UNIVERSITA' POLITECNICA DELLE MARCHE
FACOLTA' DI INGEGNERIA

Corso di Laurea magistrale **Ingegneria Elettronica**

Analisi di chiavi deboli nei sistemi crittografici post-quantum basati su codici LDPC e MDPC

Analysis of weak keys in post-quantum cryptosystems based on LDPC and MDPC codes

Tesi di Laurea di:
Gianmarco Mascilongo

Relatore:
Prof. Marco Baldi

Correlatori:
Prof. Franco Chiaraluce

Dr. Paolo Santini

A.A. **2019/2020**

Dedicato ai miei nonni

Indice

1 INTRODUZIONE

Nei prossimi anni verranno sviluppati i computer quantistici, i quali sono sensibilmente più performanti dei dispositivi attuali. Si sa che questi sistemi saranno in grado di rompere sistemi crittografici attualmente in uso. Per questo motivo il NIST (*National Institute of Standards and Technology*) ha indetto una gara per valutare quali crittosistemi siano in grado di resistere ad attacchi da parte di computer quantistici, per scegliere infine quale sia quello più adatto a diventare il nuovo standard.
Alcune proposte sono basate sull'uso di codici sparsi, come i codici LDPC o MDPC (Low/Moderate Density Parity Check). Questa tipologia di codici è stata largamente studiata nell'ambito delle comunicazioni, date le ottime caratteristiche ed in particolare per l'efficienza nel correggere errori. Di recente sta assumendo particolare interesse la possibilità di impiegarli anche in ambito crittografico. Basandosi su crittosistemi quali quelli di McEliece e Niederreiter, alcune proposte sono state avanzate per utilizzare i codici sparsi per garantire sicurezza *post-quantum*. Alcuni esempi di interesse in questo ambito sono LEDAcrypt e BIKE.
Una delle principali criticità da valutare affinché si possano usare codici sparsi in sicurezza è il valore della Decoding Failure Rate (DFR). Esiste infatti una probabilità non nulla che il decoder in ricezione non riesca a recuperare il messaggio originario quando questi codici vengono impiegati. Il problema è stato approfonditamente studiato, e gli attuali sistemi presentano una DFR estremamente bassa per scopi comunicazionistici. Tuttavia se lo scopo è quello di garantire sicurezza post-quantum, le specifiche riguardo la DFR diventano ancora più stringenti.
Il problema studiato in questo lavoro è stato di trovare un approccio che rendesse possibile definire un metodo per calcolare un limite inferiore per la DFR dei sistemi crittografici, nonostante si tratti di valori talmente piccoli da richiedere un numero di simulazioni eccessivamente grande per essere svolto in tempi ragionevoli, anche avendo a disposizione elaboratori molto performanti.
Si descriverà quindi una metodologia innovativa che permette di avere una stima della DFR ed una prima valutazione della sicurezza. Questa stima è in alcuni casi sufficiente a minare i presupposti per l'affidabilità del crittosistema.
Per fare ciò sarà necessario studiare l'impatto delle chiavi dette "deboli", perché presentano particolari caratteristiche per le quali ci si aspetta che la DFR aumenti (anche notevolmente) quando queste vengono scelte come chiavi.
Si applicherà questa idea ai due algoritmi già citati, LEDAcrypt e BIKE, per due particolari tipi di chiavi deboli, in modo da valutarne la sicurezza e fornire degli esempi di come la metodologia qui presentata possa essere applicata.

1.1 Crittografia Post-Quantica e Standardizzazione NIST

Negli ultimi anni c'è stata una notevole quantità di ricerche sui computer quantistici - macchine che sfruttano fenomeni di meccanica quantistica per risolvere problemi matematici difficili o intrattabili per i computer convenzionali. Se mai verranno costruiti computer quantistici su larga scala, essi saranno in grado di rompere molti dei sistemi di crittografia a chiave pubblica attualmente in uso. Ciò comprometterebbe seriamente la riservatezza e l'integrità delle comunicazioni digitali su Internet e sugli altri mezzi di comunicazione. L'obiettivo della crittografia "post-quantum" è quello di sviluppare sistemi crittografici che siano sicuri sia contro i computer quantistici che contro quelli classici, e che possano operare anche con i protocolli e le reti di comunicazione esistenti.

E' difficile prevedere quando sarà costruito un computer quantistico su larga scala. Mentre in passato era meno chiaro che i computer quantistici di grandi dimensioni sono una possibilità fisica, molti scienziati ora credono che sia solo una sfida di tipo meramente ingegneristico. Alcuni ingegneri prevedono addirittura che entro i prossimi venti anni circa saranno costruiti computer quantistici in grado di rompere essenzialmente tutti gli schemi a chiave pubblica attualmente in uso. Storicamente, ci sono voluti quasi due decenni per implementare la nostra moderna infrastruttura di crittografia a chiave pubblica. Pertanto, indipendentemente dal fatto che si possa stimare il momento esatto dell'arrivo dell'era del calcolo quantistico, è necessario iniziare ora a preparare i sistemi di sicurezza informatica per poter resistere al calcolo quantistico [20].

1.2 Sintesi della trattazione

La prima parte di questo elaborato è di tipo teorico, e senza pretesa di completezza tratterà gli argomenti fondamentali per la comprensione di quanto verrà esposto di seguito.

Per questo motivo nel capitolo 2 si fornirà una panoramica della teoria dei codici, perché è su questi che si basano i crittosistemi che saranno analizzati. Quindi si esaminerà più in dettaglio il funzionamento e le proprietà dei crittosistemi che saranno oggetto di studio, cioè LEDAcrypt e BIKE, quest'ultimo in due diverse versioni.

Nel capitolo 4 si approfondirà un particolare elemento di BIKE, cioè la scelta delle soglie di decodifica, che è opportuno studiare in dettaglio in quanto tramite lo studio di questo argomento si è giunti ad alcuni dei risultati più interessanti qui presentati.
Il capitolo 5 può essere considerato la parte centrale e fondamentale dell'intera trattazione. Qui verrà esposto in maniera approfonditamente il problema che si sta analizzando, comparando lo stato dell'arte con la metodologia originale qui presentata per l'analisi della DFR (e quindi della sicurezza). Inoltre saranno esposti anche i dettagli tecnici circa le elaborazioni che hanno portato ai risultati esposti successivamente.

La seconda parte del discorso riguarda la parte sperimentale, cioè lo studio di due particolari tipi di chiavi deboli usando l'approccio presentato in questo lavoro.

I capitoli 6 e 7 presenteranno due diversi casi di studio, ognuno dei quali sarà riferito a più crittosistemi; il primo sarà testato sia per LEDAcrypt che per BIKE, mentre il secondo sarà riferito al solo BIKE ma per due differenti versioni.
Si fornirà la spiegazione delle premesse teoriche e verranno mostrati e commentati i risultati. Tramite lo studio degli effetti di questi due tipi di chiavi deboli sarà possibile analizzare le proprietà di sicurezza dei sistemi e fornire un esempio di come la metodologia qui proposta possa essere impiegata concretamente.

Una volta tratte le conclusioni per ogni caso di studio, si forniranno le conclusioni finali dell'intero lavoro, mettendone in mostra anche le criticità ed i margini per sviluppi futuri.

In appendice si procederà a descrivere il codice - in linguaggio C - tramite cui i crittosistemi sono implementati; ciò è utile sia per motivi espositivi e didattici, sia perché fornisce al lettore interessato la possibilità di replicare o approfondire i test.

2 FONDAMENTI DI TEORIA DEI CODICI

Di seguito viene fornita una breve teoria dei codici per la correzione d'errore, in particolare riguardo i codici LDPC, i canali di trasmissione e le tecniche di codifica e decodifica.

2.1 Codici a correzione d'errore

I codici a correzione d'errore (*ECC: Error Correcting Code*) vengono utilizzati per rilevare e correggere gli errori nei dati trasmessi su canali di comunicazione rumorosi. L'idea centrale è che il mittente codifichi il messaggio con informazioni ridondanti, consentendo così al destinatario di rilevare un numero limitato di errori che possono verificarsi in qualsiasi punto del messaggio e spesso di correggere tali errori senza che si debba ritrasmettere il messaggio.

Definizione 1: **Codice binario lineare**

Un codice binario lineare C, definito dalla terna [n, k, d], è un sottospazio k-dimensionale di uno spazio vettoriale n-dimensionale, che viene utilizzato per fornire la struttura di un vettore di messaggi per trasmissione su un canale.

Per trasmettere messaggi su canali di comunicazione utilizzando un codice a correzione di errore, codifichiamo il messaggio utilizzando una matrice generatrice.

Definizione 2: **Matrice G**

La matrice **G** di un codice C, è una matrice che ha dimensioni k × n. Le k righe di **G** corrispondono a parole di codice linearmente indipendenti che formano una base di C.

Uno degli aspetti più importanti della correzione degli errori è il processo di decodifica. La matrice di controllo di parità permette di identificare se sono stati introdotti errori durante la trasmissione.

Definizione 3: Matrice di parità

Una matrice di parità **H** di un codice C è una matrice che genera il kernel del codice. Ciò significa che una parola di codice **c** è nel codice C se e solo se $\mathbf{H} \cdot \mathbf{c}^T = \mathbf{0}$, dove **0** è il vettore nullo di dimensioni r × 1. **H** ha dimensioni (n - k) × n.

Una particolare famiglia di codici di interesse per la trattazione seguente è quella dei codici Quasi-Ciclici. Sarà utile definire il concetto di matrice circolante.

Definizione 4: Matrice circolante

Una matrice circolante è una matrice quadrata in cui ogni vettore riga è ottenuto shiftando a destra il vettore riga precedente.

Un esempio di grafico di matrice circolante di ordine 64 è riportato in Figura 1.

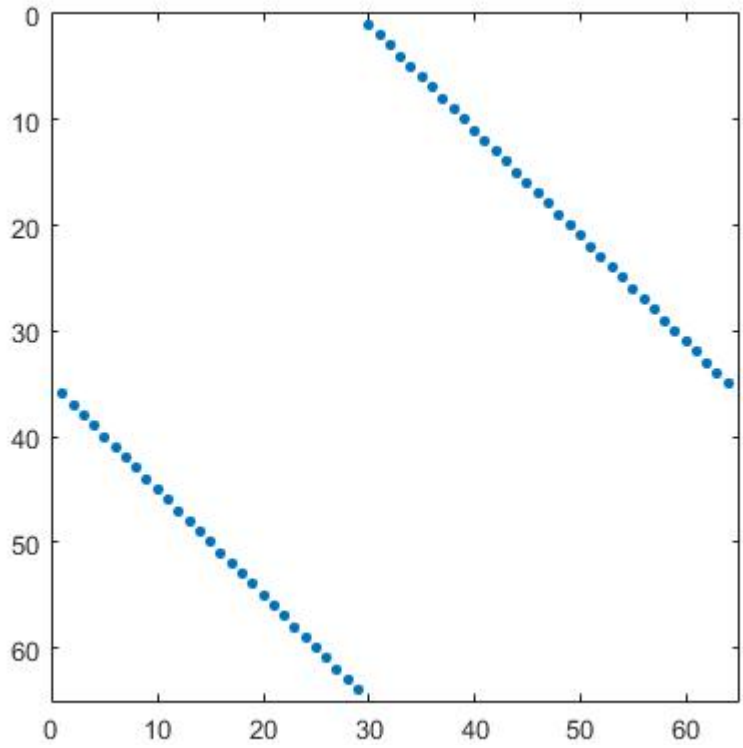

Figura 1: Esempio di matrice circolante. I punti blu rappresentano gli elementi 1, gli spazi bianchi gli 0.

Una matrice circolante a blocchi è data da un insieme di matrici circolanti concatenate.

Adesso è possibile comprendere cosa sia un codice Quasi-Ciclico.

Definizione 5: Codici Quasi-Ciclici

Un codice (binario) quasi ciclico (*QC*) di indice n_0 e di ordine r è un codice lineare che ha come matrice **G** una matrice circolante a blocchi di ordine r e di indice n_0 . Un codice QC (n_0, k_0) è un codice Quasi-Ciclico di indice n_0 , lunghezza $n_0 \cdot r$ e dimensione $k_0 \cdot r$.

Un'altra classe di codici di fondamentale importanza è quella dei codici LDPC.

Definizione 6: Codici LDPC

Se una matrice di parità di un codice è sparsa, allora il codice corrispondente C è chiamato codice LDPC (*Low Density Parity Check*).

Nel contesto dei codici LDPC viene definita sparsa una matrice in cui ci sono meno elementi 1 che elementi 0. Data la natura sparsa dei codici LDPC i processi di decodifica hanno tempi di esecuzione rapidi, in quanto ci sono meno operazioni da elaborare rispetto a matrice di parità non sparse.

I codici binari MDPC (Moderate-Density Parity-Check) sono invece codici binari lineari che che hanno matrice di parità piuttosto sparsa ma più densa rispetto ai codici LDPC, con pesi di riga tipicamente dell'ordine di $O(\sqrt{n})$. Tale matrice permette l'uso di decodificatori iterativi simili a quelli utilizzati per i codici LDPC, ampiamente impiegati per la correzione di errori nel settore delle telecomunicazioni.

Definizione 7: Codici QC-LDPC

Un codice QC-LDPC (n_0, k_0, r, w) è un codice (n_0, k_0) quasi-ciclico di lunghezza $n = n_0 \cdot r$, dimensione $k = k_0 \cdot r$, ordine r (e quindi indice n_0), caratterizzato da una matrice di parità sparsa.

Similmente si possono definire i codici QC-MDPC, per cui vale una definizione analoga quando la matrice di parità ha righe di peso costante pari a $w = O(\sqrt{n})$.

Un modo comunemente utilizzato per visualizzare i codici e studiarne le proprietà è quello di riferirsi al grafo di Tanner.

Definizione 8: Grafo di Tanner

Il grafo di Tanner è una rappresentazione grafica della matrice di parità **H.** E' un grafo bipartito con due insiemi d nodi:

- Nodi variabile, in numero pari alla lunghezza del codice.
- Nodi controllo, in numero pari alla ridondanza del codice.

Il grado di un nodo è definito come il numero di rami ad esso collegati.

Il grado dei nodi variabile e ai nodi controllo corrisponde rispettivamente al peso delle colonne e delle righe di **H**.

Viene riportato un esempio di visualizzazione del grafo di Tanner per un codice con n = 7 ed r = 3.

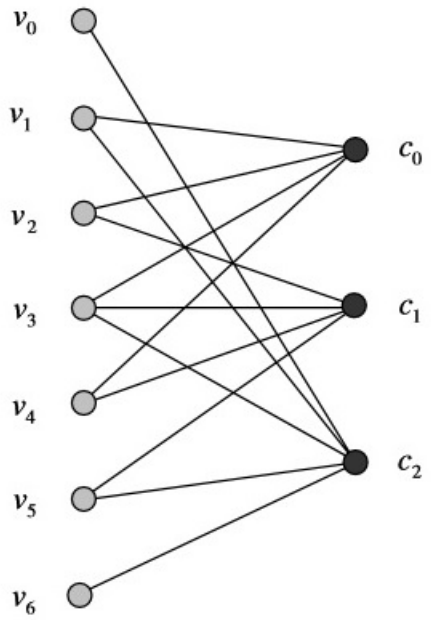

Figura 2: Esempio di grafo di Tanner.

2.2 Codifica e verifica di parità

Il processo di trasformazione di un vettore che rappresenta un messaggio in una parola di codice associata è noto come codifica.

Ogni parola in codice $\mathbf{c} = [c_0 , c_1 , ... , c_{n-1}] \in C$ può essere espresso come $\mathbf{c} = \mathbf{m} \cdot \mathbf{G}$, dove $\mathbf{m} = [m_0 , m_1 , ..., m_{k-1}]$ è il vettore che rappresenta il messaggio.

La parola in codice **c** ha i k bit informativi originali ed r bit di parità supplementari, in modo che la parola di codice abbia una lunghezza risultante di n bit.

Dato che la matrice di parità **H** rappresenta lo spazio nullo del codice, possiamo usarla come metodo di verifica per controllare se un dato vettore ricevuto è una parola in codice o meno.

Viene definita "sindrome" **s** il prodotto $\mathbf{H} \cdot \mathbf{c}^T$.

Per un dato vettore **v**, $\mathbf{v} \in C$ se e solo se $\mathbf{s} = \mathbf{H} \cdot \mathbf{v}^T = \mathbf{0}$.

Ci deve essere un numero pari nei componenti del prodotto $\mathbf{H} \cdot \mathbf{c}^T$ che aggiunge per dare $\mathbf{s} = \mathbf{0}$. Questo vincolo viene detto vincolo di parità.

Quando una parola di codice **c** viene trasmessa attraverso un canale, l'altra parte riceve un vettore **v.** Se $\mathbf{s} = \mathbf{H} \cdot \mathbf{v}^T \neq \mathbf{0}$, allora $\mathbf{v} \notin C$ e quindi si possono usare tecniche di correzione d'errore nel tentativo di correggere la **v** e recuperare **c**.

Il limite superiore per la capacità di correzione degli errori di un codice LDPC è determinato dalla distanza minima del codice. Per definire la distanza minima di un codice, prima di tutto bisogna definire il peso di Hamming e la distanza di Hamming.

