sappiamo che i coefficienti di $c\mathbf{s_2}$ sono più piccoli di β, anche questa è una condizione stabilita nella fase di firma, possiamo dire che sottrarre $c\mathbf{s_2}$ ad $\mathbf{Ay}$ non è sufficiente a provocare variazione nella valutazione di $\text{HighBits}(\mathbf{Ay} - c\mathbf{s_2}, 2\gamma_2)$ rispetto a $\text{HighBits}(\mathbf{Ay}, 2\gamma_2)$ e quindi l'equazione (3.2) risulta valida.

Parametri

A seconda del livello di sicurezza che si vuole che implementi lo schema di firma Dilithium, si avranno valori diversi per i parametri che lo caratterizzano; questi valori sono raccolti nella Tabella 3.1.

Livello di sicurezza del NIST	2	3	5
Parametri			
q [modulo]	8380417	8380417	8380417
τ[#di $\pm$ in c]	39	49	60
γ_1 [intervallo dei coefficienti di $\mathbf{y}$]	2^{17}	2^{19}	2^{19}
γ_2 [limite superiore per w_0]	$(q-1)/88$	$(q-1)/32$	$(q-1)/32$
(k, l) [dimensione di $\mathbf{A}$]	(4, 4)	(6, 5)	(8, 7)
η [intervallo per i coefficienti di $\mathbf{s_1}$ e $\mathbf{s_2}$]	2	4	2
β $[\tau \cdot \eta]$	78	196	120

Tabella 3.1: Parametri per Dilithium [12]

Osservazioni

A differenza di altri schemi, basati su reticoli, partecipanti al processo di standardizzazione del NIST, i vettori segreti $\mathbf{s_1}$ e $\mathbf{s_2}$ di Dilithium sono campionati in modo uniforme e non seguendo una distribuzione Gaussiana discreta. Gli sviluppatori di CRYSTALS- Dilithium [21] hanno intenzionalmente evitato l'uso del campionamento seguendo la distribuzione Gaussiana discreta nel loro schema perché è difficile da implementare correttamente, è ancora più difficile ottenere un'esecuzione in

tempo costante, ed inoltre espone lo schema a rischio di particolari attacchi *side-channel*. Questa omissione ha contribuito a una delle principali considerazioni di progettazione: lo schema doveva essere facile da implementare in modo sicuro.

Uno degli obbiettivi principali degli autori di Dilithium era quello di costruire uno schema di firma che minimizzasse le dimensioni di chiave pubblica e firma di un messaggio. Dilithium sfruttando il lavoro [13], nel quale si introduce una nuova tecnica, più performante, di compressione dei dati basata su LWE, è riuscito a implementare una compressione delle firme digitali, riducendo le dimensioni delle firme di oltre il 50% inviando essenzialmente solo circa la metà dei dati di firma sulla rete. Inoltre al suo interno è stata introdotta una compressione della chiave pubblica, ottenendone così una che è il 60% più piccola rispetto agli schemi precedenti, con solo un leggero aumento delle dimensioni della firma digitale (di 100 byte). Tenendo presente il vantaggio introdotto con l'utilizzo di Fiat-Shamir With Aborts di ridurre la dimensione della firma da milioni di bit a 50.000 bit, un aumento di 100 byte si può considerare trascurabile.

Capitolo 4

Firme basate su hash e SPHINCS+

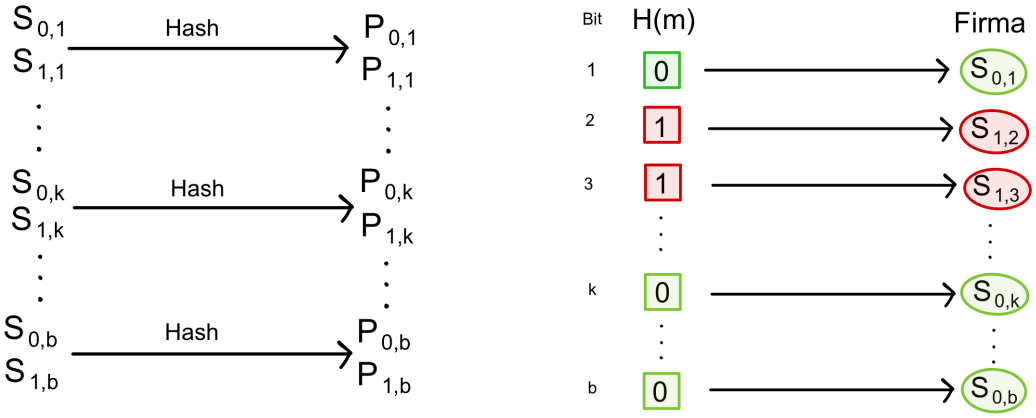
Come già anticipato, tra i primi vincitori del processo di standardizzazione del NIST per la selezione di algoritmi resistenti alla computazione quantistica, nel campo della firma digitale, sono stati scelti CRYSTALS-Dilithium, FALCON e SPHINCS+. Se però i primi due basano la loro sicurezza sulla difficoltà di problemi basati sui reticoli, SPHINCS+ basa la sua sicurezza su funzioni hash le quali, come abbiamo già spiegato quando abbiamo introdotto l'algoritmo di Grover nella Sezione 1.2.2, sono sicure nei confronti da attacchi da parte dei computer quantistici. Prima di analizzare le caratteristiche di SPHINCS+, introduciamo altri algoritmi che utilizzano funzioni hash per costruire firme digitali che hanno posto le basi per la costruzione di SPHINCS+.

4.1 Schema di firma di Lamport

La prima firma digitale basata su hash, chiamata firma Lamport, fu introdotta da Leslie Lamport nel 1979 [27]. Nello schema di firma di Lamport si considera una funzione hash che produce un valore hash di b bit; ad esempio se la funzione hash utilizzata è SHA-256 avremo $b = 256$. La funzione hash H viene applicata al messaggio m da firmare, generando così un digest $H(m)$ di b bit. Quindi per ciascun bit in $H(m)$, si generano due stringhe segrete di b bit ciascuna: $S_{0,k}$ e $S_{1,k}$, dove $k \in \{1, .., b\}$ indica la posizione del bit nel digest $H(m)$. Tutte queste stringhe segrete costituiscono la **chiave privata**. La **chiave pubblica** si ricava da quella privata applicando la funzione hash a ciascuna stringa segreta: $H(S_{0,k}), H(S_{1,k})$ con $k \in \{1, .., b\}$, come illustrato nella Figura 4.1a.

Adesso per generare la firma del messaggio m, si effettua il seguente passaggio per tutti i bit di $H(m)$: se il k-esimo bit di $H(m)$ è 0, allora il k-esimo elemento della firma è $S_{0,k}$, se invece il k-esimo bit è 1, allora il k-esimo elemento della firma è $S_{1,k}$. Si noti che la firma rivela metà della chiave privata. Per una migliore comprensione visionare la Figura 4.1b.

Per la verifica della firma, il verificatore calcola $H(m)$, i cui bit $(0, 1)$ serviranno per selezionare le stringhe $P_{0/1,k}$ della chiave pubblica associata alla chiave privata utilizzata nel processo di generazione della firma. Adesso avendo la firma, composta da una parte delle stringhe della chiave privata, il verificatore applica la funzione hash a ognuna delle b stringhe che compongono la firma, per ricavarne le stringhe che costituiscono la chiave pubblica associata. Se la chiave pubblica calcolata in questo modo corrisponde a quella calcolata inizialmente dal verificatore, a partire da $H(m)$, allora la firma è verificata.



(a) Calcolo chiave pubblica P

(b) Generazione della firma per il messaggio m

Figura 4.1

Questo schema presenta una serie di svantaggi. Come già osservato la firma del messaggio rivela metà della chiave privata; quindi ogni chiave deve essere utilizzata una sola volta poiché riutilizzarla più volte non sarebbe sicuro, in quanto si rivelerebbe gran parte della chiave privata, se non tutta, permettendo a un attaccante di firmare un qualsiasi messaggio a sua scelta.

L'altro svantaggio è che la chiave pubblica, quella privata e la firma hanno grandi dimensioni. Supponendo di firmare il messaggio con la funzione hash SHA-256, e che la dimensione di ogni stringa che compone la chiave privata e quella pubblica sia lunga 256 bits, otterremo le seguenti dimensioni:

- $H(m)$ di 256 bits
- chiave privata: 2 chiavi (una per il bit 0 e una per il bit 1), di lunghezza 256 bit, per ognuno dei 256 bit di $H(m)$ $(2 \cdot 256 \cdot 256 = 131072$ bits)
- chiave pubblica: 2 chiavi, di lunghezza 256 bit, per ognuno dei 256 bit di $H(m)$ $(2 \cdot 256 \cdot 256 = 131072$ bits)
- firma: per ognuno dei 256 bit di $H(m)$ scelgo una chiave privata da 256 bit $(256 \cdot 256 = 65536$ bits)

Questa tipologia di firma, che consente l'utilizzo della coppia di chiavi una sola volta, per mantenere la sicurezza, viene chiamata *"**one-time signature**"*.

4.2 Schema di firma Winternitz

Per risolvere il problema della lunghezza delle chiavi e della firma nello schema di firma Lamport, nel 1989 Robert Winternitz ha proposto lo schema di firma Winternitz [36], chiamato anche *Winternitz One Time Signature* (WOTS), in quanto anche esso come lo schema di Lamport può utilizzare una coppia di chiavi una sola volta per mantenere la sua sicurezza. Nonostante ciò, WOTS può essere considerato un miglioramento dello schema di firma di Lamport poiché utilizza chiavi e firma di dimensioni inferiori.