Definizione 9: **Peso di Hamming**

Il peso di Hamming, w, di un vettore è il numero dei suoi elementi non zero. Il peso di Hamming di un vettore binario è quindi il numero di elementi 1 nel vettore.

La distanza di Hamming, d, tra due vettori, **x** e **y**, è il numero di posti in cui differiscono; viene indicata come d(**x**, **y**).

In letteratura, il peso di Hamming e la distanza di Hamming sono spesso indicati semplicemente con i termini "peso" e "distanza".

Il peso delle parole di codice influisce sul numero di operazioni eseguite nella decodifica e la distanza tra il codice le parole influiscono sul numero di errori che possono essere corretti.

Definizione 10: **Distanza minima**

La distanza minima di un codice C è definita come la più piccola distanza di Hamming tra due qualsiasi parole di codice diverse.

2.3 Decodifica

Il processo di decodifica serve per eliminare eventuali errori di codifica dovuti al rumore durante la trasmissione del segnale. I decodificatori possono essere basati su algoritmi a decisione "hard" oppure "soft". Nel caso di decodifica hard si considera solo il flusso di informazioni in ricezione e non viene presa in considerazione alcuna informazione sul canale. Il decisore è basato essenzialmente sul calcolo della sindrome. Gli ingressi di un decodificatore a decisione soft invece assumere l'intero intervallo di valori intermedi tra 0 ed 1, e sono intesi come valori probabilistici di prossimità ai valori possibili. Questa informazione extra indica l'affidabilità di ciascun punto di dati di input e viene utilizzata per formare stime migliori dei dati originali.

Per i codici LDPC ed MDPC si può ricorrere ai classici algoritmi di decodifica a decisione hard utilizzando algoritmi iterativi, che rientrano nella categoria di algoritmi "Bit-Flipping" (BF). Rinunciare all'utilizzo delle informazioni soft comporta ovviamente una perdita di prestazioni.

2.3.1 Bit-Flipping

L’algoritmo Bit-Flipping consente una decodifica iterativa che considera i nodi variabile ed i relativi bit in ingresso della parola di codice.
Considerando una matrice **H** con colonne di peso costante pari a dv, il messaggio da ciascun nodo di controllo c_i a ciascun nodo variabile v_i consiste ad ogni iterazione nella somma binaria di tutti i nodi adiacenti, escluso v_i . Un’equazione a controllo di parità è soddisfatta se il suo valore è pari a 0 mentre non è soddisfatta se il valore corrisponde ad 1. Ogni nodo variabile conta il numero di equazioni di controllo di parità non soddisfatte in base al valore di controllo di parità ricevuto.
Il numero di equazioni di controllo di parità non soddisfatte viene confrontato con una soglia intera opportunamente scelta, minore o uguale a dv - 1. La scelta della soglia è un elemento cruciale che influenza molto le prestazioni di questo algoritmo.
Se tale numero è maggiore o uguale al valore della soglia, allora v_i esegue il flip del bit (se è 0 diventa 1 e viceversa) e lo invia a c_i ; altrimenti invia il suo valore originale. L’algoritmo prosegue alle successive iterazioni, aggiornando le equazioni di controllo parità con questi nuovi valori, fino a quando tutte le equazioni di parità sono soddisfatte o viene raggiunto un numero massimo di iterazioni.

2.4 Trapping e Stopping sets

Il metodo di decodifica che viene scelto ha implicazioni dirette per la precisione e l'efficienza della decodifica.

I cicli sono stati la prima caratteristica negativa nota dei codici LDPC e sono stati ampiamente studiati, in quanto hanno avuto un impatto sulla precisione di codici LDPC ad alte prestazioni.

Riferendoci al grafo di Tanner, un ciclo è un sequenza di nodi collegati che formano un anello chiuso, dove il nodo iniziale e finale sono gli stessi e nessun nodo viene coinvolto più di una volta. La lunghezza del ciclo è il numero di nodi che un ciclo contiene.

Se non esistono cicli all'interno del grafo di Tanner della matrice di parità, una decodifica iterativa di tipo *belief propagation* ha sempre successo con sufficienti

iterazioni. Tuttavia, se i vicini di un nodo non sono condizionatamente indipendenti, tali metodi diventano imprecisi.

Una soluzione potrebbe essere quella di costruire una matrice di parità senza cicli. Tuttavia ciò non è necessario in quanto non tutti i cicli hanno un impatto negativo sull'efficienza di decodifica dei codici LDPC. Infatti, la restrizione circa i cicli può portare a vincoli sulla struttura del codice, che impedisce ulteriormente l'efficienza di decodifica.

I cicli che influiscono negativamente sull'efficienza di decodifica di codici LDPC si combinano per formare i cosiddetti Stopping Sets e Trapping Sets. [12]

Questi set portano ad un alto *errror floor* in costruzione di codici LDPC altrimenti efficienti attraverso vari canali di comunicazione e influiscono su tutti gli algoritmi di decodifica ad alte prestazioni.

In seguito ci riferiremo agli studi relativi a Trapping Sets e Stopping Sets per analizzare i casi in cui le chiavi di un crittosistema, avendo particolari proprietà, inducono il decoder ad avere una probabilità di fallire relativamente alta.

3 CRITTOGRAFIA BASATA SU CODICI

Verranno ora brevemente presentati i crittosistemi esaminati in questo lavoro, insieme ad un riassunto dei loro algoritmi di funzionamento. Si tratta di alcuni versioni di LEDAcrypt e BIKE, entrambi presentati per la gara di standardizzazione di crittografia post-quantistica del NIST.

Per completezza verrà fornita una descrizione completa di entrambi i sistemi, ma è bene notare che essi sono fondamentalmente simili, si basano sugli stessi tipi di codici e differiscono principalmente per le tecniche di decodifica implementate.

Verranno però prima presentati i crittosistemi di McEliece e Niederreiter. Essi sono di fondamentale importanza, in quanto sono stati tra i primi a proporre l'uso di codici sparsi con finalità crittografiche. I crittosisistemi più moderni si basano spesso su questi (ad esempio BIKE-1 è una variante di McEliece e LEDAcrypt deriva da Niederreiter), perciò è opportuno riportarne una breve descrizione.

3.1 Crittosistema di McEliece

Il sistema di crittografia McEliece [5] è un algoritmo di crittografia asimmetrico sviluppato nel 1978 da Robert McEliece.

L'algoritmo si basa sulla difficoltà di decodificare un codice lineare qualsiasi. Per la chiave privata, viene selezionato un codice a correzione d'errore per il quale è noto un algoritmo di decodifica efficiente che è in grado di correggere t errori. L'algoritmo originale utilizza codici Goppa binari. La chiave pubblica deriva dalla chiave privata, senza rendere noto il codice che si sta usando. Per ottenere ciò, la matrice generatrice del codice **G** viene moltiplicata per due matrici invertibili selezionate in modo casuale: **S** e **P**.

Esistono una serie di parametri di sicurezza comuni: {n,k,t}, che sono noti.

Generazione delle chiavi

- **Input:** I parametri n, k, t di un codice.

- **Output:** La chiave pubblica e la chiave privata.

1. Si seleziona codice (*n*, *k*) lineare, capace di correggere t errori, per il quale è noto un algoritmo di decodifica efficiente A.
2. Si seleziona casualmente una matrice binaria **S** non-singolare, di dimensioni k x k.
3. Si genera casualmente una matrice di permutazione **P**, di dimensioni n x n.
4. Si calcola la $\mathbf{G}^{pub} = \mathbf{SGP}$, di dimensioni k x n.
5. La chiave pubblica è $\{\mathbf{G}^{pub}, t\}$; la chiave privata è data dalla terna $\{\boldsymbol{S}, \boldsymbol{P}, A\}$.

Cifratura

- **Input:** Un vettore binario che rappresenta il messaggio **m**, la chiave pubblica e il parametro t.

- **Output:** Il testo cifrato **c.**

1. Il messaggio **m** viene codificato come una stringa binaria di lunghezza k.
2. Si calcola $\mathbf{c'} = \mathbf{mG}^{\mathbf{pub}}$.
3. Si genera casualmente un vettore **z** con peso pari a t.
4. Si calcola il testo cifrato come $\mathbf{c} = \mathbf{c'} + \mathbf{z}$.

Decifratura

- **Input:** Il testo cifrato **c**.

- **Output:** Il messaggio originale **m**.

1. Si calcola $\mathbf{cP}^{-1}$.
2. Si applica l'algoritmo di decodifca A in modo da recuperare **m'**.
3. Infine si ri-ottiene **m** come $\mathbf{m} = \mathbf{m'S}^{-1}$.

3.2 Crittosistema di Niederreiter

Il sistema crittografico di Niederreiter [6], sviluppato nel 1986 da Harald Niederreiter, utilizza una sindrome come testo cifrato mentre il messaggio è un pattern di errore. E' una variante di quello di McEliece che applica la stessa idea alla matrice di controllo di parità **H** di un codice lineare. I due sistemi sono equivalenti dal punto di vista della sicurezza, ma la cifratura di Niederreiter è più veloce della cifratura di McEliece.

Generazione delle chiavi

- **Input:** I parametri n, k, t di un codice, per il quale è noto un algoritmo di decodifica efficiente.

- **Output:** La chiave pubblica e la chiave privata.

1. Si genera una matrice di parità $\boldsymbol{H}$, di dimensioni $(n - k) \times n$.
2. Si genera una matrice casuale non-singolare **S**, di dimensioni $(n - k) \times (n - k)$
3. Si genera una matrice di permutazione casuale **P**, di dimensioni $n \times n$.
4. Viene calcolata $\mathbf{H}^{\text{pub}} = \mathbf{SHP}$, di dinensioni $(n - k) \times n$.
5. La chiave pubblica è $\{\boldsymbol{H}^{\mathbf{pub}}, t\}$; la chiave privata è data dalla terna $\{\boldsymbol{S}, \boldsymbol{H}, \boldsymbol{P}\}$.

Cifratura

- **Input:** Un vettore binario che rappresenta il messaggio **m** e la chiave pubblica.

- **Output:** Il testo cifrato **c.**

1. Si codifica il messaggio **m** come vettore binario $\mathbf{e}^{\mathbf{m'}}$, di lunghezza n e peso massimo t.
2. Si calcola il testo cifrato come $\boldsymbol{c} = \boldsymbol{H}^{\text{pub}}\ \mathbf{e}^{\mathbf{m'}}$.

Decifratura

- **Input:** Il testo cifrato **c**.

- **Output:** Il messaggio originale **m**.

1. Si calcola $\boldsymbol{S}^{-\mathbf{1}}\boldsymbol{c} = \boldsymbol{HPm}^{\mathbf{T}}$.
2. Si applica un algoritmo di decodifica su $\boldsymbol{G}$ per recuperare $\boldsymbol{Pm}^{\mathbf{T}}$.

3.3 LEDAcrypt

LEDAcrypt è una suite di crittografia post-quantistica asimmetrica costruita su codici Quasi-Cyclic Low Density Parity Check (QC-LDPC). La suite include un meccanismo di incapsulamento della chiave (KEM) e un sistema di crittografia a chiave pubblica (PKC).

In questo lavoro si è analizzata la versione della specifica del sistema di crittografia v2. [1]. I risultati delle analisi possono essere semplicemente adattati a versioni future. Infatti anche nelle elaborazioni numeriche qui presentate si sono modificati alcuni parametri in modo che l'esecuzione del codice fosse più simile alle implementazioni successive. Di seguito si farà un riassunto (non esaustivo) del funzionamento di LEDAcrypt KEM-NIEDERREITER.

Lo schema LEDAcrypt-KEM impiega chiavi a lungo termine per gli scenari in cui è auspicabile il riutilizzo delle chiavi. Supponendo l'uso di un decodificatore QC-LDPC con DFR opportunamente bassa, la cifratura e la decifratura dello schema OW-CPA Niederreiter, sono utilizzati nella costruzione riportata per ottenere algoritmi di l'incapsulamento e decapsulamento della chiave (KEM) con garanzia di sicurezza IND-CCA2.

Le garanzie di sicurezza IND-CCA2 di LEDAcrypt-KEM si ottengono utilizzando un decoder QC-LDPC con DFR=$2^{-\lambda}$ e la costruzione descritta dagli algoritmi Incapsulamento e Decapsulamento, garantendo una protezione con un livello di sicurezza 2^{λ} contro gli avversari sia passivi che attivi.

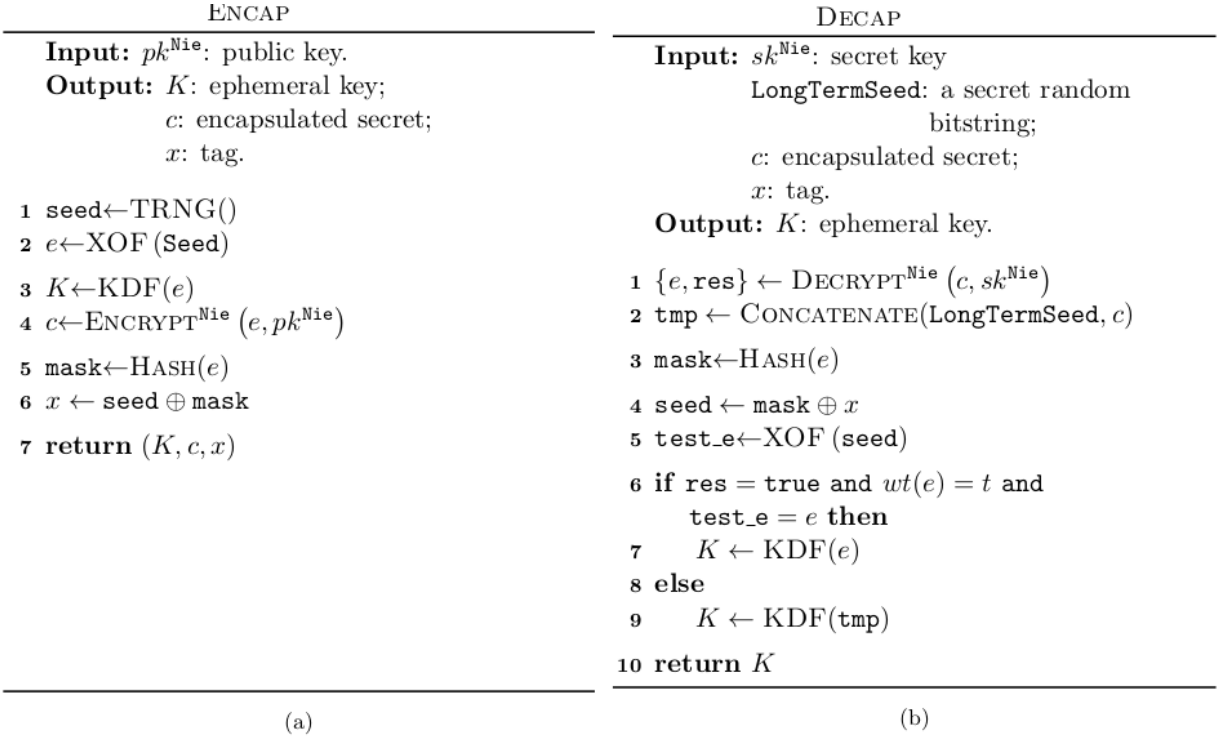

Figura 3: Algoritmi per l'incapsulamento ed il decapsulamento di LEDAcrypt.

3.3.1 LEDAcrypt KEM-NIEDERREITER

LEDAcrypt KEM-NIEDERREITER deriva da un codice Low Density Parity Check basato su matrici basato su matrici Quasi-Cicliche (QC-LDPC), con lunghezza delle parole di codice $n = pn_0$ e lunghezza dell'informazione $k = p \cdot (n_0 - 1)$, in cui $n_0 \in \{ 2, 3, 4 \}$.

Opera seguendo le fasi che vengono riportate di seguito.

Generazione delle chiavi

- **Input:** Il parametro $n_0 \in \{2, 3, 4\}$ ed il livello di sicurezza.
- **Output:** La chiave privata PK e la chiave pubblica SK.

1. Si genera casualmente **H,** matrice Quasi-Ciclica: $\mathbf{H} = [\mathbf{H_0} , \mathbf{H_1} , \dots , \mathbf{H_{n0-1}}]$. Ogni $\mathbf{H_x}$ è [p x p] con peso dv pari per ogni riga/colonna. Perciò la dimensione risultante di **H** è [p x pn_0].
2. Si genera casualmente **Q,** una matrice di dimensioni [pn_0 x pn_0] , anch'essa Quasi-Ciclica
3. Si calcola la matrice **L**, che è definita come loro prodotto:

   $\mathbf{L} = \mathbf{HQ} = [\mathbf{L_0}, \mathbf{L_1} , \dots , \mathbf{L_{n0-1}}]$, che ha lo stesso rango di **H** .
4. Si calcola la matrice M , così definita:

   $\mathbf{M} = (\mathbf{L_{n0-1}})^{(-1)} \mathbf{(L)} = [\mathbf{M_0} , \mathbf{M_1} , \dots , \mathbf{M_{n0-2}} , \mathbf{I_p}] = [\mathbf{M_l}\ \mathbf{I}]$
5. La chiave segreta è costituita dalla coppia di matrici **H** e **Q**.

   SK = { **H**, **Q** }.
6. La chiave pubblica corrisponde alla matrice **M**. PK = **M**.

Incapsulamento

- **Input:** La chiave pubblica ed un vettore **e ,** di dimensioni [1 x pn_0], con un numero di elementi 1 pari a t .

- **Output:** Il testo cifrato, dato dal vettore **s**.

1. Si calcola la chiave $\mathbf{M} = (\mathbf{L}_{n0-1})^{(-1)}(\mathbf{L}) = [\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_{n0-2}, \mathbf{I}_p] = [\mathbf{M}_l\ \mathbf{I}]$
2. Si ottiene il testo cifrato, chiamata "sindrome", come $\mathbf{s} = \mathbf{M}\,\mathbf{e}^T$.

Decapsulamento

- **Input:** La chiave segreta SK = {**H**, **Q**} ed il testo cifrato.

- **Output:** Il vettore **e**.

1. Si calcola **L** = **HQ**
2. Si calcola $\mathbf{S'} = \mathbf{L}_{no-1}\mathbf{S} = \mathbf{H}(\mathbf{Q}\mathbf{e}^T) = \mathbf{H}(\mathbf{e}\mathbf{Q}^T)^T$
3. Si recupera $\mathbf{e}\mathbf{Q}^T$ con opportune tecniche di decodifica, ed essendo **Q** nota si recupera quindi **e**.

3.3.2 LEDAcrypt v3: Adattamento

Per rendere consistente il codice usato con l'approccio delle versioni di LEDAcrypt seguenti, l'algoritmo è stato modificato, ottenendo così un peso sbilanciato per **H** e **Q**.

Ciò coincide con la considerazione di quelle particolari istanze di LEDAcrypt in cui la matrice **Q** si riduce a una matrice di permutazione quasi-ciclica e quindi può essere trascurata. Pertanto, da questo punto in poi, considereremo semplicemente il caso **L** = **HI** = **H**. Notiamo che prendendo **Q** = **I** non si influisce negativamente sull'unidirezionalità dei crittosistemi di Niederreiter e McEliece, a condizione che il peso di una colonna di **L** = **H** sia scelto tenendo conto di **Q** = **I** (cioè, prendendo $v \approx dv \cdot m$, dove dv è il peso della colonna della vecchia matrice **H** ed m il peso della colonna della vecchia matrice **Q**).

Perciò pur mantenendo per le analisi qui proposte il codice della versione v2 sono stati modificati i parametri del codice in modo che la matrice **Q** corrispondesse alla matrice identità.

3.3.3 LEDAcrypt : Decodifica

Sebbene in LEDAcrypt sia possibile eseguire la decodifica del codice QC-LDPC privato utilizzando la matrice di controllo di parità **H** e un decodificatore Bit-Flipping classico, tale scelta non sfrutterebbe al massimo l'efficienza e le capacità di correzione del codice QC-LDPC.

Infatti, data una sindrome $\mathbf{S'} = \mathbf{H}(\mathbf{eQ}^{\mathbf{T}})^{\mathbf{T}}$, le posizioni degli errori $\mathbf{e'} = \mathbf{eQ}^{\mathbf{T}}$ da correggere non sono distribuiti in modo uniforme; essi dipendono invece dagli elementi di $\mathbf{Q}^{\mathbf{T}}$, i quali sono noti al decoder.

Partendo dal classico Bit-Flipping, è stato sviluppato un decoder migliorato che è progettato specificamente per LEDAcrypt, dove le posizioni delle voci impostate in $\mathbf{e'} = \mathbf{eQ}^{\mathbf{T}}$ da correggere sono influenzate dal valore di $\mathbf{Q}^{\mathbf{T}}$, in quanto **e'** è equivalente ad un vettore di errore casuale **e** con peso **t** moltiplicato per $\mathbf{Q}^{\mathbf{T}}$.

Dato che questo decoder tiene conto di tale moltiplicazione per la trasposizione della matrice **Q** per stimare le posizioni dei bit da flippare con maggiore efficienza, è stato chiamato "Q-decoder".

Tale algoritmo tenta di a ricostruire il vettore di errore segreto della sindrome ricevuta **S**. A tal fine, la peculiarità dell'algoritmo Q-decoder è quella di ricostruire direttamente il valore di **e**, a differenza di un comune algoritmo di bit-flipping che recupererebbe $\mathbf{e'} = \mathbf{eQ}^{\mathbf{T}}$, richiedendo così un'ulteriore moltiplicazione matriciale per completare l'azione di decifratura.

Ad ogni modo quando la matrice **Q** coincide con l'identità questo algoritmo si riduce ad una decodifica Bit-Flipping.

3.4 BIKE v3

BIKE è una suite di algoritmi KEM basati su codici QC-MDPC che possono essere decodificati usando tecniche di bit-flipping.

BIKE prevede tre modi di funzionamento, chiamati BIKE-1, BIKE-2 e BIKE-3.

Nelle pagine successive si farà un riassunto del funzionamento di BIKE-1, perché è la variante che è stata studiata in questo lavoro.

Nella tabella 1 viene riassunto il funzionamento di tutte le varianti.

	BIKE-1	BIKE-2	BIKE-3
SK	(h_0, h_1) with $\lvert h_0\rvert = \lvert h_1\rvert = w/2$		
PK	$(f_0, f_1) \leftarrow (gh_1, gh_0)$	$(f_0, f_1) \leftarrow (1, h_1 h_0^{-1})$	$(f_0, f_1) \leftarrow (h_1 + gh_0, g)$
Enc	$(c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$	$c \leftarrow e_0 + e_1 f_1$	$(c_0, c_1) \leftarrow (e + e_1 f_0, e_0 + e_1 f_1)$
	$K \leftarrow \mathbf{K}(e_0, e_1)$		$K \leftarrow \mathbf{K}(e_0, e_1, e)$
Dec	$s \leftarrow c_0 h_0 + c_1 h_1$; $u \leftarrow 0$	$s \leftarrow c h_0$; $u \leftarrow 0$	$s \leftarrow c_0 + c_1 h_0$; $u \leftarrow t/2$
	$(e'_0, e'_1) \leftarrow \texttt{Decode}(s, h_0, h_1, u)$		$(e'_0, e'_1, e') \leftarrow \texttt{Decode}(s, h_0, h_1, u)$
	$K \leftarrow \mathbf{K}(e'_0, e'_1)$		$K \leftarrow \mathbf{K}(e'_0, e'_1, e')$

Tabella 1: Riassunto schematico del funzionamento degli algoritmi BIKE proposti nella versione v3.

3.4.1 BIKE-1

Nella variante BIKE-1 viene privilegiata una generazione di chiavi veloce utilizzando una variante di McEliece. A differenza di McEliece con codici QC-MDPC (e di qualsiasi variante di QC-McEliece), non viene calcolata l'inversione di uno dei blocchi ciclici privati che sarebbe poi moltiplicata per l'intera matrice privata per ottenere una forma sistematica. Si nasconde invece la struttura del codice privato semplicemente moltiplicando la sua matrice privata sparsa per un qualsiasi blocco ciclico casuale e denso. Lo svantaggio che ciò comporta è che la dimensione della chiave pubblica e dei i dati è grande il doppio, dato che la chiave pubblica non avrà più un blocco coincidente con la matrice identità. Inoltre la crittografia McEliece viene reinterpretata come se il messaggio venisse trasmesso nel vettore di errore, piuttosto che nella parola di codice.

L'algoritmo risultante opera seguendo le fasi di seguito, mentre i parametri sono riportati in tabella 2.

	n	r	w	t
Security Level 1	23558	11779	142	134
Security Level 3	49642	24821	206	199
Security Level 5	81194	40597	274	264

Tabella 2: Parametri di BIKE v3.

Generazione delle chiavi

- **Input:** λ , il livello di sicurezza desiderato.
- **Output**: Le chiavi private sparse ($\mathbf{h_0}$, $\mathbf{h_1}$) e le chiavi pubbliche dense ($\mathbf{f_0}$, $\mathbf{f_1}$) .

1. Dato λ, vengono scelti i valori per i parametri r , w .
2. Vengono generate ($\mathbf{h_0}$, $\mathbf{h_1}$) entrambe di peso pari $|\mathbf{h_0}| = |\mathbf{h_1}| = w/2$.
3. Viene generato g di peso dispari (così che $|\mathbf{g}| \approx r/2$) .
4. Sono calcolate ($\mathbf{f_0}$, $\mathbf{f_1}$) = ($\mathbf{gh_0}$, $\mathbf{gh_1}$) .

Incapsulamento

- **Input:** le chiavi pubbliche dense ($\mathbf{f_0}$, $\mathbf{f_1}$)
- **Output:** la chiave incapsulata **K** e il crittogramma **c**.

1. Si scelgono ($\mathbf{e_0}$, $\mathbf{e_1}$) $\in R^2$ in modo che $|\mathbf{e_0}| + |\mathbf{e_1}| = t$.
2. Viene generato **m** .
3. Si calcola c = ($\mathbf{c_0}$, $\mathbf{c_1}$) $\leftarrow$ ($\mathbf{mf_0} + \mathbf{e_0}$, $\mathbf{mf_1} + \mathbf{e_1}$) .
4. Si calcola $\mathbf{K} \leftarrow K(\mathbf{e_0}, \mathbf{e_1})$.

Decapsulamento

- **Input:** le chiavi private sparse ($\mathbf{h_0}$, $\mathbf{h_1}$) e il crittogramma c.
- **Output:** la chiave decapsulata **K** o un segnale di fallimento.

1. Si calcola la sindrome $\mathbf{s} = (\mathbf{c_0h_0} + \mathbf{c_1h_1})$.
2. Si prova a decodificare **s** per recuperare i vettori ($\mathbf{e_0}$, $\mathbf{e_1}$) .
   $K \leftarrow K(\mathbf{e_0}, \mathbf{e_1})$
3. Se $|(\mathbf{e_0}, \mathbf{e_1})| \neq t$ o la decodifica ha fallito, l'output è un segnale di errore .
   Altrimenti, è possibile calcolare $\mathbf{K} \leftarrow K(\mathbf{e_0}, \mathbf{e_1})$.

3.4.2 BIKE v3: Decodifica

Viene usato il decodificatore descritto nell'immagine di figura 4.

Questo algoritmo restituisce un vettore di errore valido quando si ferma, ma potrebbe non fermarsi. Perciò nella pratica viene imposto un tempo massimo di funzionamento, raggiunto il quale l'algoritmo si ferma.

Dati particolari valori per r, w, e t, si ha $n = 2 \cdot r$ e $k = r$.

Inoltre, bisogna impostare i valori per S e δ e fornire una regola per il calcolo della soglia di fallimento. Per la decodifica di una sindrome con rumore (richiesta da BIKE-3 varianti), si deve anche recuperare il vettore di errore **e** e bisogna calcolare la trasposizione della sindrome. La trasposizione inverte in pratica l'ordine della sindrome risultante **S'**, ad eccezione del primo elemento che rimane invariato.

Algorithm 3 One-Round Bit Flipping Algorithm

Require: $H \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$, integer $u \geq 0$
Ensure: $\left|s - eH^T\right| \leq u$

$T \leftarrow \texttt{threshold}(|s|)$
for $j = 0, \ldots, n-1$ **do**
  $\ell \leftarrow \texttt{min}(\texttt{ctr}(H, s, j), T)$
  $J_\ell \leftarrow J_\ell \cup \{j\}$ *// all J_ℓ empty initially*
$e \leftarrow J_T$ (**)
$s' \leftarrow s - eH^T$
while $|s'| > S$ **do** (***)
  for $\ell = 0, \ldots, \delta$ **do** (***)
    $e' \leftarrow \texttt{check}(H, s', J_{T-\ell}, d/2)$
    $(e, s') \leftarrow (e + e', s' - e'H^T)$ *// update error and syndrome*
$e' \leftarrow \texttt{check}(H, s', e, d/2)$ (**)
$(e, s') \leftarrow (e + e', s' - e'H^T)$ *// update error and syndrome*
while $|s'| > u$ **do**
  $j \leftarrow \texttt{guess\_error\_pos}(H, s', d/2)$
  $(e_j, s') \leftarrow (e_j + 1, s' + h_j)$ (*)
if BIKE-1 or BIKE-2 **then**
  return e
else
  return $(e, transpose(s'))$ *// BIKE-3*

$\texttt{check}(H, s, J, T)$
  $e \leftarrow 0$
  for $j \in J$ **do**
    if $\texttt{ctr}(H, s, j) \geq T$ **then**
      $e_j \leftarrow 1$
  return e

$\texttt{guess\_error\_pos}(H, s, T)$
  loop *// until success*
    $i \xleftarrow{\$} s$ (**)
    for $j \in eq_i$ **do** (*),(**)
      if $\texttt{ctr}(H, s, j) \geq T$ **then**
        return j

$\texttt{ctr}(H, s, j)$
  return $|h_j \cap s|$ (*),(**)

$\texttt{threshold}(S)$
  return function of r, w, t, and S

(*) *h_j the j-th column of H (as a row vector), eq_i the i-th row of H*
(**) *we identify binary vectors with the set of their non zero positions*
(***) *the algorithm uses two parameters S and δ which depend of r, w, and t*

Figura 4: Algoritmo di decodifica usato in BIKE v3.

3.5 BIKE v4

L'ultima versione di BIKE rilasciata nel momento in cui viene scritto questo elaborato (giugno 2020) è la versione v4 proposta per il Round 3 del *Post-Quantum Cryptography Standardization project* del NIST.

Su consiglio del NIST, per ridurre il numero di opzioni proposte dai progettisti di BIKE, la versione v4 consolida BIKE in una sola versione, basata su BIKE-2-CCA, che viene quindi chiamata semplicemente BIKE.

BIKE v4 non coincide esattamente con BIKE-2-CCA v3, perché sono stati introdotti alcuni piccoli cambiamenti algoritmi di incapsulamento e decapsulamento. Queste modifiche sono apportate in base a quanto evidenziato dallo studio in [11], al fine di dimostrare che BIKE abbia sicurezza IND-CCA, sempre a patto che il decoder associato abbia un DFR sufficientemente basso.

Di seguito riportiamo quindi l'algoritmo dell'unica variante KEM prevista da BIKE v4. La funzione H() è una espansione pseudorandom del seme lungo 256 bit che viene dato in ingresso. K() e L() sono funzioni che restituiscono i 256 bit meno significativi del digest SHA384 dell'input. Per informazioni più dettagliate si può consultare il paper di riferimento [3].

Generazione delle chiavi

- **Input:** I parametri {n, w, t, l}.
- **Output:** La chiave privata ($\mathbf{h_0}$,$\mathbf{h_1}$,σ) e la chiave pubblica **h**.

1. Si generano ($\mathbf{h_0}$,$\mathbf{h_1}$) entrambe di peso pari $|\mathbf{h_0}| = |\mathbf{h_1}| = w/2$.
2. Viene generato σ nell'intervallo {0,1} con distribuzione casuale e uniforme.
3. Si calcola $\mathbf{h} \leftarrow \mathbf{h_1}\mathbf{h_0}^{-1}$
4. Restituisce ($\mathbf{h_0}$,$\mathbf{h_1}$,σ) e **h**.

Incapsulamento

- **Input:** La chiave pubblica **h**.
- **Output:** La chiave incapsulata **K** e il testo cifrato $\mathbf{C}= (\mathbf{c_0},\mathbf{c_1})$.

1. Viene generato m in {0,1} con distribuzione casuale e uniforme.
2. Si calcola $(\mathbf{e_0},\mathbf{e_1}) \leftarrow H(m)$.
3. Si calcola $\mathbf{C}= (\mathbf{c_0},\mathbf{c_1}) \leftarrow (\mathbf{e_0}+\mathbf{e_1}\ \mathbf{h}, m \oplus L(\mathbf{e_0},\mathbf{e_1}))$.
4. Si elabora $\mathbf{K} \leftarrow K(m,\mathbf{C})$.
5. Vengono restiuiti (**C**,**K**).

Decapsulamento

- **Input:** La chiave privata ($\mathbf{h_0}$,$\mathbf{h_1}$,σ) e il testo cifrato $\mathbf{C}= (\mathbf{c_0},\mathbf{c_1})$.
- **Output:** La chiave decapsulata **K**.

1. Vengono generati ($\mathbf{e_0}'$,$\mathbf{e_1}'$).
2. Si calcolano le sindromi $\mathbf{s} \leftarrow \mathbf{c_0}\mathbf{h_0}$.
3. Si calcola $\{(\mathbf{e''_0},\mathbf{e''_1}), \perp\} \leftarrow$ decoder($\mathbf{s}$,$\mathbf{h_0}$,$\mathbf{h_1}$). ($\perp$ indica il simbolo di fallimento).

4. Se $(\mathbf{e''_0},\mathbf{e''_1}) \leftarrow \text{decoder}(\mathbf{s},\mathbf{h_0},\mathbf{h_1})$ e $|(\mathbf{e''_0},\mathbf{e''_1})| = \mathbf{t}$ allora

$(\mathbf{e'_0},\mathbf{e'}) \leftarrow (\mathbf{e''_0},\mathbf{e''_1})$.