WOTS si basa sul concetto di *"**catene hash**"* nelle quali una funzione hash è applicata più volte in successione. In questo schema di firma, al contrario di quello Lamport dove si genera una coppia di chiavi per ogni bit del digest $H(m)$ del messaggio m, il digest viene diviso in "blocchi" e si genera una coppia di chiavi per ogni blocco. Il numero di blocchi in cui dividere il digest dipende da un parametro che parametrizza tutto lo schema WOTS, questo parametro è indicato di solito con w, che sta per "*parametro di Winternitz*", ed indica il *numero di bit di ogni blocco*.

Nella fase di generazione delle chiavi, come già anticipato, si generano in modo casuale con un generatore random, tante chiavi private quanti i blocchi in cui abbiamo diviso il digest; le chiavi pubbliche vengono generate da quelle private, applicando 2^w volte la funzione hash a ogni blocco di chiave privata, questo è proprio il concetto di catena hash. La Figura 4.2 mostra come dato un digest di 8 bit, con $w = 4$ (il digest viene quindi diviso in 2 blocchi) si generano 2 chiavi private dalle quali in seguito all'applicazione, su ognuna di esse, della funzione hash per $2^w = 2^4 = 16$ volte, si ottengono le due chiavi pubbliche associate.

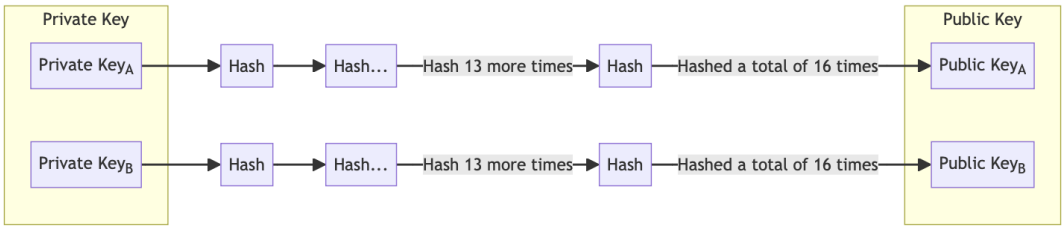


Figura 4.2: Generazione chiave pubblica con parametro $w = 8$ e lunghezza del digest $H(m)$ del messaggio m da firmare di 8 bit [45]

Il parametro w può essere settato in modo da scegliere il compromesso che si desidera tra la *dimensione* di ogni blocco (al crescere di w diminuisce il numero di blocchi, ma cresce la loro dimensione) e il *lavoro computazionale* (al crescere di w diminuisce il numero di blocchi, ma aumenta il numero 2^w di applicazioni della funzione hash).

Per firmare il messaggio si effettuano i seguenti passaggi:

- Si valuta il valore decimale rappresentato da ogni blocco del digest $H(m)$
- Ad ogni blocco del digest si applica la funzione hash tante volte quanto il valore decimale che è codificato da quel particolare blocco, ottenendo così tutti i blocchi della firma.

Riallacciandoci all'esempio precedente, supponiamo che il primo blocco, composto dai bits da 1 a 4, codifichi in decimale il valore $N = 6$. Il primo blocco della firma, relativo al primo blocco del digest, si ottiene applicando $N = 6$ volte la funzione hash al primo blocco della **chiave privata**. Questo procedimento viene ripetuto per tutti i blocchi del digest, per generare l'intera firma.

Il verificatore, per verificare la firma effettua i seguenti passaggi:

- Come prima cosa deve applicare la funzione hash al messaggio m ottenendo così il digest $H(m)$.
- Divide il digest in blocchi di dimensione w e per ogni blocco si calcola il valore decimale N corrispondente
- Per verificare la firma ricevuta, applica la funzione hash ad ogni blocco della firma un totale di $2^w - N$ volte (dove N sarà peculiare per ogni blocco)
- Se il risultato di questa ultima operazione produce un output che coincide con i blocchi che costituiscono la chiave pubblica allora la firma è verificata, altrimenti viene rifiutata.

Possiamo osservare che in questo schema la chiave privata è l'inizio della catena di hash, la firma un valore intermedio e la chiave pubblica l'ultimo valore della catena, come illustrato nella Figura 4.3.

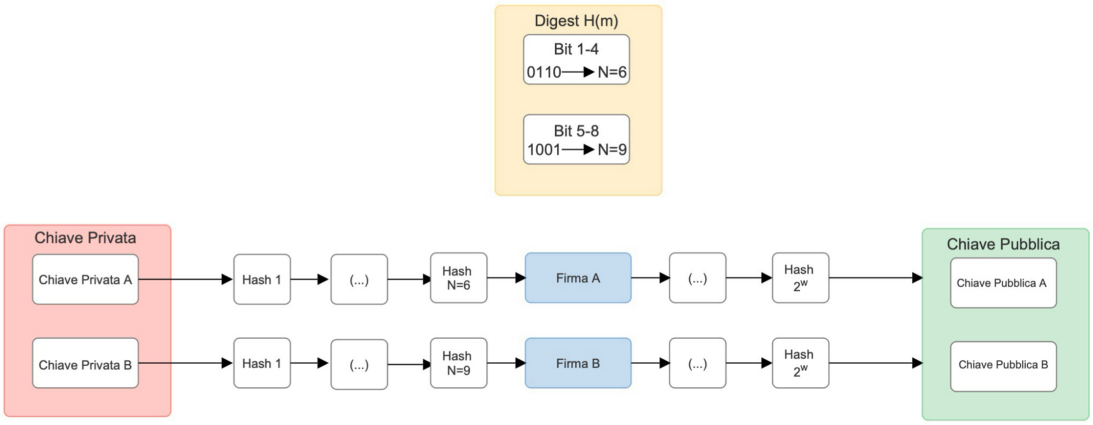


Figura 4.3: Catena di hash dalla chiave privata, alla firma, alla chiave pubblica

Il compromesso che fa WOTS è quello di aggiungere maggiore computazione nella fase di firma e di verifica, ottenendo però firme e chiavi più corte rispetto a quelle di Lamport.

Nella Tabella 4.1 riportiamo un confronto tra i due schemi relativo ad un esempio in cui si assume che i blocchi che costituiscono la chiave privata siano ognuno da 256 bit, si sia utilizzata la funzione hash SHA-256, il messaggio da firmare sia lungo 8 bit, e infine il parametro w sia uguale a 4.

	Lamport	**WOTS**
Chiave Privata	$8 \cdot 2 \cdot 256 = 4096$ bits	$2 \cdot 256 = 512$ bits
Chiave Pubblica	$8 \cdot 2 \cdot 256 = 4096$ bits	$2 \cdot 256 = 512$ bits
Firma	$8 \cdot 256 = 2048$ bits	$2 \cdot 256 = 512$ bits
Totale da spedire	$4096 + 2048 = 6144$ bits	$512 + 512 = 1024$ bits

Tabella 4.1: Confronto dimensioni firma WOTS con Lamport

Come si può osservare WOTS è ancora una **one-time signature**, perché una volta che una chiave è stata utilizzata, un attaccante può produrre un messaggio e firmarlo, anche non conoscendo la chiave privata. Questo però è possibile se il digest del messaggio fraudolento ha tutti i blocchi con valori decimali uguali o maggiori dei corrispettivi blocchi del primo messaggio firmato. Questo è possibile poiché basterebbe applicare qualche altra volta in più la funzione hash alla firma del primo messaggio, per ottenere una firma valida sul messaggio fraudolento.

4.2.1 WOTS+

SPHINCS+ al suo interno sfrutta lo schema di firma WOTS+ [25] che è una variante di WOTS pubblicata da Andreas Hülsing nel 2013, con la quale viene utilizzata una funzione hash che prende in input oltre che la chiave privata anche un elemento di randomizzazione. Con WOTS+ si riescono ad avere delle firme ancora più piccole e ad incrementare la sicurezza rispetto alla versione base WOTS, ma rimane comunque una **one-time signature**. Essendo WOTS+ una variante dello schema WOTS già definito, non lo discuteremo in questa tesi, ma rimandiamo il lettore alla documentazione ufficiale [25].

4.3 Schema di firma con alberi di Merkle

Abbiamo visto che gli schemi di firma Lamport, WOTS e WOTS+ sono del tipo **one-time signature** cioè possono firmare un solo un messaggio con la stessa chiave; se occorre firmare più messaggi, si dovranno utilizzare più chiavi. Ralph Merkle ha proposto una tecnica chiamata *Merkle Tree Signature Scheme (MSS)* [35] per gestire le chiavi dello schema di firma di Lamport, utilizzando il concetto di albero hash. Questo schema consente al firmatario di precalcolare una serie di coppie di chiave

pubblica e privata che possono essere utilizzate per generare firme verificabili con una stessa chiave pubblica. L'idea base è di utilizzare le foglie di un albero di Merkle per memorizzare le chiavi pubbliche pregenerate (per l'esattezza l'hash delle chiavi pubbliche).