5. $\mathbf{m'} \leftarrow \mathbf{c_1} \oplus L(\mathbf{e'_0},\mathbf{e'})$

6. Se $H(\mathbf{m'}) \neq (\mathbf{e'_0},\mathbf{e'})$ si calcola $\mathbf{K} \leftarrow K(\sigma,\mathbf{C})$.

7. Altrimenti si calcola $\mathbf{K} \leftarrow K(\mathbf{m'},\mathbf{C})$.

8. L'output restituito è K.

I parametri di w e t sono gli stessi della versione precedente, mentre il valore suggerito per r presenta una leggera variazione (+4.61% per SL=1 e -0.5% per SL=3).

	n	r	w	t
Security Level 1	23558	11779	142	134
Security Level 3	49642	24821	206	199
Security Level 5	81194	40597	274	264

Tabella 3: Parametri di BIKE v4.

In questa versione non è prevista una versione capace di soddisfare i requisiti per il Security Level 5; inoltre gli autori non dichiarano che BIKE abbia sicurezza IND-CCA.

4 SOGLIE DI DECODIFICA IN BIKE

Si è ritenuto utile approfondire il modo in cui in BIKE (sia v3 che v4) vengono scelte le soglie di decodifica. Si tratta di un argomento di particolare importanza perché a partire dallo studio di queste sono state individuati il tipo di chiavi deboli su cui si basa l'analisi del capitolo 8.

Le soglie sono scelte a partire da considerazioni teoriche [10].

Sono implementate di fatto in maniera molto semplice, ed i loro valori sono definiti come di seguito:

- for BIKE-1 and BIKE-2
  - security level 1: $T = \lceil 13.530 + 0.0069722\ |s| \rceil$,
  - security level 3: $T = \lceil 15.932 + 0.0052936\ |s| \rceil$,
  - security level 5: $T = \lceil 17.489 + 0.0043536\ |s| \rceil$,
- for BIKE-3
  - security level 1: $T = \lceil 13.209 + 0.0060515\ |s| \rceil$,
  - security level 3: $T = \lceil 15.561 + 0.0046692\ |s| \rceil$,
  - security level 5: $T = \lceil 17.061 + 0.0038459\ |s| \rceil$.

Figura 5: BIKE: Formule per il calcolo delle soglie di decodifica.

Quindi in pratica le soglie vengono decise con delle formule empiriche che dipendono di fatto solo dal peso della sindrome e dal livello di sicurezza. Ciò viene fatto ad ogni iterazione della decodifica.

Di seguito viene graficato l'andamento della relazione tra peso della sindrome e soglia di decodifica per BIKE-1 con Security Level 1.

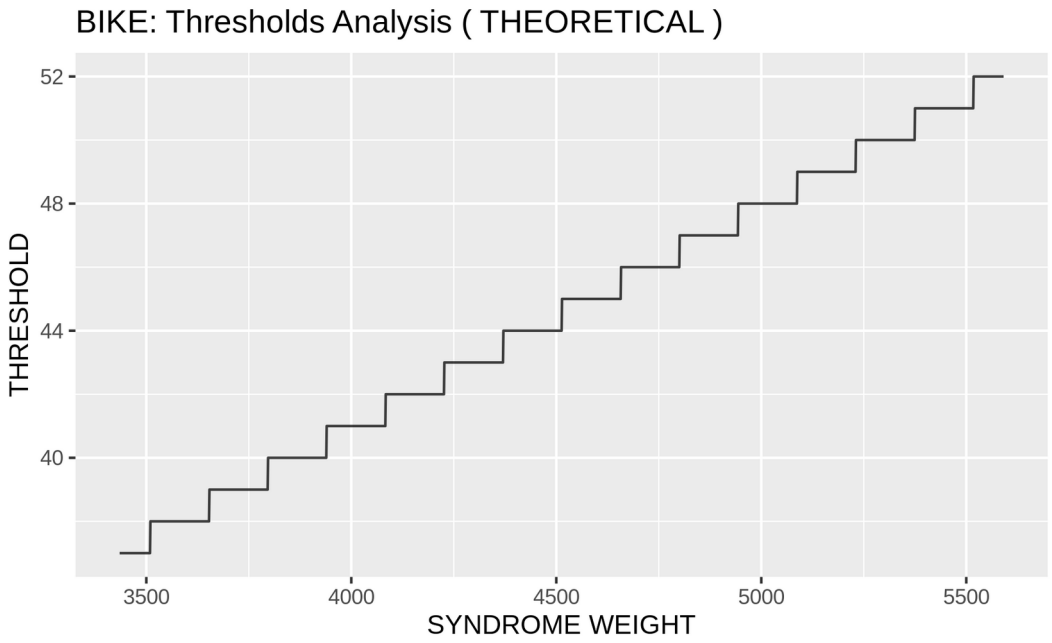

Figura 6: Relazione tra il peso della sindrome e la soglia di decodifica in BIKE. Grafico ottenuto graficando la formula in figura 7.

Può essere utile notare che i valori decimali sono arrotondati sempre per difetto. L'andamento è perfettamente congruente a quanto restituito effettivamente dal codice quando viene eseguito, come mostrato nella figura 7 di seguito; essa mostra la stessa relazione graficata in base ai dati risultanti da prove sperimentali.

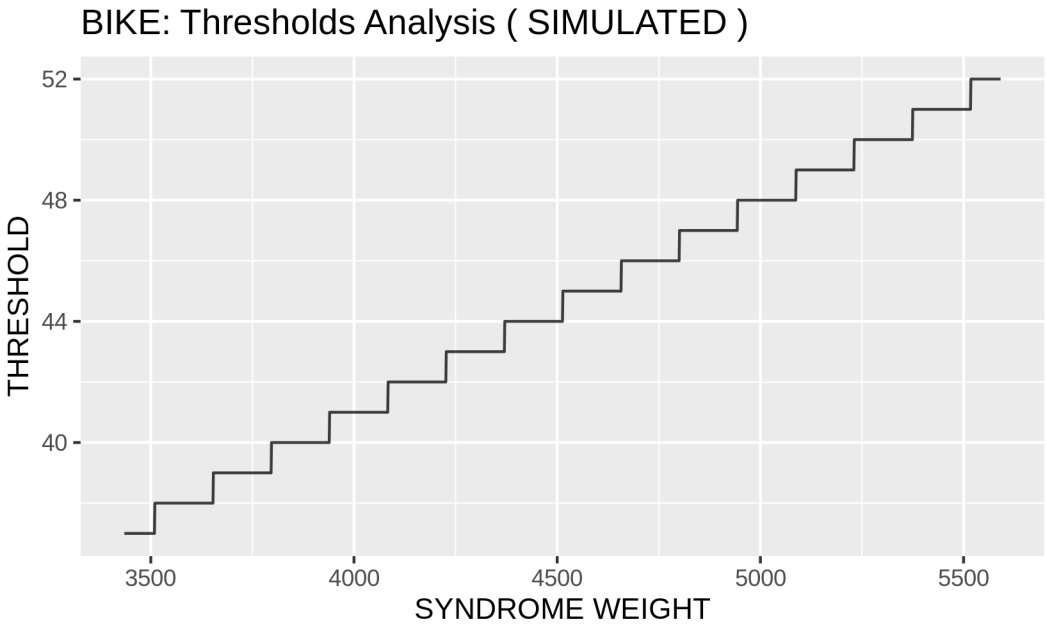

Figura 7: Relazione tra il peso della sindrome e la soglia di decodifica in BIKE. Grafico ottenuto graficando gli esiti di 1000 simulazioni..

Per ottenerla sono state effettuati dei test, eseguendo 10000 prove di cifratura/decifratura per ciascuno valore di t, rappresentato dalla variabile NUM_ERRORS_T, nell'intervallo 137 (valore suggerito dalla specifica) ± 60, cioè t ∈ [77,197].

Si è così ottenuta una statistica sul valore del peso della sindrome $\mathbf{s} = (\mathbf{c_0 h_0} + \mathbf{c_1 h_1})$ al variare di t. Si possono dedurre di conseguenza considerazioni anche sul valore delle soglie di decodifica, dato che come si è detto le due variabili sono legate da una relazione lineare.

Di seguito sono riportate le distribuzioni del peso della sindrome per alcuni valori di t selezionati, ottenuti simulando sempre 10000 prove per ogni valore di t.

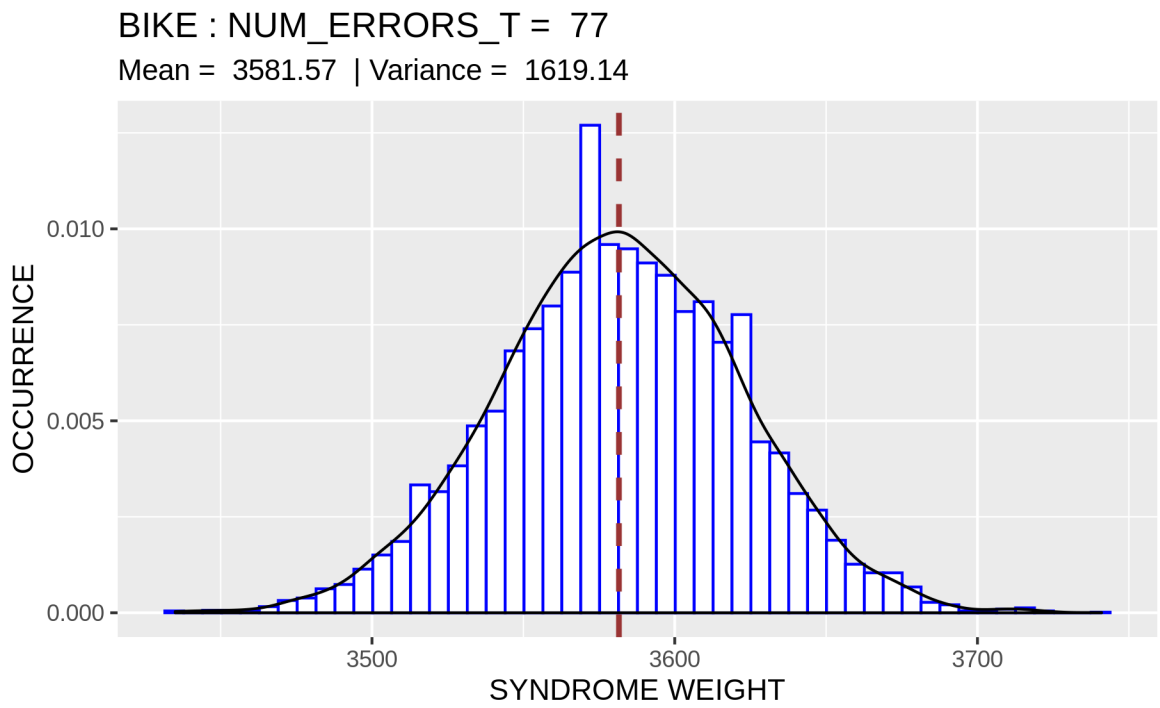

Figura 8: BIKE v3 : Distribuzione statistica del peso della sindrome quando t = 77.

Per alcuni valori di NUM_ERRORS_T, come in figura 9 e 10, ci sono dei valori del peso della sindrome che hanno un picco di probabilità di occorrenza. Questo fenomeno non è stato approfondito.

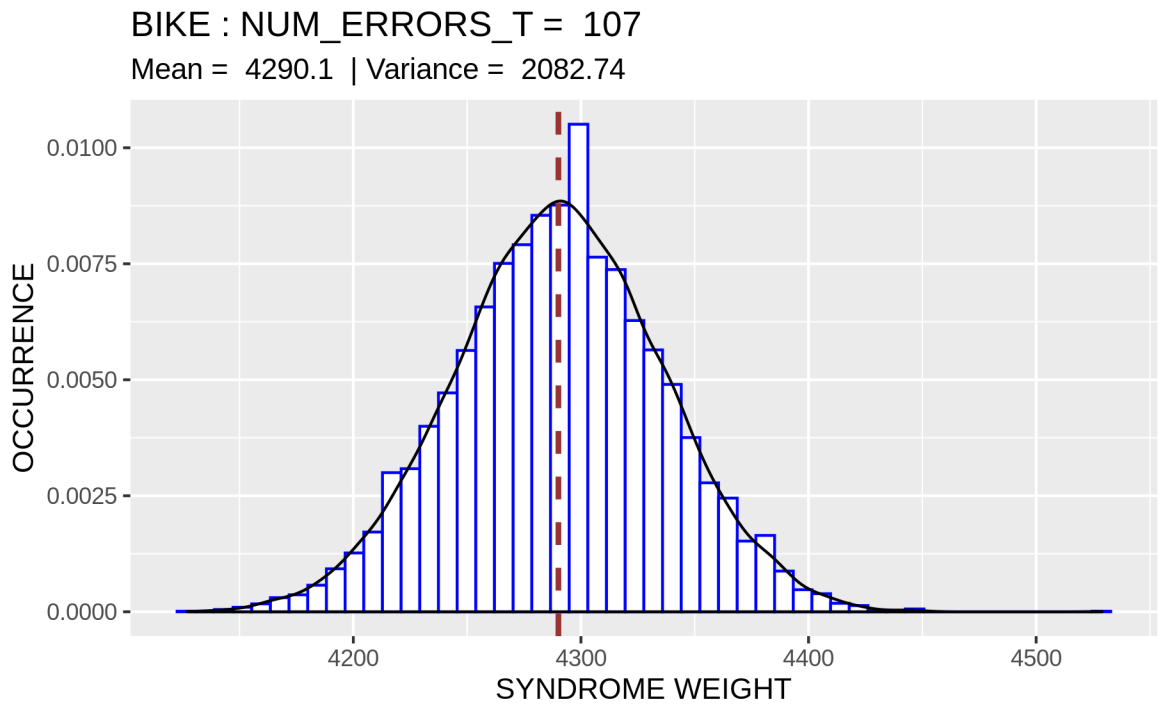

Figura 9: BIKE v3 : Distribuzione statistica del peso della sindrome quando t = 107.

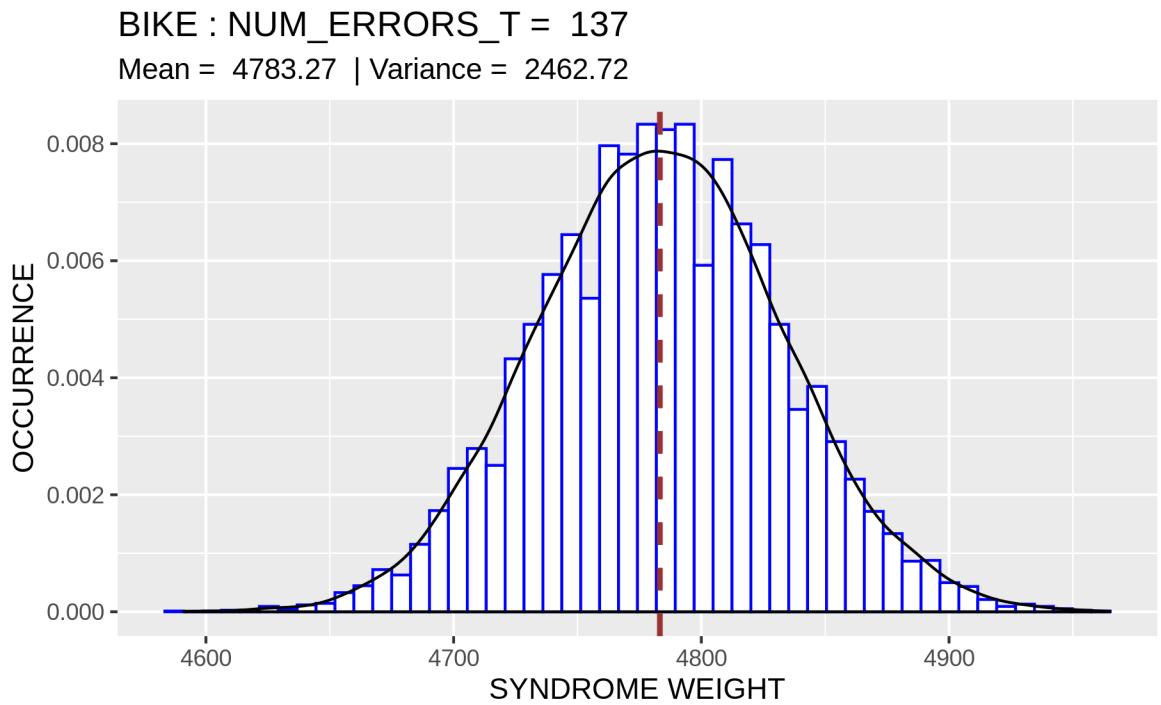

Figura 10: BIKE v3 : Distribuzione statistica del peso della sindrome quando t = 137.