Un albero di Merkle è un albero binario bilanciato; quindi se la sua altezza è n allora avrà 2^n foglie, riuscendo così a gestire $N = 2^n$ chiavi pubbliche. Una volta generate le N chiavi private X_i con $0 \leq i \leq N-1$, si calcolano le corrispondenti chiavi pubbliche $Y_i = H(X_i)$, $0 \leq i \leq N-1$. Si ricordi che nello schema di Lamport, ogni chiave pubblica è generata dalla concatenazione di $2\,b$ valori hash, ottenuti applicando una funzione hash alle $2\,b$ chiavi private $S_{0,k}$ e $S_{1,k}$ dove $k \in \{1, .., b\}$, e b indica la lunghezza in bit del digest $H(m)$.

L'albero di Merkle si costruisce partendo dalle foglie, che contengono l'hash $H(Y_i)$ delle chiavi pubbliche Y_i pregenerate. Ciascun nodo dell'albero è etichettato come $h_{i,j}$ dove i denota il livello del nodo e corrisponde alla distanza dal nodo da una foglia, quindi una foglia risiede al livello 0 e la radice dell'albero al livello n. La Figura 4.4 mostra un albero con altezza $n = 3$.

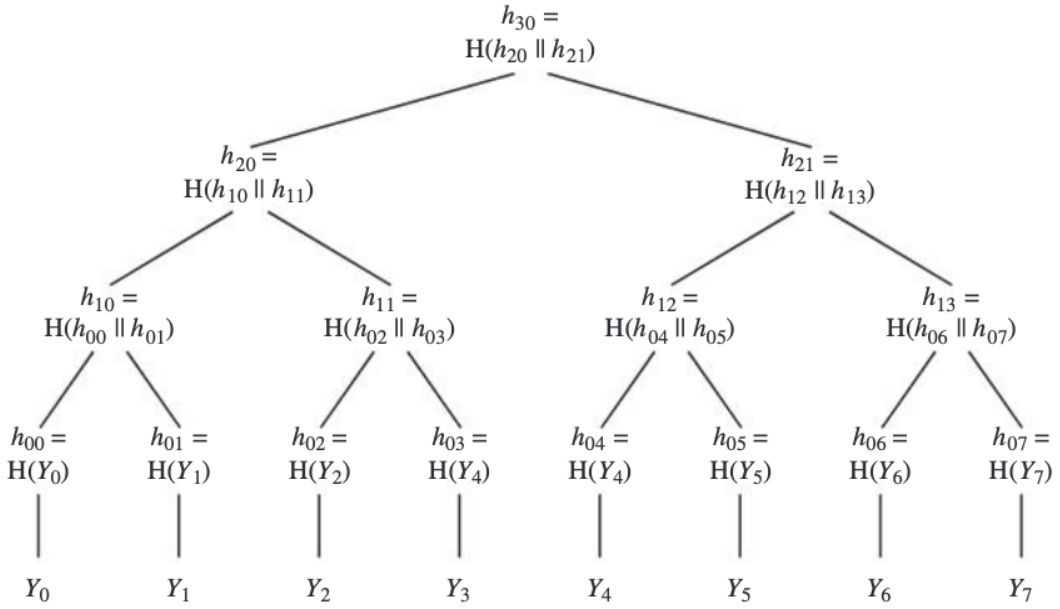


Figura 4.4: Albero di Merkle di altezza $n = 3$ le cui $2^n = 8$ foglie contengono l'hash delle chiavi pubbliche (Y_i) [52].

Ciascuna coppia di valori hash in un livello viene concatenata e viene calcolato l'hash di questa concatenazione (es $H(h_{0,0}||h_{0,1}) = h_{1,0}$) che andrà a formare un nodo del livello superiore. Il processo continua fin quando risulterà un singolo valore, noto come radice di Merkle ($h_{3,0}$ nella Figura 4.4). La radice di Merkle diventa la **singola chiave pubblica**, chiamata anche chiave pubblica generale, che consente di verificare fino a N firme. Ciò ha due vantaggi: la chiave pubblica è abbastanza piccola (dimensione dell'output della funzione hash utilizzata) e può essere utilizzata per verificare più firme.

Il processo di firma per un messaggio m_i è il seguente: Si genera la firma di Lamport LS_i, seguendo il procedimento già illustrato, per il messaggio m_i, scegliendo una delle coppie chiave privata e chiave pubblica (X_i, Y_i) precalcolate. In particolare si seleziona una coppia che non è stata ancora utilizzata e tale che l'hash della chiave pubblica è memorizzato in una foglia nell'albero.

Si osservi tuttavia che in questo schema il verificatore conosce solo la chiave pubblica generale, che corrisponde al valore hash nella radice dell'albero di Merkle (ad esempio il valore h_{30} in Figura 4.4), e non conosce la chiave pubblica Y_i associata alla chiave privata X_i utilizzata per firmare. Per questo motivo, anche la chiave pubblica Y_i viene fornita come parte della firma.

Inoltre il verificatore deve essere in grado di autenticare la chiave pubblica Y_i, ossia verificarne la validità. Nella firma dovrà essere quindi incluso anche il cosiddetto *"percorso di autenticazione"* per la chiave pubblica Y_i, illustrato nella Figura 4.5. Questo percorso è costituito da tutti gli hash dei nodi intermedi dell'albero, necessari per calcolare l'hash della radice, ovvero la chiave pubblica generale, a partire dalla foglia $H(Y_i)$.

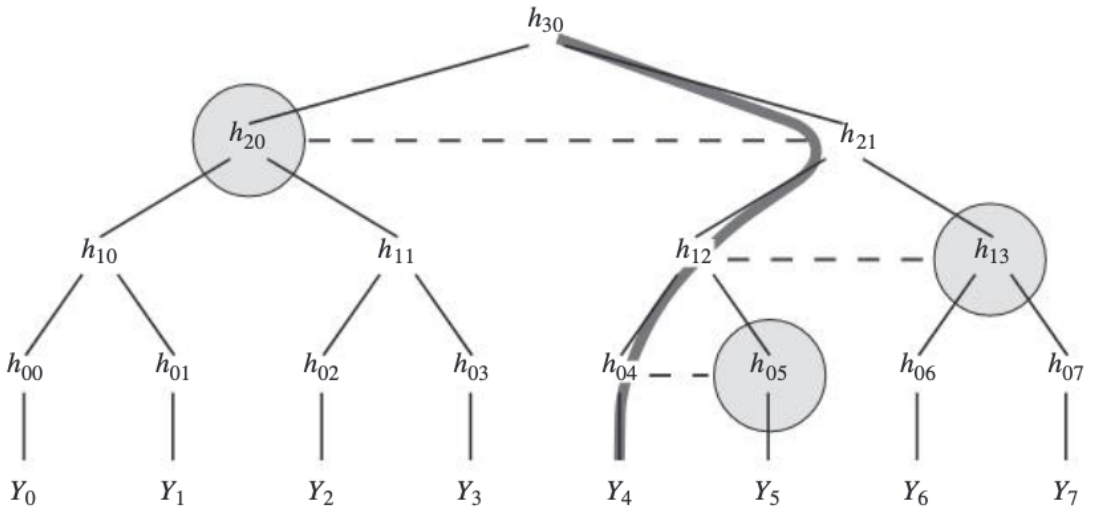


Figura 4.5: Il percorso di autenticazione per la chiave pubblica Y_4 è $h_{0,5}$, $h_{1,3}$,$h_{2,0}$) [52]

Per esempio se la chiave privata per firmare il messaggio m_4 è X_4 la firma avrà la seguente struttura

$$\sigma(m_4) = (LS_4, Y_4, h_{0,5}, h_{1,3}, h_{2,0}) \,,$$

dove LS_4 indica la firma di Lamport generata usando X_4. Il verificatore, per autenticare la chiave pubblica dovrà calcolarsi

$$H(H(H(H(Y_4)||h_{0,5})||h_{1,3})||h_{2,0})$$

Se tale valore equivale alla chiave pubblica generale $h_{3,0}$, allora Y_4 è autenticato e può essere utilizzato per verificare la firma LS_4.

Lo stesso concetto di utilizzare un albero hash di Merkle per gestire le chiavi di Lamport, può essere applicato anche per gestire le chiavi di WOTS o WOTS+. Due algoritmi popolari che lo fanno sono eXtended Merkle Signature Scheme (XMSS) [24] e Leighton–Micali Signatures (LMS) [30].

Dato che negli schemi di firma basati su hash è fondamentale assicurarsi che ogni chiave venga utilizzata una sola volta, il firmatario deve ricordarsi quali sono le coppie chiave private e chiave pubblica che ha già utilizzato per firmare i messaggi precedenti. Per questo motivo, questi schema di firma sono detti "**stateful**".

Un problema frequente è che prima o poi può succedere che avvengano degli errori nel processo di memorizzazione delle chiavi già utilizzate, portando al riuso di chiavi e quindi al fallimento dello schema. Per esempio questo può succedere perché il codice del programma non aggiorna correttamente le chiavi già utilizzate, o perché la memoria non le salva in maniera opportuna. Anche se queste situazioni sono molto rare, visto l'utilizzo in grande scala della crittografia, gli schemi di firma di tipo stateful falliranno ad un certo punto del loro utilizzo. Questo è il motivo per cui il NIST [37] non ha approvato l'uso generale delle firme stateful, comunicando che gli schema di firma basati su hash stateful sono sicuri contro lo sviluppo di computer quantistici, ma non sono adatti all'uso generale perché la loro sicurezza dipende dalla gestione attenta dello stato.

4.4 HORS una few time signature

Hash to Obtain Random Subset (HORS) [47] è uno schema di firma "**few time signature**" in quanto permette di firmare più messaggi utilizzando la stessa chiave. Il numero di riutilizzi della chiave è limitato in questa tipologia di firma perché ogni riutilizzo espone informazioni sulla chiave, riducendone la sicurezza.