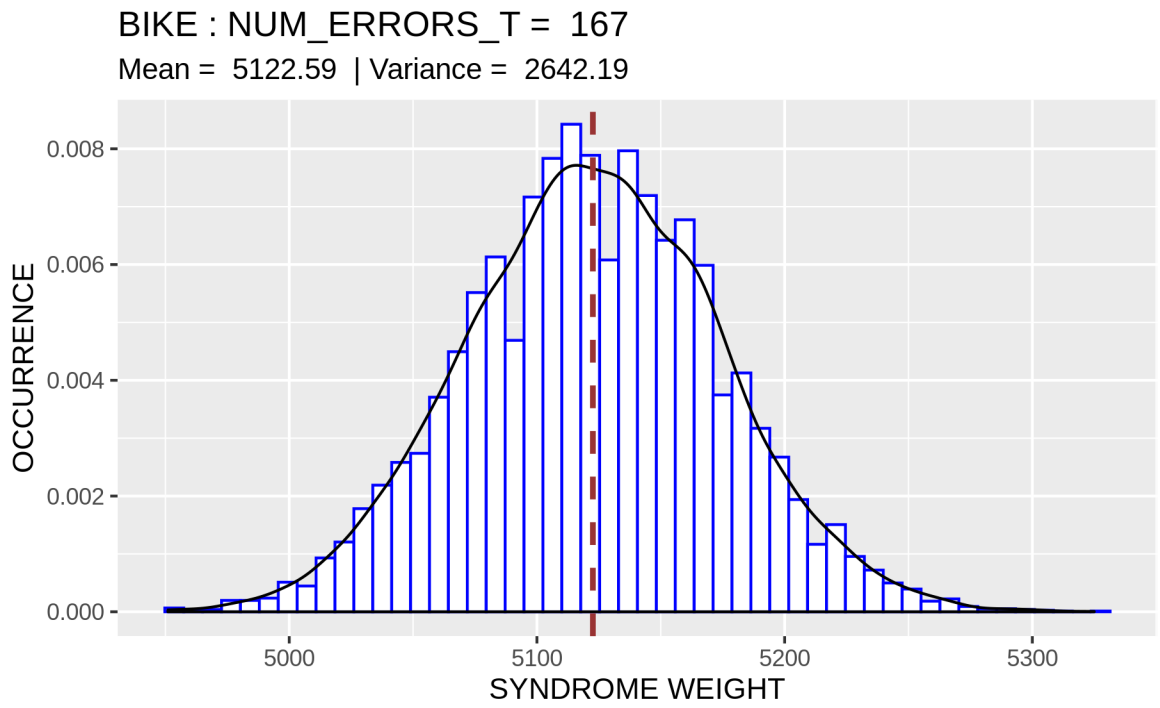

Figura 11: BIKE v3 : Distribuzione statistica del peso della sindrome quando t = 167.

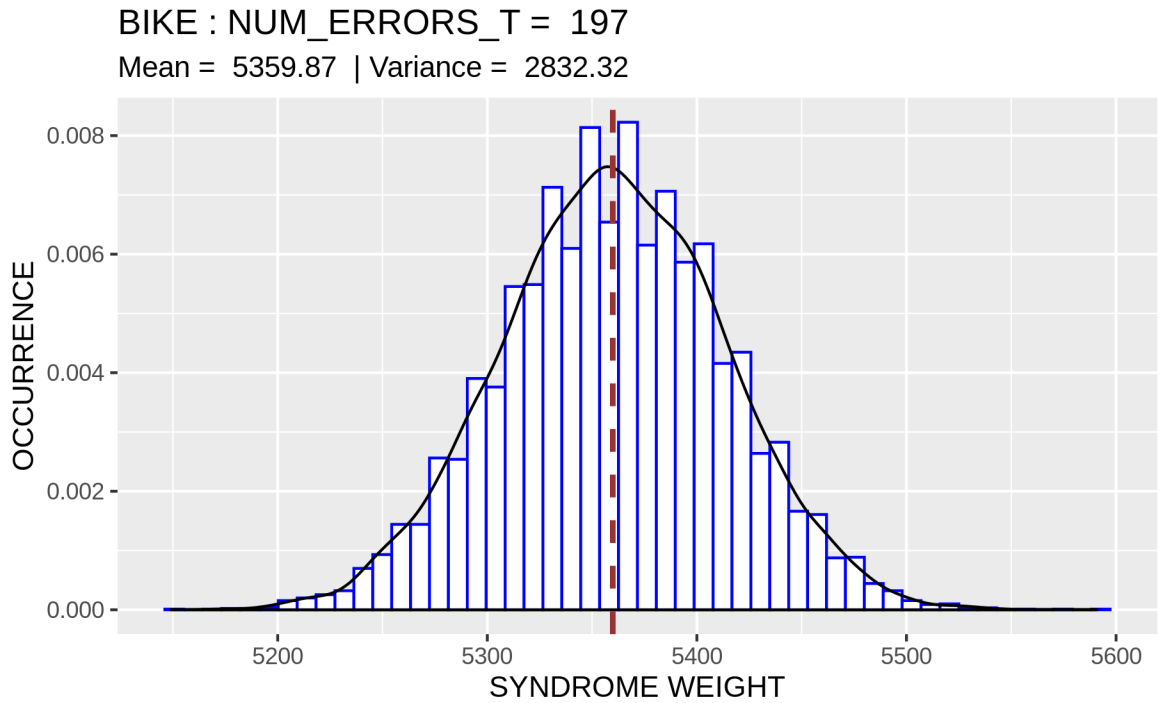

Figura 12: BIKE v3 : Distribuzione statistica del peso della sindrome quando t = 197.

Analizzando questi risultati si può vedere che la distribuzione può essere approssimativamente come di tipo normale. Si vede chiaramente come il peso medio della sindrome cresca al crescere di t, passando da 3581.57 a 5359.87 (+49,65 %) per t da 77 a 197. Per la configurazione proposta dalla specifica (t = 137) la media è 4783.27.

L'effetto è rappresentato più chiaramente nella figura 13, nella quale è stato graficato il valore medio del peso della sindrome s al variare di t, calcolato mediando il risultato ottenuto per ciascuno dei 10000 test effettuati.

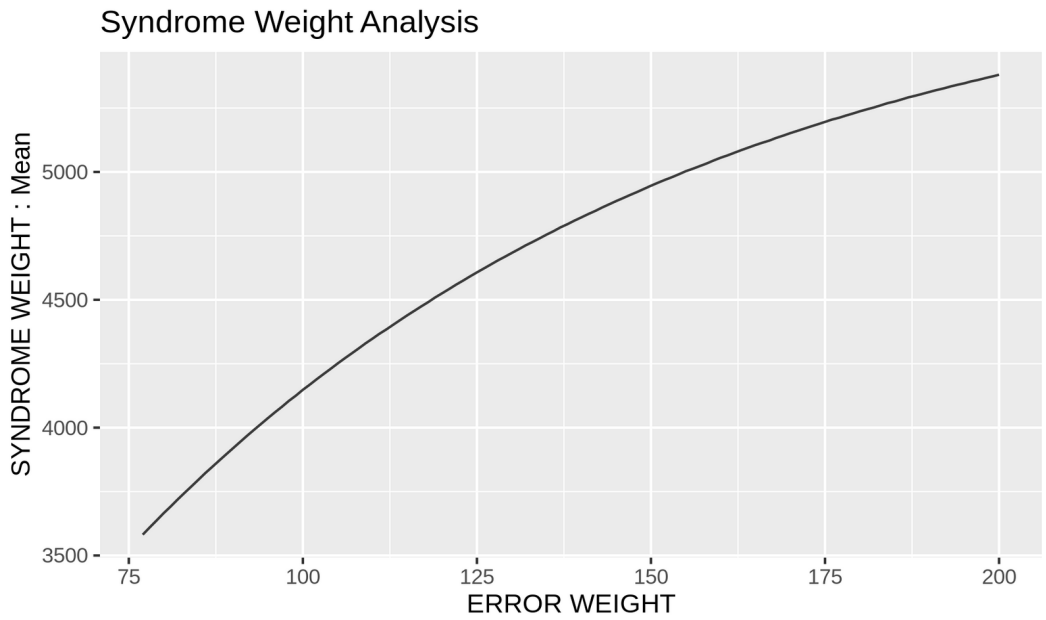

Figura 13: BIKE v3: Peso medio della sindrome graficato rispetto a t.

Può essere studiato similmente anche l'andamento della deviazione standard, per analizzare se ci sono differenze nel tipo di distribuzione oltre che nei valori.

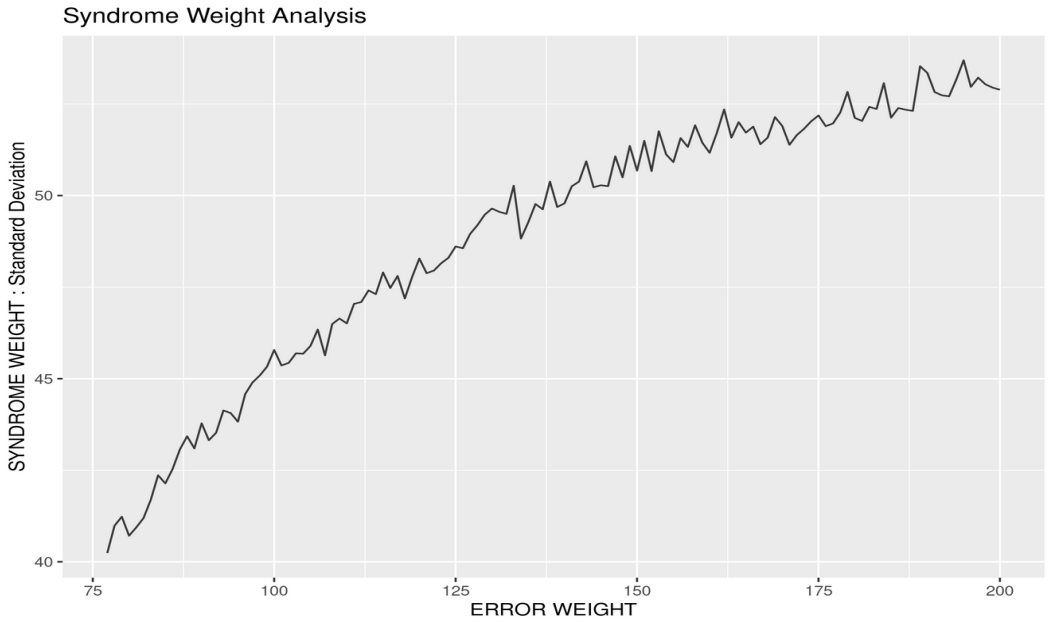

Figura 14: BIKE v3: Deviazione standard del peso della sindrome graficata rispetto a t.

La deviazione standard cresce al crescere di **t** così come il valore medio, con un andamento dello stesso tipo. Essa aumenta significativamente, ma una analisi più attenta rivela che essa diminuisce leggermente quando messa in rapporto con la media, anche se in maniera poco significativa: -10% per t da 75 a 200.

Nella figura 15 viene riportato una curva data dal rapporto tra il valore medio e la deviazione standard.

Figura 15: BIKE v3: Rapporto tra varianza e peso medio della sindrome graficato rispetto a t.

5 SICUREZZA E ANALISI DELLA DFR

La DFR (Decoding Failure Rate) di un codice è la probabilità che, usando un dato decoder e per un messaggio qualsiasi, il processo di decodifica non riesca a riottenere il messaggio originale che era stato trasmesso.

Il valore della DFR è una delle principali criticità da valutare affinché si possano usare codici sparsi in ambito crittografico. Infatti essa è strettamente collegata al livello di sicurezza del crittosistema. Le specifiche riguardo la DFR diventano estremamente stringenti se lo scopo è quello di garantire sicurezza post-quantum, al punto tale da richiedere un numero di simulazioni eccessivamente grande per poter essere effettivamente simulato con le attuali potenze di calcolo a disposizione.
Il problema studiato in questo lavoro è stato di trovare un approccio che rendesse possibile stimare un valore limite per la DFR dei sistemi crittografici.

5.1 Obiettivi di sicurezza

Lo sforzo computazionale che deve essere necessario per violare un crittosistema che possa essere considerato sicuro, da avere come riferimento per la progettazione dei parametri per i sistemi crittografici post-quantistici, è pari a quello richiesto su un computer classico o un computer quantistico per rompere l'AES con una dimensione di chiave di λ bit, $\lambda \in \{128, 192, 256\}$, attraverso un ricerca esaustiva delle chiavi.

L' AES (*Advanced Encryption Standard*) è una specifica per la crittografia dei dati elettronici stabilita dal NIST nel 2001.

Gli sforzi richiesti su un computer classico e quantistico corrispondono rispettivamente alle categorie Security Level (*SL*) 1, 3 e 5 del NIST.

Affinché si possa affermare che un crittosistema ha un livello di sicurezza pari a λ, è necessario provare che il valore della DFR è inferiore a $2^{-\lambda}$.

5.2 Sicurezza IND-CPA e IND-CCA

Le definizioni più comuni usate in crittografia per la sicurezza si declinano in base ai tipi di attacchi da cui un crittosistema riesce a difendersi.

- Sicurezza IND-CPA : Indistinguishability Under Chosen Plaintext Attack.
- Sicurezza IND-CCA1 : Indistinguishability Under non-adaptive Chosen Ciphertext Attack.
- Sicurezza IND-CCA2 : Indistinguishability Under adaptive Chosen Ciphertext Attack .

Questi livelli di sicurezza possono essere definiti dal seguente gioco tra un avversario e uno sfidante. Uno schema è sicuro IND-CCA1/IND-CCA2 se nessun avversario ha un vantaggio non trascurabile nel vincere la sfida di seguito.

1. Lo sfidante genera una coppia di chiavi PK, SK basata su alcuni parametri di sicurezza k (ad esempio, la dimensione della chiave in bit), e pubblica PK all'avversario. Lo sfidante tiene per sé SK.
2. L'avversario può eseguire qualsiasi numero di cifrature, chiamate all'oracolo di decifratura basate su cifratti arbitrari, o altre operazioni.
3. Alla fine, l'avversario presenta due distinti plaintex scelti M_0 ed M_1 allo sfidante.
4. Lo sfidante seleziona un bit $b \in \{0, 1\}$ in modo uniforme a caso, e invia il testo cifrato "sfida" $C = E(PK, M_b)$ all'avversario.
5. L'avversario è libero di eseguire qualsiasi numero di calcoli o cifrature aggiuntive.
   (a) Nel caso non adattivo (IND-CCA1), l'avversario non può effettuare ulteriori chiamate all'oracolo di decrittazione.
   (b) Nel caso adattivo (IND-CCA2), l'avversario può effettuare ulteriori chiamate all'oracolo di decifratura, ma non può presentare la sfida cifrata C.
6. Infine, l'avversario invia un'ipotesi per il valore di b.

Uno schema è invece sicuro IND-CPA se un avversario non ha alcun vantaggio (o ne ha uno trascurabile) nel vincere la partita di cui sopra, quando non sono presenti i punti 5.a e 5.b. In pratica l'indistinguibilità in caso di attacco non adattivo e adattivo del testo cifrato scelto (IND-CCA1, IND-CCA2) utilizza una definizione simile a quella di IND-CPA. La differenza è che oltre alla chiave pubblica (o all'oracolo di cifratura, nel caso simmetrico), l'avversario ha accesso ad un oracolo di decifratura che decifra i testi cifrati arbitrari su richiesta dell'avversario, restituendo il testo in chiaro. Nella definizione non adattiva, l'avversario può interrogare questo oracolo solo fino a quando non riceve il testo in chiaro. Nella definizione adattiva, l'avversario può continuare ad interrogare l'oracolo di decodifica anche dopo aver ricevuto il testo cifrato della sfida, con l'eccezione che non può passare il testo cifrato della sfida per la decodifica (altrimenti la definizione sarebbe banale).

5.3 Stima della DFR tramite estrapolazione

La sicurezza IND-CCA di BIKE si basa sull'assunzione di poter ottenere una stima della DFR tramite estrapolazione, dato che non può essere stimata direttamente tramite simulazioni. La metodologia usata dagli autori di BIKE è descritta di seguito [7].

In primo luogo, la DFR é stata misurata per valori piccoli di r, che è la dimensione dei blocchi delle matrici circolanti del codice, per le quali le simulazioni forniscono una stima affidabile.

Successivamente si è definita una curva sulla base dei valori acquisiti di r-DFR. La curva è stata poi estrapolata a valori più grandi della dimensione del blocco, in modo che la DFR estrapolata avesse l'ordine di grandezza desiderato.

Dato che i modelli asintotici per varianti più semplici del bit-flipping prevedono un andamento concavo per la curva nel relativo campo di valori r, si è assunto anche qui un comportamento simile, per il quale un'estrapolazione lineare su due punti simulati porta ad un stima conservativa della r richiesta.