HORS come prima cosa applica una funzione hash H al messaggio m che deve essere firmato; a questo punto il digest $H(m)$ viene diviso in blocchi. Supponiamo che il digest $H(m)$ venga diviso in N blocchi $H(m) = (m_1, \ldots, m_N)$ ognuno di l bit. A questo punto vengono generate $t = 2^l$ chiavi private $SK = (sk_0, \ldots, sk_{t-1})$ utilizzando un generatore casuale. Applicando ad ogni elemento sk_i della chiave privata una funzione hash, si ottiene la chiave pubblica $PK = (pk_0, \ldots, pk_{t-1})$.

Avendo a disposizione le chiavi si può generare una firma HORS nel seguente modo:

- Per ogni blocco m_i del digest del messaggio, si valuta il suo valore espresso in notazione decimale
- La firma sull'i-esimo blocco corrisponde alla chiave privata la cui posizione è pari al valore decimale dell'i-esimo blocco. Per esempio, se il valore decimale del primo blocco del digest del messaggio è 5 ($m_1 = 5$) la firma su questo blocco corrisponde alla chiave privata $sk_{m_1} = sk_5$.

- L'intera firma viene generata ripetendo il passaggio precedente su ogni blocco del digest del messaggio, ottenendo così la firma $\sigma(m) = (\sigma_1, \ldots, \sigma_N) = (sk_{m_1}, \ldots, sk_{m_N})$.

In particolare nella Figura 4.6 è mostrato come si firma un messaggio, applicando una funzione hash che restituisce un digest di 256 bit. Il numero di blocchi in cui è stato deciso di dividere il digest è $N = 16$, quindi la dimensione di ogni blocco in bit è $l = 16$ che comporta $2^l = 2^{16} = 65535$ chiavi private.

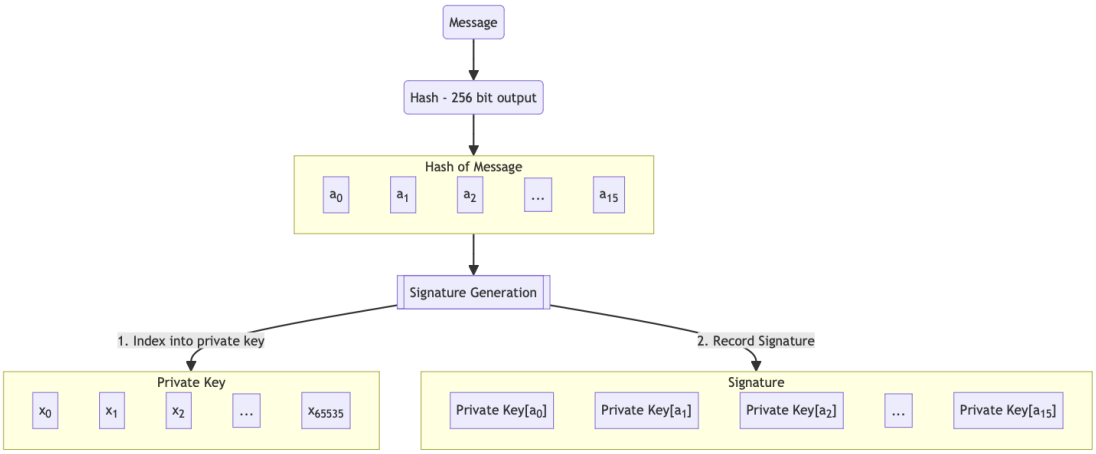


Figura 4.6: Generazione firma HORS [45]

Il verificatore, per verificare la firma, applica nuovamente la funzione hash al messaggio m, ottenendo così il digest $H(m)$, il quale viene diviso in N blocchi. Si calcola quindi il valore decimale di ogni blocco, e lo si usa come indice per identificare la chiave pubblica associata a quel blocco, da utilizzare per la verifica della sua firma. Questo procedimento viene applicato a tutti i blocchi, ottenendo una chiave pubblica $PK = (pk_{m_1}, \ldots, pk_{m_N})$. Dopodiché la stessa funzione hash viene applicata a tutti i blocchi della firma $(\sigma_1, \ldots, \sigma_N) = (sk_{m_1}, \ldots, sk_{m_N})$, che in sostanza sono chiavi private, ottenendo così blocchi di chiavi pubbliche $PK' = (pk_{m_1}, \ldots, pk_{m_N})$; se $PK' = PK$ allora la firma è verificata, altrimenti viene rifiutata.

HORS è una *few time signature* perché ogni volta che si firma un messaggio si "perdono" meno informazioni sulla chiave privata utilizzata rispetto ad una firma Lamport. Questo perché per firmare un messaggio, il cui digest è stato diviso in N blocchi da l bit, si utilizzano N chiavi pk_i sulle $t = 2^l$ totali, che è una quantità molto piccola rispetto all'utilizzo di una firma Lamport con la quale si rivelerebbe metà della chiave privata utilizzata.

HORS ha alcuni aspetti negativi come la dimensione, molto lunga, della chiave pubblica e il fatto che potenzialmente in una firma può capitare di ripetere più volte la stessa chiave sk_i. Questo è dovuto al fatto che blocchi diversi del digest possono codificare lo stesso numero decimale. Un utente malintenzionato, che controlla il messaggio su cui desidera forgiare una firma, può risparmiare del lavoro cercando di trovare un messaggio in cui più blocchi hanno lo stesso valore decimale, riducendo

così il numero di blocchi di chiave privata che devono essere rivelati per permettere di falsificare una firma.

4.4.1 HORST: HORS Tree

Per ridurre la dimensione della chiave pubblica in HORS è stato creato lo schema di firma HORST (HORS Tree), che consiste nell'applicazione dell'albero di Merkle ad HORS, risolvendo il problema della distribuzione delle lunghe chiavi di HORS. Questo è possibile assegnando ogni blocco della chiave pubblica di HORS alle foglie di un albero di Merkle, cosicché sia sufficiente distribuire solo la radice dell'albero di Merkle come chiave pubblica, di dimensioni nettamente minori. Per raggiungere questo risultato vengono tuttavia complicati il processo di firma e di verifica: ogni firma dovrà adesso includere blocchi di chiave pubblica e il percorso di autenticazione. Inoltre questo schema HORST, nonostante introduca dei miglioramenti, non fa nulla per risolvere il fatto che una firma possa contenere più volte lo stesso blocco sk_i della chiave privata [45].

4.5 Forest of Random Subset

Forest of Random Subset (FORS) [11] è uno schema di firma della tipologia *few time signature* utilizzato all'interno dello schema di firma di SPHINCS+. Si tratta di un miglioramento di HORST, ed ha una struttura a *hypertree*, cioè è un albero di alberi. Descriviamo nel seguito il suo funzionamento a grandi linee.

Come HORST, FORS calcola il digest $H(m)$ del messaggio m, lo divide in N blocchi da l bit e calcola $N \cdot 2^l$ chiavi private, dalle quali applicando la funzione hash, si ottengono $N \cdot 2^l$ chiavi pubbliche, che vanno a formare le foglie di N alberi all'interno dell'hypertree. Per una maggiore comprensione si consulti la Figura 4.7.

Supponiamo di dividere il digest in $N = 6$ blocchi, ognuno di $l = 3$ bits. Lo schema di firma FORS genera quindi $N \cdot 2^l = 6 \cdot 2^3 = 48$ chiavi private e 48 chiavi pubbliche. Per firmare ad esempio un messaggio m, il cui digest è $H(m) = 100|011|011|101|111|000$, per il *primo* blocco (100) si considera il *primo* albero alla base dell'hypertree. Dato che 100 è la codifica binaria di 4, sceglieremo da questo albero la chiave privata s_4 e così via per il resto dei blocchi, come mostrato nella Figura 4.8.

La firma a questo punto è costituita dalle chiavi private sk_i^j scelte per ogni blocco del digest, dove in sk_i^j $j \in \{1, \ldots, N\}$ indica uno degli N alberi alla base dell'hypertree e $i \in \{1, \ldots, 8\}$ è l'indice della chiave privata in quell'albero. Quindi la firma, per questo esempio, sarà

$$\sigma(m) = (sk_4^1, auth_4^1, sk_3^2, auth_3^2, sk_3^3, aut_3^3, sk_5^4, auth_5^4, sk_7^5, auth_7^5, sk_1^6, auth_1^6)$$

dove $auth_i^j$ è il percorso di autenticazione per la chiave privata sk_i^j.