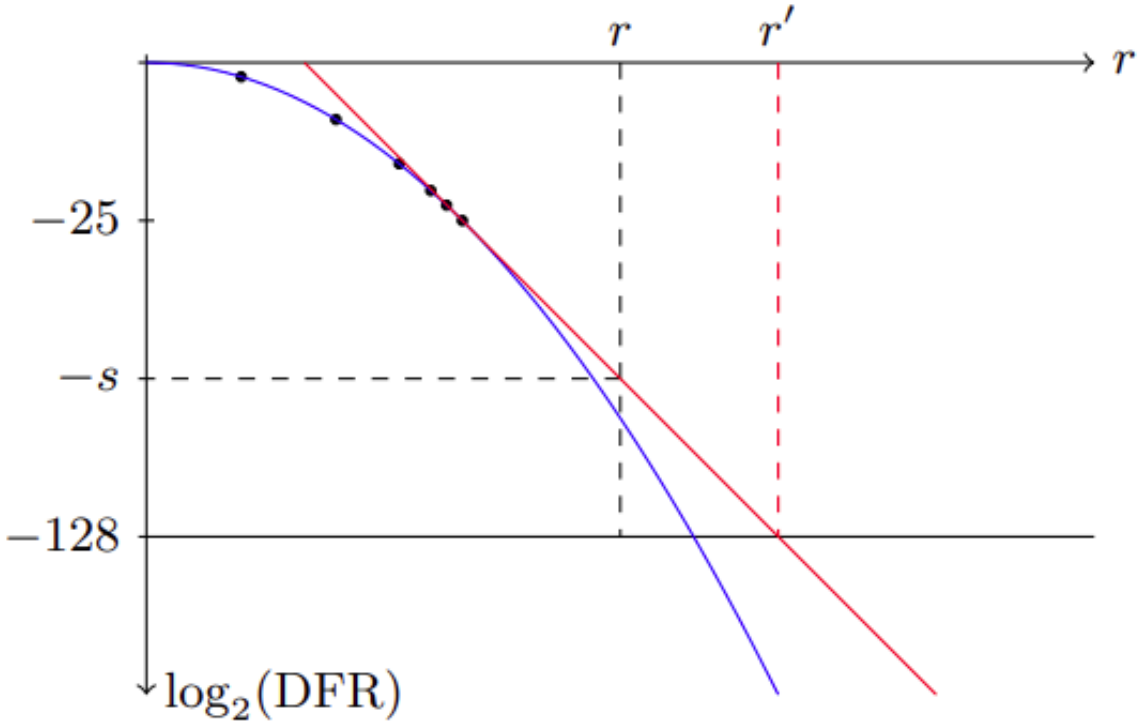

Figura 16: Andamento della DFR ottenuto tramite estrapolazione.

Osservando il grafico del logaritmo della DFR simulata rispetto alla dimensione del blocco r (gli altri parametri w e t sono fissi), si osserva che l'andamento è sempre concavo. Perciò viene supposto che rimanga tale e la DFR è estrapolata di conseguenza. La strategia consiste quindi nel realizzare una simulazione per il più grande r possibile per misurare con precisione la tangente del punto più basso possibile della curva. Ad esempio, nella figura 16, la curva più in basso (in blu) è il logaritmo in base 2 della DFR e che si riesce ad effettuare una simulazione accurata finché la DFR è superiore a 2^{-25} (punti neri). Prendendo la tangente nell'ultimo punto si ottiene la linea rossa da cui si ricava un limite superiore per r per una dimensione del blocco con una DFR inferiore a 2^{-128} e un limite superiore per la DFR pari a 2^{-s} per una data dimensione del blocco r.

Questa assunzione è tuttavia forzata, perché la relazione tra un dato valore di r e la rispettiva DFR non può essere concava per tutti i possibili $r \in [0, \infty[$.

Uno studio DFR così fatto differisce da quello che viene fatto per i sistemi di comunicazione, in cui il codice è fisso e il rapporto segnale/rumore aumenta (cioè la probabilità di errore per bit diminuisce).

Ci si aspetta di osservare lo stesso tipo di comportamento per la DFR del codice QC-MDPC quando si fissano (w, t) e si lascia crescere r. Alcune classi di codici a correzione d'errore, cioè i codici turbo e i codici LDPC (ai quali i MDPC sono affini) soffrono di un fenomeno noto come *error floor*. Il logaritmo della DFR segue una curva che è prima concava ed in seguito rapidamente decrescente (*waterfall*). Da un certo punto in la concavità cambia e la DFR diminuisce molto più lentamente, e ciò è noto come *error floor*.

Gli *error floor* di solito si verificano per valori molto bassi delle curve di DFR; tuttavia l'analisi seguente stima la DFR solo in relazione a particolari casistiche (*Near-Codewords* e *Low Weight Codewords*) e viene concluso che sarebbe opportuno studiare più approfonditamente questo aspetto.

In questo lavoro ci si ripropone di studiare invece casistiche particolari da valutare in base all'algoritmo di crittografia di cui si vuole studiare la sicurezza. Un sistema crittografico, pur basandosi su codici QC-MDPC, presenta delle elaborazioni complesse, e ci possono essere svariati casi particolari che diano origine a situazioni che pregiudichino la sicurezza, la cui probabilità di occorrenza pur essendo esigua potrebbe essere sufficiente a portare ad un *floor* della DFR per i valori di r che si intenderebbe utilizzare.

In un altro articolo [8], sono stati analizzati e comparati vari tipi di decoder (B, BG, BGB e BGF) seguendo due criteri:

1. La DFR per un dato numero di iterazione e un particolare valore di r.

2. Il valore di r necessario per ottenere la DFR desiderata con un dato numero di iterazioni.

Per stimare la DFR è stato usato un metodo di estrapolazione, ed in particolare due tipi di estrapolazione: “best linear fit” e “two large r’s fit”.

Gli autori fanno notare come tali tecniche di estrapolazione si basano sull'assunto che la dipendenza della DFR dalla grandezza del blocco r sia una funzione concava quando r assume valori nel range di interesse.

			Best linear fit				Two large r's fit			
Decoder	#I	#S	DFR = 2^{-23}	2^{-64}	2^{-128}	DFR at 11,779	DFR = 2^{-23}	2^{-64}	2^{-128}	DFR at 11,779
BG	3	9	10,253	11,213	12,739	2^{-88}	10,253	11,171	12,619	2^{-90}
	4	12	10,163	11,003	12,347	2^{-100}	10,163	10,909	12,107	2^{-110}
	5	15	10,133	10,909	12,107	2^{-111}	10,133	10,853	11,987	2^{-116}
BGB	4	9	10,253	11,093	12,491	2^{-95}	10,253	11,083	12,491	2^{-96}
	5	11	10,163	10,973	12,227	2^{-105}	10,163	11,027	12,413	2^{-99}
	6	13	10,133	10,973	12,269	2^{-104}	10,133	10,949	12,197	2^{-107}
BGF	5	7	10,301	11,171	12,539	2^{-92}	10,301	11,131	12,491	2^{-95}
	6	8	10,253	11,027	12,277	2^{-102}	10,253	10,973	12,197	2^{-107}
	7	9	10,181	10,949	12,149	2^{-108}	10,181	10,949	12,107	2^{-112}
B	4	8	10,259	11,699	13,901	2^{-67}	10,301	11,813	14,221	2^{-63}
	5	10	10,133	11,437	13,229	2^{-79}	10,133	11,437	13,451	2^{-76}
	6	12	10,067	11,213	13,037	2^{-84}	10,067	11,437	13,397	2^{-78}

Tabella 4: Analisi della DFR per vari tipi di decoder, seguendo l'analisi proposta in [8].

La Tabella 4 riassume i risultati. Essa mostra quale valore di r sia necessario ad avere una DFR pari a $2^{-23}(\approx 10^{-8})$, $2^{-64,}$ e 2^{-128}.

Inoltre presenta una stima della DFR per r = 11779 , che è il valore usato per BIKE-1 CCA (v3) quando SL=1.

Le informazioni complete sugli esperimenti e le tecniche di estrapolazione possono essere trovate nel paper di riferimento [8].

5.4 DFR: Metodologia Generale

L'analisi proposta in questo lavoro è un esempio di come operare nei casi in cui si voglia analizzare la sicurezza di sistemi che richiedono una DFR così bassa da non poter essere stimata direttamente, neanche con mezzi potenti ed analisi prolungate. Si vuole dimostrare che per la DFR si può stimare un valore limite in maniera indiretta e che ciò può essere di per sé sufficiente a stabilire se un dato sistema crittografico in esame soddisfi i requisiti per avere un determinato livello di sicurezza o meno. Si noti che qualora l'analisi dovesse rivelare che il valore limite è inferiore a quello necessario, ciò non sarebbe sufficiente a trarre alcuna conclusione.

Per fare ciò si è voluto utilizzare un approccio diverso al problema della stima della DFR. Dato che stimarla direttamente è impossibile a causa dei valori estremamente bassi desiderati (che implicano un enorme numero di test) si andrà a stimare quale sia la probabilità di errore in decodifica per alcuni particolari vettori "deboli". Cosa voglia dire "deboli" dipende dal sistema crittografico che si sta studiando, e deve essere valutato preliminarmente sulla base della struttura dell'algoritmo. Bisogna quindi analizzare quali siano le variabili del sistema che non sono deterministiche, come le chiavi o il messaggio, ed analizzare i possibili casi specifichi per cui è noto che il sistema possa fallire la decodifica. E' possibile riferirsi alle letteratura riguardante i codici su cui il sistema si basa (come verrà fatto nel capitolo 7) o studiare il problema autonomamente (capitolo 8). In questo consiste la prima fase del lavoro, che può dirsi terminata una volta trovato un insieme di casi particolari che si pensa possano portare ad un peggioramento della DFR.

Si possono quindi studiare le proprietà del sistema crittografico in quel particolare caso, ritenuto sfortunato, o in relazione alla variabile che rende il vettore "debole". Lo si può fare andando ad alterare il codice dell'algoritmo in modo che la generazione non sia casuale ma portata a generare esattamente valori che rientrano nella casistica che si sta studiando. Così facendo è sufficiente un numero di test molto minore per stimare la DFR, perché se le assunzioni di partenza sono corrette quest'ultima sarà auspicabilmente abbastanza alta da poter essere simulata in tempi ragionevoli. Ad esempio nelle simulazioni riportate in

seguito in questo lavoro, si è scelto di effettuare 10000 test, riuscendo a valutare in quali casi la DFR è maggiore o uguale di 10^{-4}.

La stima è sempre valida se il fine ultimo è l'analisi della sicurezza perché il valore così ottenuto è cautelativo, in quanto in tutti quei casi in cui non si riveleranno errori la DFR verrà considerata 0, anche se è chiaro che il valore reale è più alto; invece nei casi in cui la DFR risulti non nulla, anche un numero di test relativamente basso fornisce una stima abbastanza accurata che non si discosta molto dal valore reale. In ogni caso, qualora una volta completata l'analisi si voglia maggiore precisione, è sempre possibile svolgere delle simulazioni più accurate.

Dopo aver stimato la DFR per tutte le casistiche che era di interesse analizzare, è sufficiente trovare la probabilità con cui queste si verificano.

La rispettiva probabilità di occorrenza può essere calcolata matematicamente in modo esatto, dato che si suppone che le variabili siano generate casualmente e uniformemente.

Si riesce quindi ad ottenere un valore limite per la DFR effettuando una media pesata tra la DFR stimata per ogni casistica e la probabilità che venga generato un vettore "debole" che appartiene a quella particolare casistica.

Si tratta di un valore limite perché per tutti i vettori che non sono stati considerati può accadere di avere ugualmente degli errori. Dato che l'obiettivo di una analisi di questo tipo è valutare la sicurezza del sistema, non è necessario avere una stima della DFR estremamente accurata: E' sufficiente provare che tale valore per la DFR è superiore a $2^{-\lambda}$ per negare l'assunzione di sicurezza di livello λ.

La metodologia è valida, generale e può essere applicata a qualunque tipo di algoritmo la cui sicurezza sia legata alla DFR, studiandone volta per volta le criticità ed analizzando i casi particolari che possono portare difficoltà in decodifica. Partendo da questi dati e a patto di conoscere la probabilità che tali condizioni si verifichino, la stima che si ottiene è accurata, perché si basa su assunzioni logiche e su simulazioni che essendo molto specifiche richiedono un numero di test ragionevole per essere statisticamente significative.

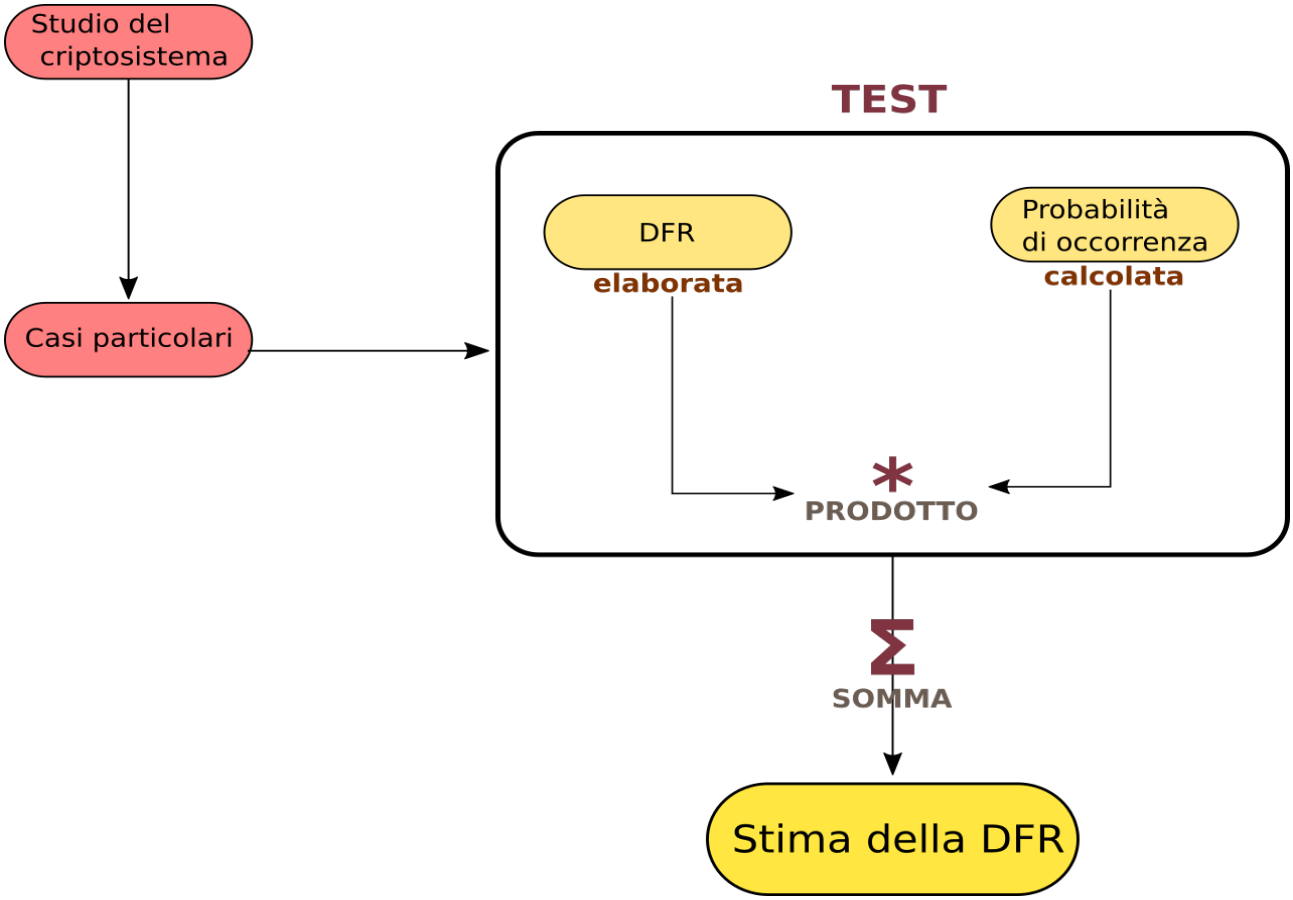

Figura 17: Schema riassuntivo dell'algoritmo qui proposto per la stima del DFR.

Nel seguito della trattazione verrà adottata questa metolodogia per valutare la sicurezza di alcuni sistemi proposti per la gara Post-Quantum Cryptography Standardization del NIST. Si studieranno due particolari tipi di chiavi "deboli" relativamente ai sistemi LEDAcrypt e BIKE e si potranno trarre conclusioni sulla loro sicurezza, dimostrando come la procedura proposta in questo lavoro possa essere efficiente per dimostrare che le ipotesi sulla DFR ottenuta per estrapolazione non siano sempre valide.

5.5 Esperimenti ed elaborazione

Di seguito viene esposta la metodologia attraverso la quale si sono effettuati i test fatti per ottenere i risultati qui riportati. Per ogni caso di studio sono stati scritti dei programmi particolari, ma il modus operandi generale è comune. La procedura è stata adattata di volta in volta al codice del crittosistema originale e alle variabili che si desiderava analizzare.

Data la complessità dei codici C originali ed il loro alto grado di ottimizzazione, si è scelto di mantenere come MACRO (direttiva `DEFINE` in C) quelle variabili che erano così definite, in modo da minimizzare le modifiche al codice per avere maggior efficienza e sicurezza. Così facendo si ha la necessità di dover cambiare e compilare nuovamente il codice ad ogni diverso test che si vuole effettuare. Per automatizzare la procedura è stato scritto del codice complementare in linguaggio R, in ambiente di sviluppo RStudio.