Per verificare una firma in FORS, il verificatore dovrà calcolare tutte le radici $(r_0, \ldots, r_5)$ a partire dalle chiavi sk_i e dai percorsi di autenticazione contenuti nella

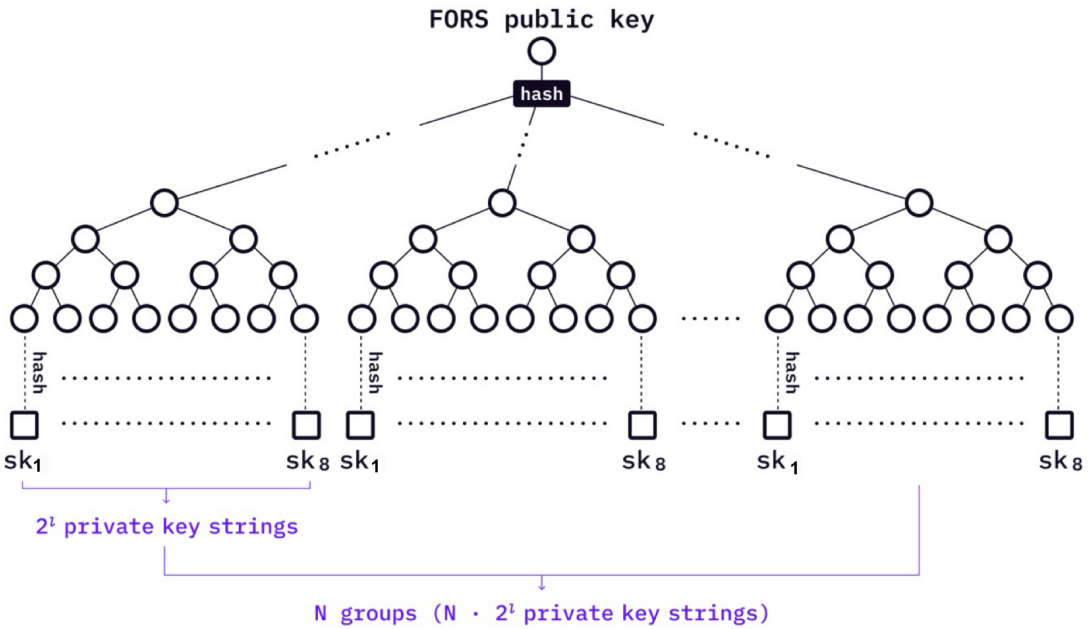


Figura 4.7: Struttura di un hypertree FORS.

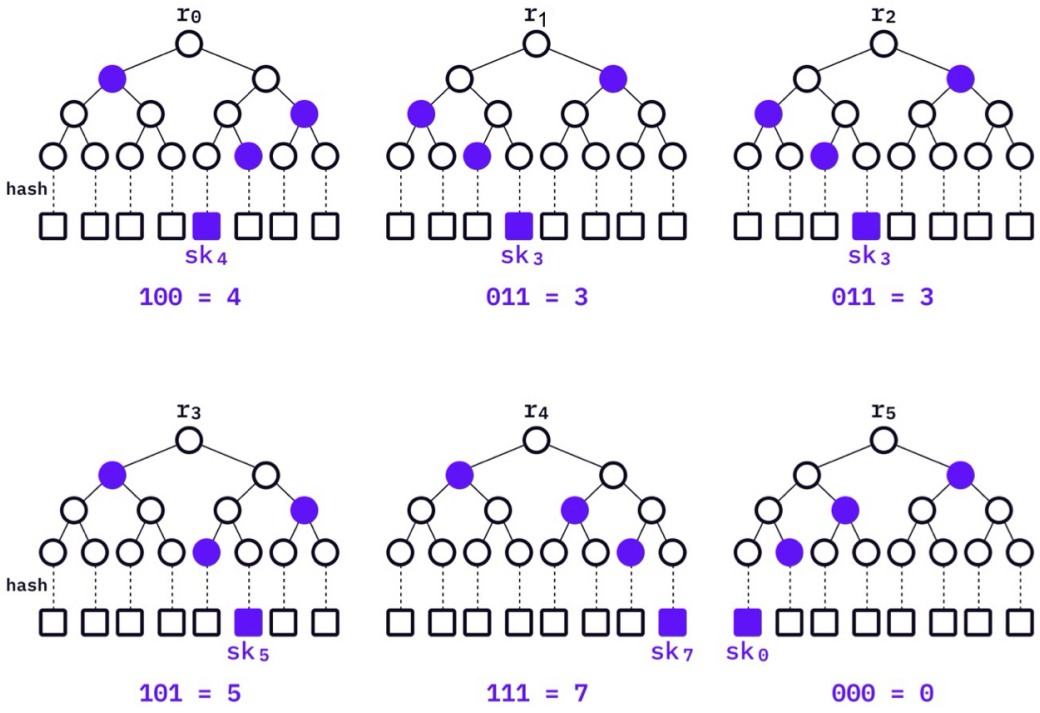


Figura 4.8: Scelta delle chiavi private s_i per ogni blocco del digest. I nodi in viola formano il percorso di autenticazione per la specifica chiave privata sk_i

firma $\sigma(m)$. Poi dovrà calcolare i valori hash delle radici, proprio come in un albero di Merkle. Se il valore hash finale è equivalente alla chiave pubblica di FORS, cioè la radice dell'albero FORS, la firma è valida altrimenti viene rifiutata.

Si noti che adesso ogni blocco ha la sua personale serie di chiavi private, da cui può prelevare il suo blocco di chiave privata a seconda del valore decimale del blocco. Questo significa che se per esempio il terzo e il quinto blocco dovessero codificare entrambi il valore decimale 5, sk_5 del terzo albero sarà diversa da sk_5 del quinto albero, evitando così di avere la stessa sk_i ripetuta più volte nella firma [17].

4.6 SPHINCS+:una Stateless Signature

SPHINCS+ è uno schema di firma "**stateless**". Se uno schema di firma basato su hash di tipo **stateful** è uno schema in cui ogni volta che una chiave è utilizzata, deve essere tracciata in modo che non sia riutilizzata, uno schema di firma **stateless** è uno schema in cui invece non è necessario tracciare le chiavi già utilizzate per firmare un messaggio. Tuttavia, questo non significa che firmare con una chiave privata in uno schema di firma stateless non riveli parte della chiave privata, per questo motivo ci sono ancora limiti su quanti messaggi possono essere firmati con la stessa chiave. Il superamento di questo limite consentirà ad un attaccante di generare firme contraffatte sui messaggi.

Nella Sezione 4.4 abbiamo introdotto con HORS il concetto di **few time signature**, mostrando che una firma di questa tipologia permette di firmare messaggi diversi con la stessa chiave, poiché ogni firma rivela poche informazione sulla chiave privata. Se la quantità di informazione rivelata è mantenuta bassa, un attaccante dovrebbe impiegare troppo lavoro per falsificare una firma.

Questa è anche l'idea alla base degli schemi di firma **stateless**: si imposta un *numero massimo di firme* che si vuole effettuare, diciamo per esempio 2^{64} firme, dopodiché si imposta una *soglia* che indica quante volte una chiave può essere riutilizzata senza che sia rilevata informazione sufficiente che permetta ad un attaccante di falsificare una firma. Con questi due parametri, numero massimo di firme e soglia, è possibile determinare un numero di chiavi che permetta di creare il numero di firme prefissato mentre si rimane al di sotto della soglia accettabile di riutilizzo della chiave.

Come si può immaginare il numero di chiavi sarà molto elevato, quindi verrà utilizzato un hypertree per memorizzare le chiavi e verrà distribuita solo la chiave pubblica radice dell'albero.

Diamo adesso una panoramica sulla struttura di SPHINCS+, rimandando il lettore alla documentazione originale per ulteriori approfondimenti [11].

Ad essere precisi, SPHINCS+ è un framework di firma piuttosto che un singolo schema di firma, perché possiede molti parametri che forniscono flessibilità, permettendo agli utilizzatori di scegliere il compromesso desiderato in termini di **dimensione della firma** (s=small), **velocità di firma** (f=fast), numero di firme da effettuare e livello di sicurezza.

SPHINCS+ è costituito da una combinazione di FORS e un grande albero di Merkle di alberi WOTS+ di dimensione h, infatti può essere visto come un hypertree di dimensione h, suddiviso in h/d sottoalberi WOTS+ di dimensione d, combinato con più FORS. La Figura 4.9 mostra l'intera struttura.

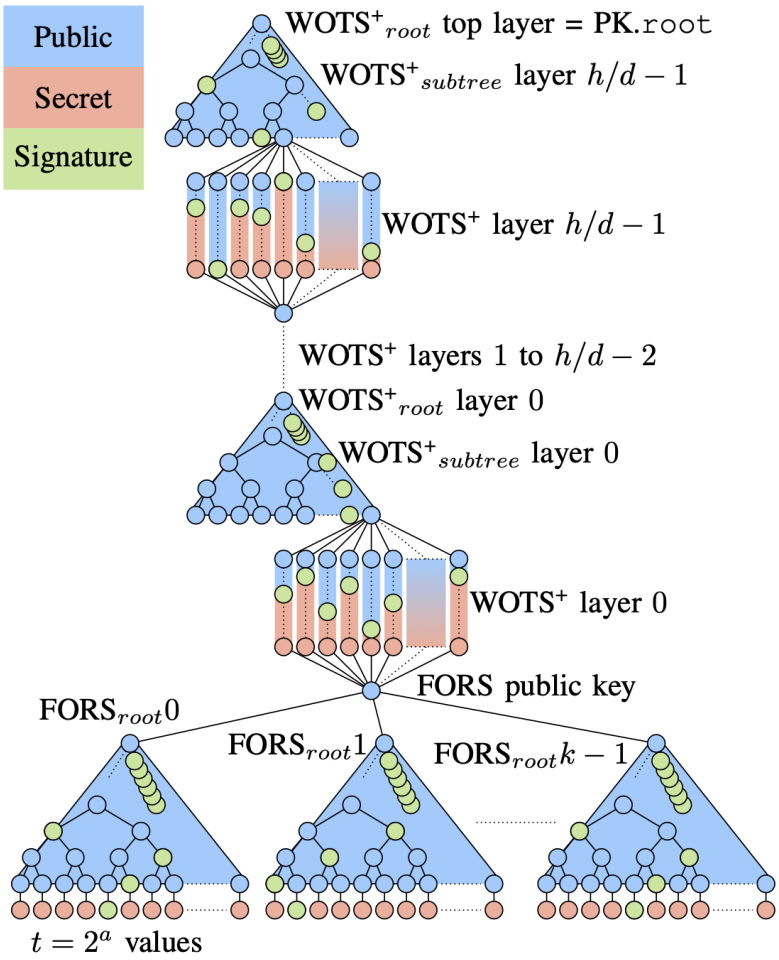


Figura 4.9: Una firma SPHINCS+ è composta da una firma FORS (che firma il messaggio) e da diverse firme WOTS+ e percorsi di autenticazione (che firmano la chiave pubblica FORS) [10]

Una chiave privata SPHINCS+, chiamata SK.seed, è essenzialmente un seme (generato casualmente) che viene utilizzato per derivare tutti i valori privati. Una chiave pubblica SPHINCS+, chiamata PK.root, è il nodo radice del sotto albero WOTS+ più alto.

La procedura di firma inizia firmando il messaggio utilizzando lo schema di firma FORS, sfruttando lo schema di firma FORS alla base dell'hypertree di SPHINCS+. La chiave pubblica radice dell'albero FORS viene firmata diverse volte con WOTS+ fino ad arrivare alla radice dell'hypertree. La dimensione 2^h (dell'ordine di 2^{64}) garantisce che la probabilità di scegliere la stessa chiave privata molte volte sia molto bassa. In combinazione con il fatto che utilizzare la stessa chiave privata in

FORS poche volte non è pericoloso, questo comporta che un firmatario non deve tenere traccia degli indirizzi di partenza precedentemente selezionati. È proprio questo fatto a rendere SPHINCS+ stateless.

Il prezzo per essere stateless comporta però un maggiore sforzo di elaborazione e una dimensione di firma più grande. Una firma FORS completa e gli alberi di Merkle basati su WOTS+ devono essere valutati per firmare un messaggio. La firma richiede quindi (a seconda dei parametri) fino a dieci milioni di chiamate a funzioni hash. Il tempo necessario per calcolare una funzione hash è quindi un criterio importante per la velocità di elaborazione di SPHINCS+. Nella documentazione di SPHINCS+ sono proposte tre diverse funzioni di hash: SHA-256 (SHA-2), SHAKE256 (SHA-3) e Haraka.

Capitolo 5

Valutazione sperimentale

In questo capitolo verranno mostrati i risultati di alcuni test condotti sugli schemi di firma ECDSA, Dilitihum e SPHINCS+ in modo da poterne fare un confronto in termini di prestazioni in tempo, dimensioni delle chiavi, firma e livello di sicurezza. In particolare per ogni algoritmo di firma testato sono stati effettuati 100 test nei quali si richiedeva di firmare sempre la stessa stringa "*Hello Word!*". Per ognuno dei 100 test sono stati raccolti i tempi di generazione delle chiavi, firma e verifica; per poi farne una media. I test sono stati eseguiti su un Mac con processore M2 @ 3.20 GHz e 8 GB di memoria RAM.

Per una miglior comprensione dei confronti che effettueremo ricordiamo, come già discusso nella sezione 3.2, che il NIST definisce i seguenti livelli di sicurezza (elencati in ordine di forza crescente) [39]:

- **Livello 1**: qualsiasi attacco che infrange la definizione di sicurezza pertinente deve richiedere risorse computazionali paragonabili o superiori a quelle richieste per la ricerca di chiavi su un cifrario a blocchi con una **chiave a 128 bit** (ad es. AES128)
- **Livello 2**: qualsiasi attacco che infrange la definizione di sicurezza pertinente deve richiedere risorse computazionali paragonabili o superiori a quelle richieste per la ricerca di collisioni su una **funzione hash a 256 bit** (ad es. SHA256/ SHA3-256)
- **Livello 3**: Qualsiasi attacco che infrange la definizione di sicurezza pertinente deve richiedere risorse computazionali paragonabili o superiori a quelle richieste per la ricerca di chiavi su un cifrario a blocchi con una **chiave a 192 bit** (ad es. AES192)
- **Livello 4**: Qualsiasi attacco che infrange la definizione di sicurezza pertinente deve richiedere risorse computazionali comparabili o superiori a quelle richieste per la ricerca di collisione su una **funzione hash a 384 bit** (ad es. SHA384/ SHA3-384)

- **Livello 5**: Qualsiasi attacco che infrange la definizione di sicurezza pertinente deve richiedere risorse computazionali comparabili o superiori a quelle richieste per la ricerca di chiavi su un cifrario a blocchi con una **chiave a 256 bit** (ad es. AES 256)

5.1 ECDSA

Per eseguire i test sullo schema di firma ECDSA è stata utilizzata la libreria "Pure-Python ECDSA and ECDH" [3] con la quale è fornita una semplice implementazione dell'ECC (Elliptic Curve Cryptography) con supporto per **ECDSA** (Elliptic Curve Digital Signature Algorithm), EdDSA (Edwards-curve Digital Signature Algorithm) e ECDH (Elliptic Curve Diffie-Hellman), implementata esclusivamente in Python e rilasciata con licenza **MIT** (Massachusetts Institute of Technology).

In particolare questa libreria fornisce meccanismi per la generazione di chiavi, firma, verifica della firma e la derivazione di segreti condivisi per cinque curve della "Suite B" $F(p)$ (campo primo) del NIST, con lunghezze di chiave di 192, 224, 256, 384 e 521 bit. Inoltre se "gmpy2" o "gmpy" sono installati, verranno utilizzati all'interno della libreria per un'aritmetica più veloce, nel nostro caso i test sono stati eseguiti senza questo aumento di performance.

Nello specifico nei test effettuati si è scelto di utilizzare le curve secp256k1 (utilizzata anche da Bitcoin), NIST384p e NIST521p con rispettivamente i seguenti livelli di sicurezza: livello 1, livello 2 e livello 3.

Nella Tabella 5.1 sono espresse le lunghezze di chiave privata, pubblica e firma dello schema di firma ECDSA per ognuna delle curve prese in considerazione. Su 100 test per ogni curva utilizzata, sono stati inoltre raccolti i **tempi medi** per la generazione delle chiavi, la generazione della firma sulla stringa "*Hello Word!*" e la verifica.

Curva	Funzione Hash	Livello di sicurezza NIST	Chiave Privata (bytes)	Chiave Pubblica (bytes)	Firma (bytes)	Generazione chiavi (ms)	Gen. Firma (ms)	Verifica (ms)
SEPC256K1	SHA2-256	1	32	32	64	0.50	0.47	1.81
NIST384p	SHA2-256	2	48	48	96	1.23	1.14	4.72
NIST521p	SHA2-256	3	66 [1]	66 [1]	131[2]	2.89	2.64	11.24

Tabella 5.1: Tests su ECDSA

La dimensione in bytes della **chiave pubblica** in tabella è la dimensione compressa perché ricordiamo che la chiave pubblica è il punto $Q = dG$ e in quanto punto avrà le coordinate X e Y, ognuna rappresentabile in tanti bit quanti quelli della chiave privata d. Quindi in totale la chiave pubblica dovrebbe avere 2·(bit della chiave privata). Ma essendo l'equazione della curva pubblica, è possibile comprimere la chiave pubblica con solo la coordinata X, che ha la stessa dimensione in bit della

[1] In realtà sarebbero 521 bit, non 66 bytes·8=528 bits
[2] In realtà sarebbero 1042 bit, non 131 bytes·8=1048 bits

chiave privata, in quanto la coordinata Y può essere ricavata dalla equazione della curva.

5.2 Schemi di firma post quantistici: Dilitium e SPHINCS+

Per eseguire i test sugli schemi di firma Dilitium e SPHINCS+ è stata utilizzata la libreria "liboqs-python" [2] la quale offre un wrapper scritto in Python 3 per la libreria "liboqs" [1] [20] di Open Quantum Safe, che è una libreria in C per algoritmi crittografici resistenti alla computazione quantistica. In particolare il progetto Open Quantum Safe (OQS) ha l'obbiettivo di sviluppare e integrare all'interno delle applicazioni la crittografia post quantistica, per facilitarne la distribuzione e testarla in contesti di uso reale. Tra i vari algoritmi di crittografia post quantistica supportati da questa libreria, per effettuare i test su Dilithium e SPHINCS+ sono stati utilizzati i seguenti algoritmi di firma:

- **CRYSTALS-Dilithium**: Dilithium2, Dilithium3, Dilithium5:
- **SPHINCS+-SHA2**: SPHINCS+-SHA2-128f-simple, SPHINCS+-SHA2-128s-simple, SPHINCS+-SHA2-192f-simple, SPHINCS+-SHA2-192s-simple, SPHINCS+-SHA2-256f-simple, SPHINCS+-SHA2-256s-simple
- **SPHINCS+-SHAKE**: SPHINCS+-SHAKE-128f-simple, SPHINCS+-SHAKE-128s-simple, SPHINCS+-SHAKE-192f-simple, SPHINCS+-SHAKE-192s-simple, SPHINCS+-SHAKE-256f-simple, SPHINCS+-SHAKE-256s-simple

Il numero intero in fondo al nome di Dilithium indica il livello di sicurezza stabilito dal NIST soddisfatto da quella specifica implementazione di Dilithium. Invece per quanto riguarda SPHINCS+ sono state testate sia le versioni che utilizzano le funzioni hash SHA2 da 128, 192 e 256 bit e la versioni che utilizzano la funzione hash SHAKE da 128, 192 e 256 bit. I suffissi "f" e "s" negli algoritmi di firma SPHINCS+ indicano i parametri, come già accennato nella Sezione 4.6, che parametrizzano lo schema di firma SPHINCS+. In particolare "f" sta per "fast" ovvero si vuole che il processo di generazione della firma sia veloce; il suffisso "s" invece sta per "small" cioè indica la volontà di generare firme brevi. Come per ECDSA sono stati eseguiti 100 test per ogni algoritmo di firma di Dilithium e SPHINCS+ con i loro peculiari parametri, ottenendo i tempi medi di generazione delle chiavi, firma e verifica riassunti nella Tabella 5.2 per Dilithium e nella Tabella 5.3 per SPHINCS+.