Il codice R prende come ingresso il codice C e modifica la MACRO di interesse; esso legge il file C come testo e ricerca la riga dove è definito il parametro in esame, e quindi lo modifica. Allo stesso modo modifica anche le variabili aggiuntive, nei test dove sono presenti.

Il criterio con cui modificare questi fattori è gestito in maniera intelligente da un ulteriore script R, in modo da ottimizzare l'impiego delle risorse e i tempi necessari, e segue l'algoritmo di seguito:

- Si inizializzano le variabili da studiare.
- Il test consiste in numero massimo di 10000 iterazioni, tuttavia si blocca automaticamente al raggiungimento di 50 errori in decodifica, in quanto tale valore viene comunemente accettato come abbastanza valido per stimare correttamente la DFR.
- I risultati ottenuti sono scritti un file (ad esempio .txt o .csv). Si verifica quindi l'esito del test e si aggiornano i parametri con un algoritmo intelligente. Una volta individuato il range di valori per cui la DFR ha un ordine grandezza compatibile con i tempi a disposizione, si può andare ad approfondire il suo andamento per quei particolari valori.

- Il codice originale del test è stato invece modificato così da scrivere su un file i dati significativi dell'elaborazione effettuata, che sono stati poi analizzati e graficati. I grafici sono stati ottenuti sempre tramite RStudio, in particolare facendo uso della libreria ggplot2.

Inoltre per far sì che programmi scritti in linguaggi diversi fossero eseguiti automaticamente sono stati scritti degli script bash (.sh) . In questo modo le prove sono state eseguite iterativamente.

Il codice è stato eseguito su un computer ASUS K550J (con Intel Core i7-4710HQ, 8 GB di RAM) e sistema operativo Linux Ubuntu 19.10 eoan.

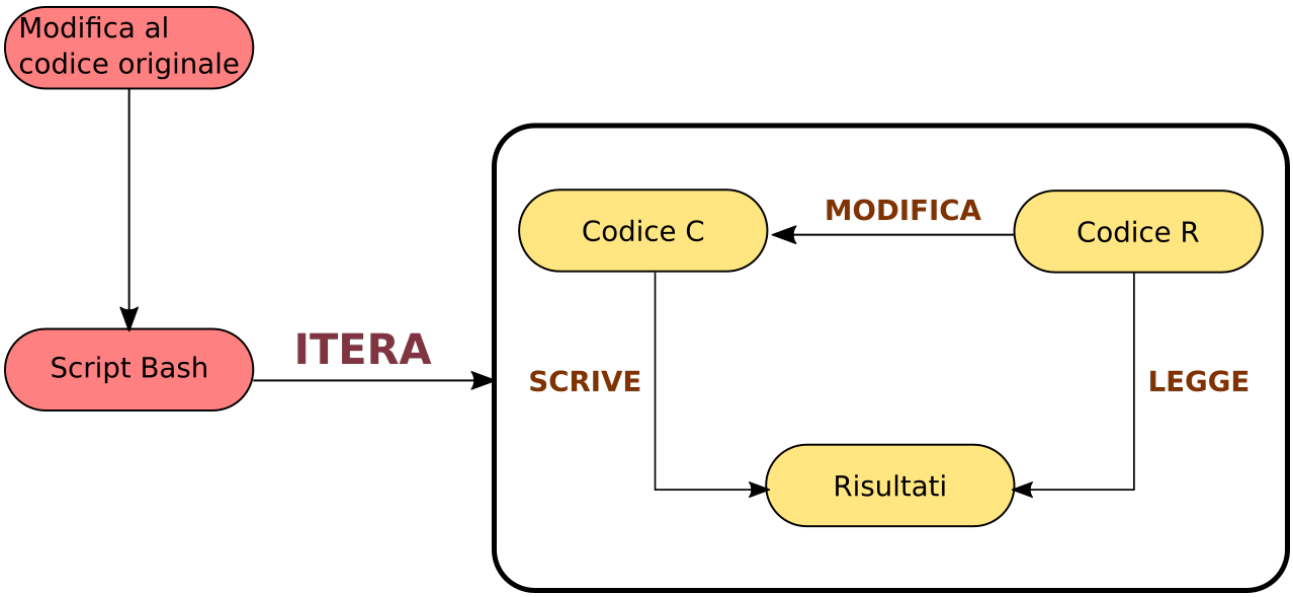

Figura 18: Schema riassuntivo dell'organizzazione usata per svolgere i test in maniera efficiente.

6 CASO DI STUDIO 1 : ANALISI DI MATRICI H DEBOLI

Nei sistemi di crittografia basati su codici sparsi, alcune scelte della matrice **H** portano alla generazione di chiavi deboli.

Quando le matrici circolanti che compongo **H** hanno sequenze di elementi 1 troppo lunghe, la decodifica può non essere efficace a ricostruire il messaggio originario.

In questo capitolo è stato studiato l'effetto di chiavi deboli modificando il codice C che genera **H**. Si sono svolti test riferiti a LEDAcrypt e a BIKE v3.

La posizione degli elementi 1 è stata impostata in modo che non fosse in prima istanza casuale; un certo numero di 1 consecutivi vengono imposti a priori. Questo valore è chiamato N_FIXED_ONES.

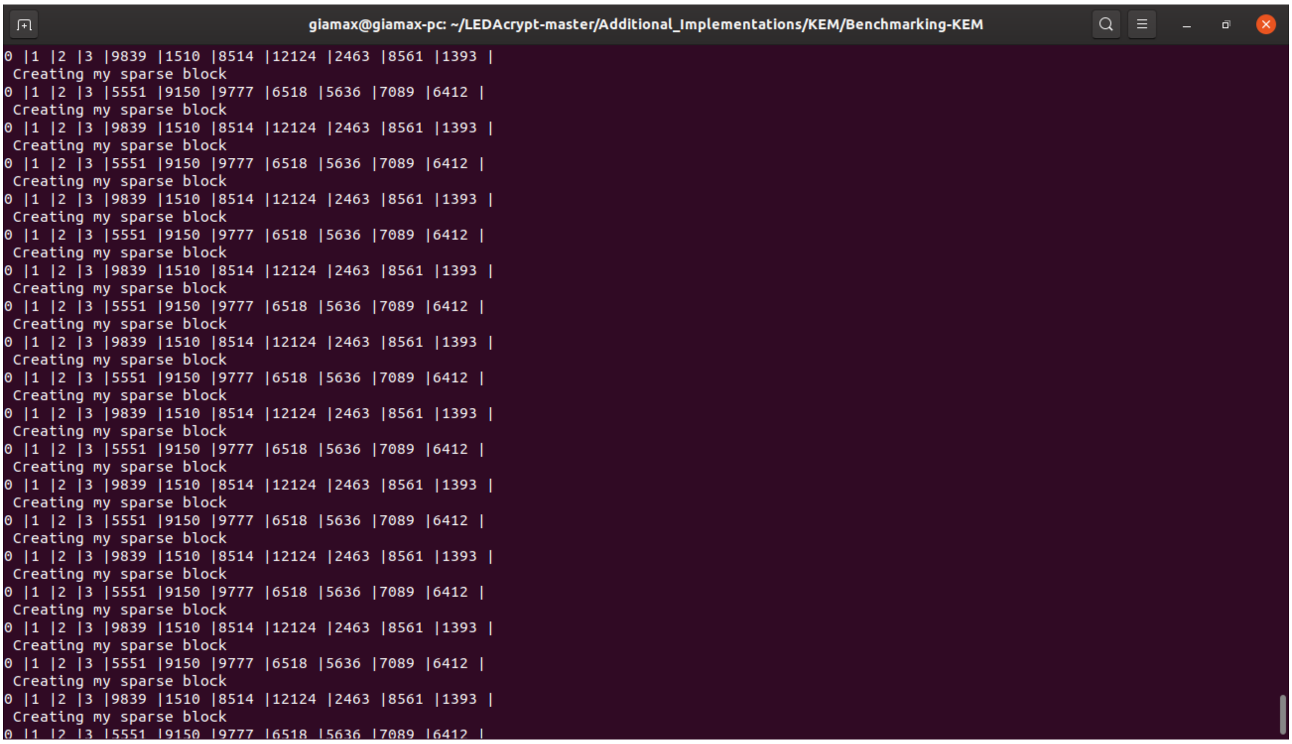

Figura 19: La finestra di comanda mentre viene eseguito un test di questo tipo. In questo esempio vengono stampate a video le posizioni degli elementi non-nulli. In questo caso N_FIXED_ONES era pari a 4.

Fissato il numero di elementi 1, si vedrà che aumentando N_FIXED_ONES la DFR (Decoding Failure Rate) aumenta. Questo risultato coincide con quanto atteso. Ciò è dovuto al fatto che serie lunghe di 1 rendono il processo di decodifica difficoltoso, perché le righe della matrice circolante, essendo copie shiftate della riga che presenta la sequenza, vengono ad avere un gran numero di cicli, di cui molti anche di lunghezza minima (4), e come è noto (vedasi [12]) i cicli sono uno degli elementi cruciali da evitare per l'efficienza degli algoritmi di decodifica. L'effetto diventa molto più pronunciato quando il numero di 1 consecutivi è grande.

Tuttavia bisogna considerare che la probabilità che una sequenza di una certa lunghezza si verifichi casualmente quando viene generata **H** scegliendo un valore qualsiasi tra i possibili, diminuisce di molto all'aumentare di N_FIXED_ONES. Quindi anche se la DFR aumenta di molto in quel particolare caso, l'effetto sulla DFR in generale potrebbe non essere significativo.

Essendo questo test uno studio preliminare, si è voluto avere una analisi più comprensiva e generale. Perciò si è studiato l'effetto di queste chiavi deboli anche al variare di t, che indica il peso del vettore di errore, e che è definito del codice dalla variabile chiamata NUM_ERRORS_T.

Questo tipo di analisi non è servita a valutare la sicurezza del codice, dato che il valore di t è in realtà fisso e definito nello standard dei crittosistemi, ma ad avere una comprensione più approfondita degli effetti di questo tipo di chiavi deboli. Può essere utile in futuro per la progettazione dei futuri algoritmi o essere ulteriormente analizzato per concepire attacchi basati su questa particolarità.

6.1 LEDAcrypt

Per studiare l'effetto di chiavi deboli con lunghe sequenze di 1 in LEDAcrypt è stata modificato il codice che genera casualmente i blocchi circolanti che compongono la matrice **H**. Un numero pari alla variabile N_FIXED_ONES di 1 consecutivi viene imposto a priori, mentre le altre posizioni dei NUM_ERRORS_T - N_FIXED_ONES vengono scelti randomicamente . Per ottenere questo risultato è stato modificato il codice C di LEDAcrypt, come mostrato nell'immagine di seguito. Per ulteriori informazioni si rimanda all'appendice (capitolo 9.1).

```
void mod_circulant_sparse_block(POSITION_T *pos_ones,
                                const int countOnes,
                                AES_XOF_struct *seed_expander_ctx, int N_FIXED_ONES)
{
   //printf("\n Creating my sparse block \n");
   int duplicated, placedOnes = 0;

   //First 'N_FIXED_ONES' ONES are not randomly chosen
      for (int j = 0; j < N_FIXED_ONES; j++){
         pos_ones[placedOnes] = j;
         //printf("%d |",j);
         placedOnes++;
   }

   //Other ONES are randomly chosen
   while (placedOnes < (countOnes)) {
      int p = rand_range(NUM_BITS_GF2X_ELEMENT,
                         BITS_TO_REPRESENT(P),
                         seed_expander_ctx);
      duplicated = 0;
      for (int j = 0; j < placedOnes; j++) if (pos_ones[j] == p) duplicated = 1;
      if (duplicated == 0) {
         pos_ones[placedOnes] = p;
         //printf("%d |",p);
         placedOnes++;
      }
   }
}
```

Figura 20: Codice della versione modificata di LEDAcrypt, usata per generare le chiavi con un certo numero di elementi 1 non casuali.

Si riporta a titolo d'esempio la procedura esatta utilizzata per svolgere i test, di cui una panoramica generale e completa è già stata fornita nel paragrafo 6.5.

Inizialmente il valore di NUM_ERRORS_T viene posto pari a 200 ed N_FIXED_ONES a 0.
Il test consiste in numero massimo di 10000 iterazioni, tuttavia si stoppa automaticamente al raggiungimento di 50 errori in decodifica, in quanto tale valore viene comunemente accettato come valido per stimare correttamente la DFR.

Si verifica quindi l'esito del test:

- Se la DFR è 1, si procede ad un nuovo test diminuendo NUM_ERRORS_T di 3.
- Se la DFR è compresa tra 0 e 1, si procede ad un nuovo test diminuendo NUM_ERRORS_T di 1.
- Se la DFR è 0, si riporta NUM_ERRORS_T al valore di default (200) incrementando però N_FIXED_ONES.

Il codice originale del test è stato invece modificato così da scrivere su un file i dati significativi dell'elaborazione effettuata, che sono stati poi analizzati e graficati. I risultati sono riportati nella prossima sezione.

6.1.1 LEDAcrypt : Risultati

In questa sezione verranno esaminati i grafici che racchiudono i risultati degli esperimenti effettuati, analizzando le effetto di chiavi deboli in LEDAcrypt con lunghe sequenze di elementi 1.

Sull'asse delle ascisse viene riportato il peso del vettore di errore, mentre in ordinata viene graficata la DFR simulata: ne viene riportato il logaritmo in base 10 in modo che sia ben visibile la dinamica del suo andamento. Il colore della linea infine codifica la lunghezza della sequenza.

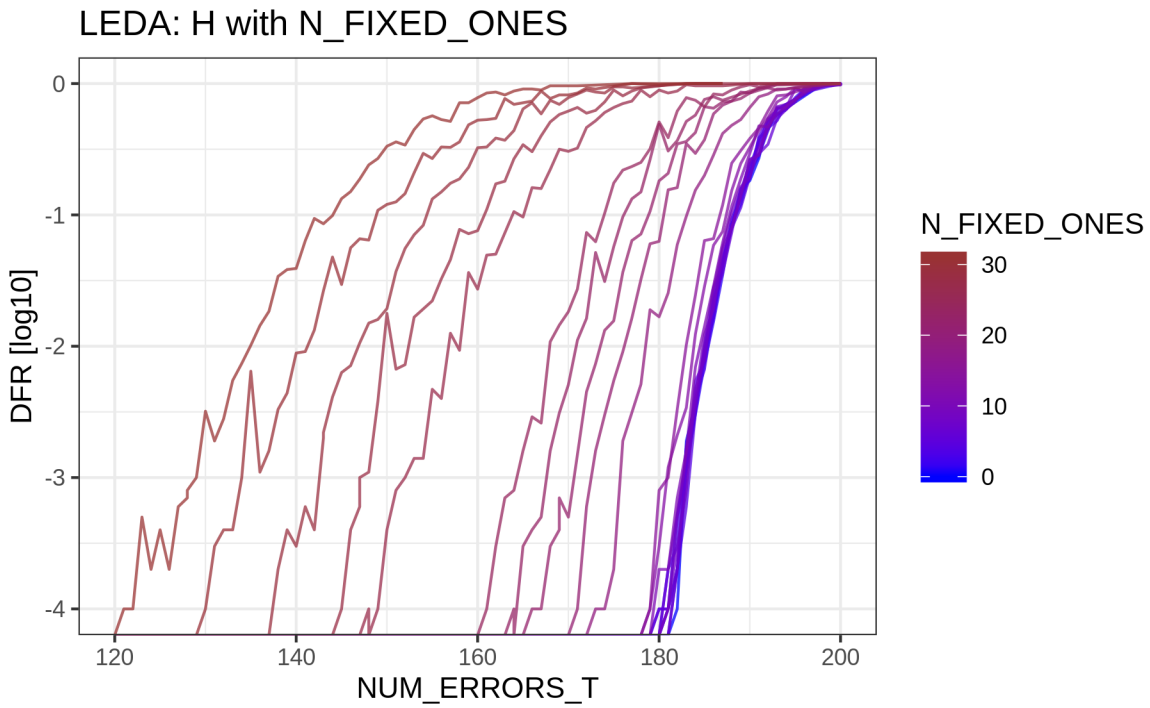

Figura 21: Andamento della DFR in funzione di NUM_ERRORS_Tper diversi valori di N_FIXED_ONES.

All'aumentare di N_FIXED_ONES la DFR aumenta, e tale effetto è tanto più pronunciato quanto NUM_ERRORS_T è piccolo. I dati confermano quindi le ipotesi fatte analizzando il caso con le sole conoscenze teoriche. Vettori in cui gli elemeni 1 sono concentrati maggiormente in una parte sola, e formano delle sequenze, sono effettivamente delle chiavi deboli che una volta scelte minano la sicurezza del crittosistema. Si potrebbero imporre dei controlli per far sì che tali casistiche vengano scartate, ma ciò limiterebbe lo spazio delle chiavi e questo ciò non è auspicabile.

Il grafico appare leggermente frastagliato, a cause del fatto che 10000 test non sono sufficienti per valori di DFR molto bassi. Perciò il prossimo grafico riporta le stesse curve, dopo una elaborazione che serve a smussarle.