Dai risultati dei test su SPHINCS+ si puo notare che, considerando lo stesso livello di sicurezza, una firma "fast" ha tempi di generazione delle chiavi e di firma nettamente minori rispetto a una firma "small"; generando però una firma di dimensioni maggiori rispetto ad una "small", che quindi richiederà un maggior tempo di verifica.

Algoritmo di firma	Funzione Hash	Livello di sicurezza NIST	Chiave Privata (bytes)	Chiave Pubblica (bytes)	Firma (bytes)	Gen. chiavi (ms)	Gen. Firma (ms)	Verifica (ms)
Dhilitium-2	SHAKE-128 e SHAKE-256	2	2528	1312	2420	0.07	0.11	0.04
Dhilitium-3	SHAKE-128 e SHAKE-256	3	4000	1952	3293	0.11	0.16	0.05
Dhilitium-5	SHAKE-128 e SHAKE-256	5	4864	2592	4595	0.12	0.21	0.09

Tabella 5.2: Tests su Dilithium

Algoritmo di firma	Funzione Hash	Livello di sicurezza NIST	Chiave Privata (bytes)	Chiave Pubblica (bytes)	Firma (bytes)	Gen. chiavi (ms)	Gen. Firma (ms)	Verifica (ms)
SPHINCS+-SHA2-128f-simple	SHA2-128	1	64	32	17088	1,25	28.47	1.71
SPHINCS+-SHA2-128s-simple	SHA2-128	1	64	32	7856	76.92	580.35	0.59
SPHINCS+-SHA2-192f-simple	SHA2-192	3	96	48	35664	1.80	48.67	2.61
SPHINCS+-SHA2-192s-simple	SHA2-192	3	96	48	16224	114.19	1114.03	0.98
SPHINCS+-SHA2-256f-simple	SHA2-256	5	128	64	49856	4.73	100.82	2.71
SPHINCS+-SHA2-256s-simple	SHA2-256	5	128	64	29792	76.18	1022.73	1.42
SPHINCS+-SHAKE-128f-simple	SHAKE-128	1	64	32	17088	2.25	51.99	3.11
SPHINCS+-SHAKE-128s-simple	SHAKE-128	1	64	32	7856	143.68	1095.02	1.11
SPHINCS+-SHAKE-192f-simple	SHAKE-192	3	96	48	35664	3.49	89.67	4.83
SPHINCS+-SHAKE-192s-simple	SHAKE-192	3	96	48	16224	215.17	1930.03	1.62
SPHINCS+-SHAKE-256f-simple	SHAKE-256	5	128	64	49856	9.03	180.65	4.81
SPHINCS+-SHAKE-256s-simple	SHAKE-256	5	128	64	29792	142.95	1703.55	2.38

Tabella 5.3: Tests su SPHINCS+

5.3 Confronto

Per una maggiore comprensione del confronto che effettueremo, si consiglia di consultare la Tabella 5.4 nella quale sono stati raccolti i risultati dei test su ECDSA, Dilithium e SPHINCS+. Le considerazioni che effettueremo saranno fatte tra gli algoritmi di firma a pari livello di sicurezza. Se in termini di prestazioni dal punto di vista dei tempi Dilithium è il migliore tra i tre, per quanto riguarda la dimensione delle chiavi, ponendo attenzione sulla chiave pubblica che sarà quella che dovrà essere trasmessa, Dilithium ha chiavi di dimensioni nettamente maggiori rispetto a ECDSA e SPHINCS+, nonostante sia lo schema di firma basato su reticoli che ha le chiavi e la firma di dimensione minori.

Per quanto riguarda la dimensione della firma, possiamo osservare che Dilithium genera firme più piccole di circa un ordine di grandezza rispetto a quelle di

Algoritmo	Funzione Hash	Livello di sicurezza NIST	Chiave Privata (bytes)	Chiave Pubblica (bytes)	Firma (bytes)	Gen. chiavi (ms)	Gen. Firma (ms)	Verifica (ms)
ECDSA con SEPC256K1	SHA2-256	1	32	32	64	0.50	0.47	1.81
ECDSA con NIST384p	SHA2-256	2	48	48	96	1.23	1.14	4.72
ECDSA con NIST521p	SHA2-256	3	66	66	131	2.89	2.64	11.24
Dhilitium-2	SHAKE-128 e SHAKE-256	2	2528	1312	2420	0.07	0.11	0.04
Dhilitium-3	SHAKE-128 e SHAKE-256	3	4000	1952	3293	0.11	0.16	0.05
Dhilitium-5	SHAKE-128 e SHAKE-256	5	4864	2592	4595	0.12	0.21	0.09
SPHINCS+-SHA2-128f-simple	SHA2-128	1	64	32	17088	1,25	28.47	1.71
SPHINCS+-SHA2-128s-simple	SHA2-128	1	64	32	7856	76.92	580.35	0.59
SPHINCS+-SHA2-192f-simple	SHA2-192	3	96	48	35664	1.80	48.67	2.61
SPHINCS+-SHA2-192s-simple	SHA2-192	3	96	48	16224	114.19	1114.03	0.98
SPHINCS+-SHA2-256f-simple	SHA2-256	5	128	64	49856	4.73	100.82	2.71
SPHINCS+-SHA2-256s-simple	SHA2-256	5	128	64	29792	76.18	1022.73	1.42
SPHINCS+-SHAKE-128f-simple	SHAKE-128	1	64	32	17088	2.25	51.99	3.11
SPHINCS+-SHAKE-128s-simple	SHAKE-128	1	64	32	7856	143.68	1095.02	1.11
SPHINCS+-SHAKE-192f-simple	SHAKE-192	3	96	48	35664	3.49	89.67	4.83
SPHINCS+-SHAKE-192s-simple	SHAKE-192	3	96	48	16224	215.17	1930.03	1.62
SPHINCS+-SHAKE-256f-simple	SHAKE-256	5	128	64	49856	9.03	180.65	4.81
SPHINCS+-SHAKE-256s-simple	SHAKE-256	5	128	64	29792	142.95	1703.55	2.38

Tabella 5.4: Confronto tra ECDSA Dilithium e SPHINCS+

SPHINCS+. SPHINCS+ presenta invece come unico vantaggio rispetto a Dilithium quello di generare chiavi di dimensioni minori, paragonabili a quella di ECDSA; questo è però l'unico vantaggio perché SPHINCS+ ha tempi di generazione delle chiavi e firma nettamente maggiori, da 1 a 4 ordini di grandezza maggiori, quindi è molto più lento.

Dilithium viene dunque preferito come schema di firma post quantistico per la sua efficienza in tempo, ma presenta il problema di chiave pubblica e firma molto grandi. Si noti che in uno schema di firma i dati da trasmettere sono proprio la firma e la chiave pubblica per verificarla. La dimensione totale di chiave pubblica e firma di Dilithium rimane comunque minore rispetto alla dimensione totale di chiave pubblica e firma di SPHINCS+, penalizzato da dimensioni di firma molto grandi.

In conclusione, Dilithium ha tempi più brevi ma alla fine ha anche quantità di dati totali da trasmettere inferiore rispetto a SPHINCS+. Come già osservato però

Dilithium ha chiave pubblica e firma di dimensioni nettamente maggiori rispetto a ECDSA. Anche SPHINCS+ ha una firma molto grande, e queste dimensioni risultano essere maggiori della *Maximum Trasmission Unit* (MTU), tipicamente fissata a 1500 byte, che rappresenta la massima capacità di un pacchetto IP. Quindi in una futura transazione a schemi di firma post quantistici si dovrà tenere conto di trovare meccanismi per mitigare al meglio problemi di latenza e frammentazione di pacchetti causati da queste grandi dimensioni di firma e chiave pubblica da dover trasmettere, aspetto che non riguarda invece gli attuali schemi di firma come ECDSA dove chiavi e firme sono di dimensioni più contenute.

Conclusioni

Ormai siamo sicuri del fatto che la costruzione di un computer quantistico su larga scala comprometterebbe la sicurezza di tutta la crittografia a chiave pubblica, e abbiamo capito che non è opportuno aspettare fin quando i computer quantistici saranno disponibili per modificare i sistemi di sicurezza delle informazioni basati sulla crittografia pubblica; ma è necessario apportare modifiche adesso, proprio per evitare la tipologia di attacchi "raccogli ora, decifra dopo".

Il NIST ha un ruolo molto importante nello sviluppo della crittografia post quantistica, infatti, grazie al suo processo di standardizzazione sono già stati scelti nuovi schemi nell'ambito della crittografia a chiave pubblica che sono resistenti alla computazione quantistica e che possono essere usati per creare standard per il Key-Encapsulation Mechanism e le firme digitali.

A proposito di firme digitali, quella che tra i vari vincitori è la più promettente, è lo schema di firma di Dilithium che basa la sua sicurezza sull'intrattabilità dei problemi su reticoli. Oltre a Dilithium, anche l'altro vincitore per gli schemi di firma FALCON si basa sui reticoli. Essendo il campo della matematica sui reticoli un campo in via di sviluppo, e all'interno del quale vengono fatte sempre nuove scoperte, ad oggi possiamo dire che non si conoscono algoritmi quantistici efficienti per risolvere i problemi sui reticoli come SVP, LWE e SIS su cui basano la loro sicurezza questi schemi di firma. Ma ciò non implica che un algoritmo del genere non venga scoperto nel futuro, quindi il NIST ha scelto come terzo vincitore per gli schemi di firma SPHINCS+, vedendolo come schema di firma di backup, proprio perché la sua sicurezza si basa sulle funzioni hash crittografiche, che risultano resistenti alla computazione quantistica.

Altra conclusione evidente è che i nuovi schemi di firma post quantistici sono sicuri rispetto alla computazione quantistica, ma hanno dimensioni di chiavi e firma superiori a quelle degli schemi classici di firma, non resistenti però alla computazione quantistica. Inoltre SPHINCS+ è molto più lento anche nelle fasi di generazione delle chiavi, firma e verifica, quindi questo porta a porre attenzione nel trovare meccanismi che riducano i tempi di questi nuovi protocolli, sia dal punto di vista di elaborazione che di trasmissione dei dati, in una futura, e sempre più imminente, transazione agli schemi di firma post quantistici.

Bibliografia

[1] liboqs. https://github.com/open-quantum-safe/liboqs/blob/main/README.md, Pubblicazione: 17 Gennaio 2024. Versione 0.9.2.

[2] liboqs-python: Python 3 bindings for liboqs. https://github.com/open-quantum-safe/liboqs-python/tree/main, Pubblicazione: 30 Ottobre 2023. Versione 0.9.0.

[3] Pure-Python ECDSA and ECDH. https://pypi.org/project/ecdsa/, Pubblicazione: 9 Luglio 2022. Versione 0.18.0.

[4] Leonard M. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 55–60, 1979.

[5] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). *Electron. Colloquium Comput. Complex.*, TR96, 1996.

[6] Miklós Ajtai. The shortest vector problem in L2 is NP-hard for randomized reductions (extended abstract). In *Symposium on the Theory of Computing*, 1998.

[7] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi Kai Liu. *Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process*. Gennaio 2019.

[8] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi Kai Liu. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. Giugno 2020.

[9] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai, Liu Carl, Miller Dustin, Moody Rene, Peralta Ray, Perlner Angela, Robinson Daniel, Smith-Tone, and Yi-Kai Liu. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. Giugno 2022.

[10] Dorian Amiet, Lukas Leuenberger, Andreas Curiger, and Paul Zbinden. FPGA-based SPHINCS+ Implementations: Mind the Glitch. *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 229–237, 2020.

[11] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Maria Eichlseder Christoph Dobraunig, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. *SPHINCS + Submission to the NIST post-quantum project, v.3.1.* 10 Giugno 2022.

[12] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation (Version 3.1). 8 Febbraio 2021.

[13] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on Learning with Errors. Cryptology ePrint Archive, Paper 2013/838, 2013. https://eprint.iacr.org/2013/838.

[14] Elaine B. Barker. Recommendation for Key Management - Part 1 General. 2014.

[15] A. Bernasconi, P. Ferragina, and F. Luccio. *Elementi di crittografia.* Pisa University Press, 2015.

[16] Ward Beullens. Breaking Rainbow Takes a Weekend on a Laptop. *IACR Cryptol. ePrint Arch.*, 2022:214, 2022.

[17] Dora Research Blog. How do hash-based post-quantum digital signatures work? (Part 2). https://research.dorahacks.io/2022/12/16/hash-based-post-quantum-signatures-2/, 16 dicembre 2022.

[18] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully Homomorphic Encryption without Bootstrapping. Cryptology ePrint Archive, Paper 2011/277, 2011. https://eprint.iacr.org/2011/277.

[19] Lidong Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on Post-Quantum Cryptography. 2016-04-28.

[20] Michele Mosca Douglas Stebila. Post-quantum key exchange for the Internet and the Open Quantum Safe project. https://openquantumsafe.org. In Roberto Avanzi, Howard Heys, editors, Selected Areas in Cryptography (SAC) 2016, LNCS, vol. 10532, pp. 1–24. Springer, October 2017. https://openquantumsafe.org.

[21] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018:238–268, 2018.

[22] Veronica Cristiano e Marco Rinaudo. La matematica dietro la PQC: i reticoli. https://www.telsy.com/la-matematica-dietro-la-pqc-i-reticoli/, 13 Febbraio 2023.

[23] Gianluca Salvalaggio (Relatore: Prof. Riccardo Focardi). Crittografia basata su Curve Ellittiche, Anno Accademico 2004-2005. Tesi di Laurea, Università Ca' Foscari – Venezia.

[24] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. *RFC*, 8391:1–74, 2018.

[25] Andreas Hülsing. WOTS+ – Shorter Signatures for Hash-Based Signature Schemes. Cryptology ePrint Archive, Paper 2017/965, 2017. https://eprint.iacr.org/2017/965.

[26] IBM. Practical introduction to quantum-safe cryptography. https://learning.quantum.ibm.com/course/practical-introduction-to-quantum-safe-cryptography, 2023.

[27] Leslie Lamport. Constructing Digital Signatures from a One Way Function. Technical Report CSL-98, October 1979. This paper was published by IEEE in the Proceedings of HICSS-43 in January, 2010.

[28] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75:565 – 599, 2014.

[29] Harashta Tatimma Larasati and Howon Kim. Quantum Cryptanalysis Landscape of Shor's Algorithm for Elliptic Curve Discrete Logarithm Problem. In Hyoungshick Kim, editor, *Information Security Applications*, pages 91–104, Cham, 2021. Springer International Publishing.

[30] T. Leighton and S. Micali. Large provably fast and secure digital signature schemes from secure hash functions. *U.S. Patent 5,432,852*, 1995.

[31] Arjen K. Lenstra, Hendrik W. Lenstra, and László Miklós Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[32] Vadim Lyubashevsky. Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2009.

[33] Vadim Lyubashevsky and Daniele Micciancio. Generalized Compact Knapsacks Are Collision Resistant. In *International Colloquium on Automata, Languages and Programming*, 2006.

[34] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. *J. ACM*, 60:43:1–43:35, 2010.

[35] Ralph C. Merkle. Secrecy, authentication, and public key systems. 1979.

[36] Ralph C. Merkle. A Certified Digital Signature. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 218–238, New York, NY, 1990. Springer New York.

[37] David A. Cooper Daniel C. Apon Quynh H. Dang Michael S. Davidson Morris J. Dworkin Carl A. Miller. Recommendation for Stateful Hash-Based Signature Schemes. *NIST Special Publication 800-208*, Ottobre 2020.

[38] NIST. NIST Announces First Four Quantum-Resistant Cryptographic Algorithms. https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms, 2022.

[39] NIST. Security (Evaluation Criteria). https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-(evaluation-criteria), Pubblicazione: 3 Gennaio 2017, aggiornato: 26 Febbraio 2024.

[40] Information Technology Laboratory National Institute of Standards and MD 20899-8900 T echnology Gaithersburg. Module-Lattice-Based Digital Signature Standard. 24 Agosto 2023.

[41] Information Technology Laboratory National Institute of Standards and MD 20899-8900 T echnology Gaithersburg. Module-Lattice-based Key-Encapsulation Mechanism Standard. 24 Agosto 2023.

[42] Information Technology Laboratory National Institute of Standards and MD 20899-8900 T echnology Gaithersburg. Stateless Hash-Based Digital Signature Standard. 24 Agosto 2023.

[43] Chris Peikert. A Decade of Lattice Cryptography. *Found. Trends Theor. Comput. Sci.*, 10(4):283–424, 2016.

[44] Chris Peikert and Alon Rosen. Efficient Collision-Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices. *Electron. Colloquium Comput. Complex.*, TR05, 2006.

[45] Ethan Rahn. SPHINCS+ - Step by Step. https://er4hn.info/blog/2023.12.16-sphincs_plus-step-by-step/, 16 Dicembre 2023.

[46] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Symposium on the Theory of Computing*, 2005.

[47] Leonid Reyzin and Natan Reyzin. Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In Lynn Batten and Jennifer Seberry, editors, *Information Security and Privacy*, pages 144–153, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[48] Wikipedia. Elliptic-curve cryptography. https://en.wikipedia.org/wiki/Elliptic-curve_cryptography. [Online; accessed marzo 2024].

[49] Wikipedia. Elliptic Curve Digital Signature Algorithm. https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm. [Online; accessed marzo 2024].

[50] Wikipedia. NP (complexity). https://en.wikipedia.org/wiki/NP_(complexity). [Online; accessed marzo 2024].

[51] Wikipedia. Short integer solution problem. https://en.wikipedia.org/wiki/Short_integer_solution_problem. [Online; accessed marzo 2024].

[52] W.Stalling. *Crittografia*. Pearson, 2022.