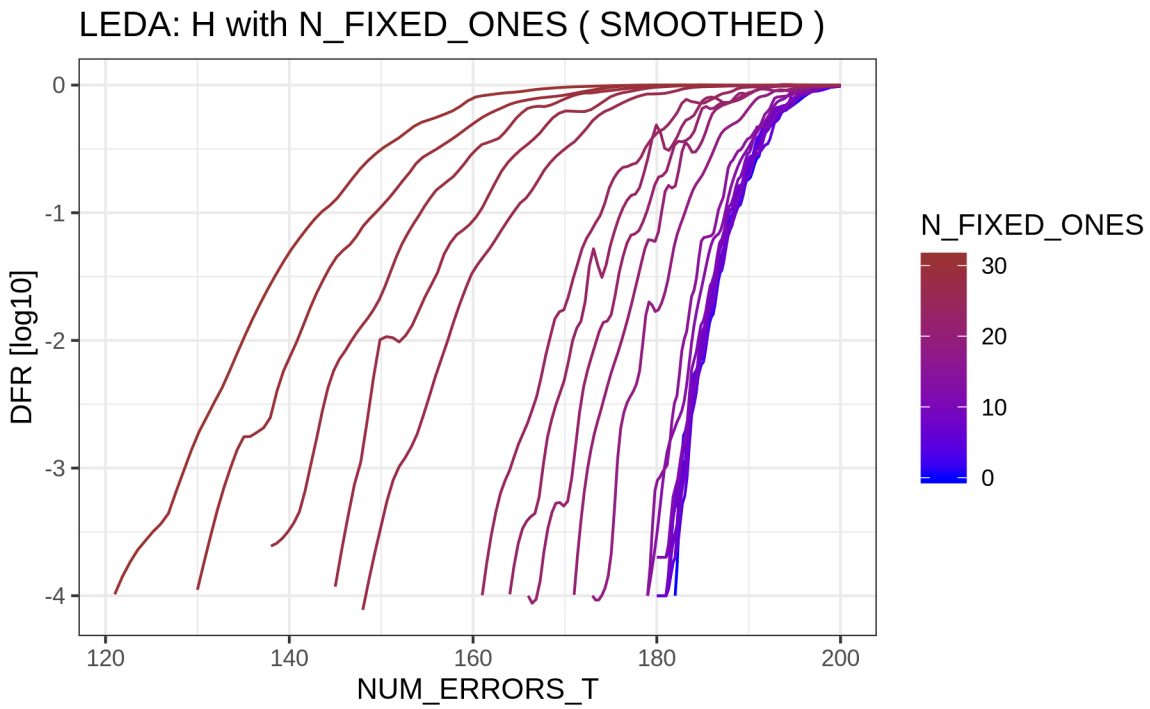

Figura 22: Andamento della DFR in funzione di NUM_ERRORS_T, per diversi valori di N_FIXED_ONES. Curve smussate.

Questi grafici sono molto compatti e riassumono un gran numero di informazioni. Può essere utile presentare gli stessi risultati riportati in grafici separati.

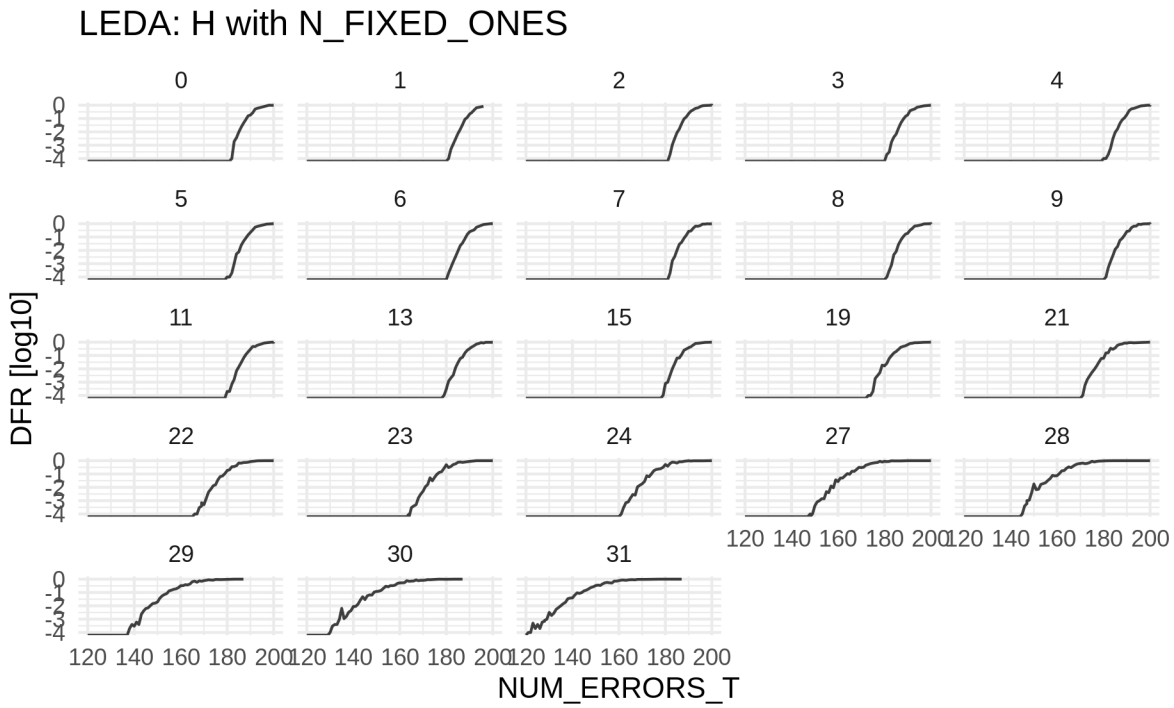

Figura 23: Andamento della DFR in funzione di NUM_ERRORS_T. Grafici presentati per diversi valori di N_FIXED_ONES.

All'aumentare di N_FIXED_ONES diminuisce il valore di NUM_ERRORS_T per cui si trovano errori in decodifica simulando 10000 test, stimano quindi una DFR pari a 10^{-4}. Ad esempio nel caso N_FIXED_ONES=5 ciò avviene per NUM_ERRORS_T=182, mentre per N_FIXED_ONES=19 ciò avviene per NUM_ERRORS_T=173.

Si nota che le linee corrispondente a valori N_FIXED_ONES poco elevati le linee sono sovrapposte. Si vuole evidenziare questo fenomeno nella prossima immagine, in cui si esaminerà il caso in cui ci siano 5 o meno elementi non-nulli consecutivi.

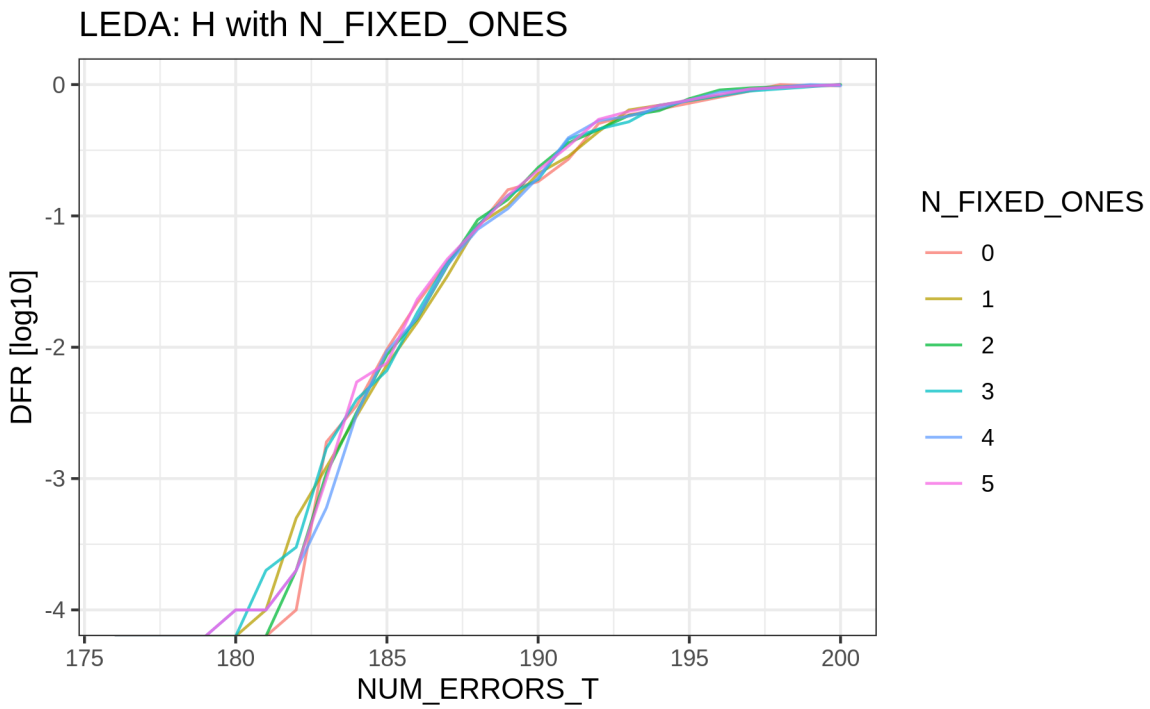

Figura 24: Andamento della DFR in funzione di NUM_ERRORS_T. Il colore rappresentata valori di N_FIXED_ONES da 0 a 5.

L'immagine conferma che sequenze di elementi 1 corte non costituiscono delle chiavi deboli. Ciò è congruente con quanto atteso; infatti questa casistica si verifica comunemente nella generazioni delle chiavi, perché ha una probabilità di verificarsi alta, ed in fase di progettazione è necessario prevedere un decoder che sia in grado di gestirla. Il caso `N_FIXED_ONES = 0` coincide con l'effettuare i test sul crittosistema originale.

Nelle prossime figure verranno mostrati gli stessi dati, presentati in maniera che sia più facile osservare quale sia l'andamento della DFR, così sono evidenziati solo alcuni valori a campione per N_FIXED_ONES.

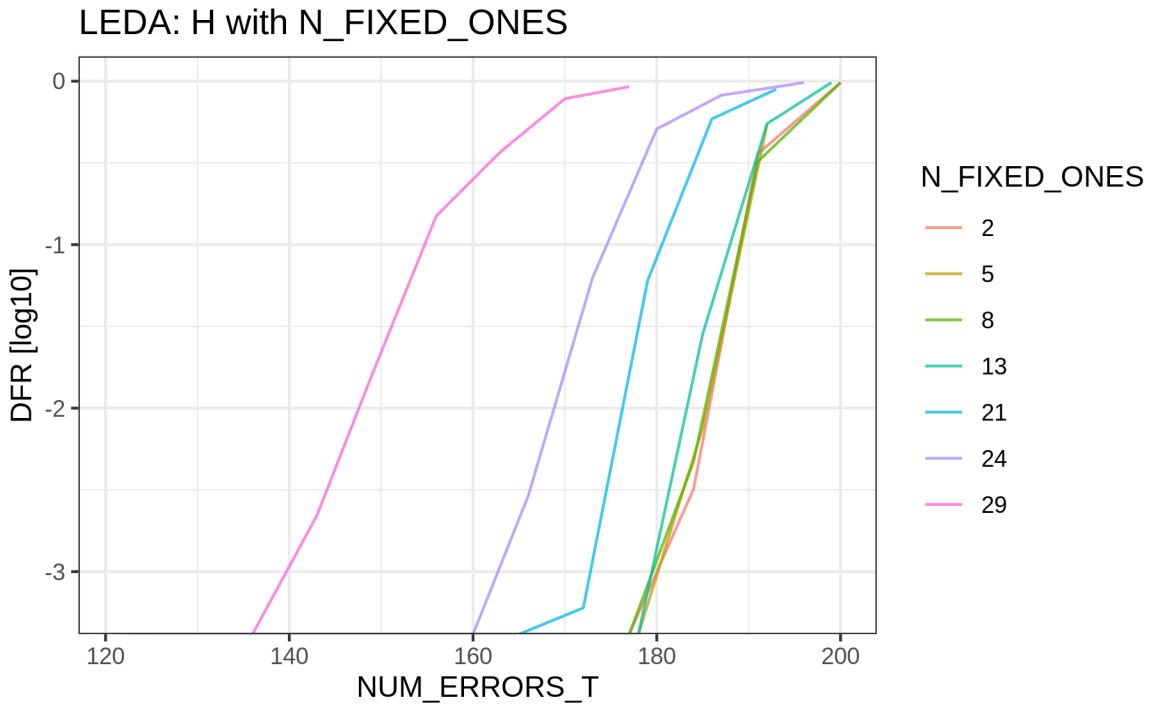

Figura 25: Andamento della DFR in funzione di NUM_ERRORS_T. Il colore rappresentata alcuni valori di N_FIXED_ONES, presi a campione.

Infine sono riportati i valori della DFR in forma tabellare. Per ogni valore di N_FIXED_ONES (nella prima colonna) viene riportato il minimo valore di t per cui la DFR diventa minore o uguale a 0.5 (seconda colonna) . La terza colonna corrisponde al valore minimo di t per cui in 10000 test non vengono mai rilevati errori.

N_FIXED_ONES	T:DFR==0.5	T:DFR==0
0	192	182
1	192	181
2	192	181
3	192	180
4	191	181
5	191	181
6	192	180
7	192	181
8	192	181
9	192	181

N_FIXED_ONES	T:DFR==0.5	T:DFR==0
11	192	179
13	191	179
15	191	179
19	188	174
21	185	171
22	184	167
23	182	164
24	181	161
27	172	149
28	167	145

Tabella 5: Tabella riassuntiva dei valori della DFR per diversi valori di N_FIXED_ONES.

L'insorgere di blocchi con sequenze di 1 lunghe in particolare 11 o più porta ad un peggioramento sensibile della DFR, e all'aumentare del valore le difficoltà in decodifica diventano progressivamente di più. Sequenze di 20 o più 1 portano ad una degradazione estrema delle prestazioni, tuttavia bisogna considerare che la generazione casuale di chiavi di questo tipo è estremamente rara.

Si può concludere che la generazione di chiavi deboli di questo tipo porta ad un peggioramento della DFR e potrebbe quindi essere un aspetto problematico per la sicurezza del crittosistema. Tuttavia queste chiavi capitano con una probabilità molto bassa per cui, anche se hanno una DFR molto alta, non costituiscono un problema in quanto la probabilità di prendere una chiave di questo tipo è molto bassa.

6.2 BIKE v3

E’ stato studiato l’impatto delle chiavi deboli nell’algoritmo BIKE-1 CCA in maniera del tutto analoga a quanto fatto per LEDAcrypt.

Le premesse teoriche sono le stesse, dato che i due sistemi sono fondamentalmente simili e si basano sulla stessa famiglia di codici. Ci si aspetta perciò di trovare risultati qualitativamente simili.

Ricordando che in BIKE la chiavi private è data dalla coppia di matrici sparse ($\mathbf{h}_0$, $\mathbf{h}_1$), si è tracciata la curva della DFR al variare di N_FIXED_ONES, cioè il numero di uni non generati casualmente in entrambe $\mathbf{h}_0$ e $\mathbf{h}_1$.

Si è dovuto modificare il codice sorgente di BIKE come mostrato nell'immagine di seguito. Ulteriori dettagli sono nel precedente capitolo 5.2.

```
ret_t
mod_generate_sparse_rep(OUT uint64_t *    a,
                        OUT idx_t         wlist[],
                        IN const uint32_t weight,
                        IN const uint32_t len,
                        IN const uint32_t padded_len,
                        IN OUT aes_ctr_prf_state_t *prf_state,
                       uint64_t N_FIXED_ONES)
{
  assert(padded_len % 64 == 0);
  // Bits comparison
  assert((padded_len * 8) >= len);

  uint64_t ctr = 0;

  //I PRIMI 1 SONO FISSI
  printf("N_FIXED_ONES = %ld \n", N_FIXED_ONES);
  do
  {

    //GUARD(get_rand_mod_len(&wlist[ctr], len, prf_state));
    wlist[ctr] = ctr;
    //printf("POS 1 = %d | ", wlist[ctr]);
    ctr += is_new(wlist, ctr);

  } while(ctr < N_FIXED_ONES); //weight

  printf("\n - END FIXED_ONES -> \n");

  // Generate weight rand numbers
  do
  {
    GUARD(get_rand_mod_len(&wlist[ctr], len, prf_state));

    //printf("POS 1 = %d | ", wlist[ctr]);

    ctr += is_new(wlist, ctr);
  } while(ctr < weight); //weight

  printf("\n \n");
  // Initialize to zero
  memset(a, 0, (len + 7) >> 3);

  // Assign values to "a"
  secure_set_bits(a, wlist, padded_len, weight);

  return SUCCESS;
}
```

Figura 26: Codice della versione modificata di BIKE, usata per generare le chiavi con un certo numero di elementi 1 non casuali.

6.2.1 BIKE v3 : Risultati

Di seguito saranno presentati i risultati dei test effettuati analizzando le effetto di chavi deboli in BIKE v3 con sequenze di elementi 1.

La metodologia usata è del tutto simile a quella usata per LEDAcrypt nel paragrafo precedente. I parametri dei test, gli orientamenti degli assi delle figura, e tutto quanto non diversamente specificato è congruente con quanto esposto in precedenza riferendosi a LEDAcrypt.

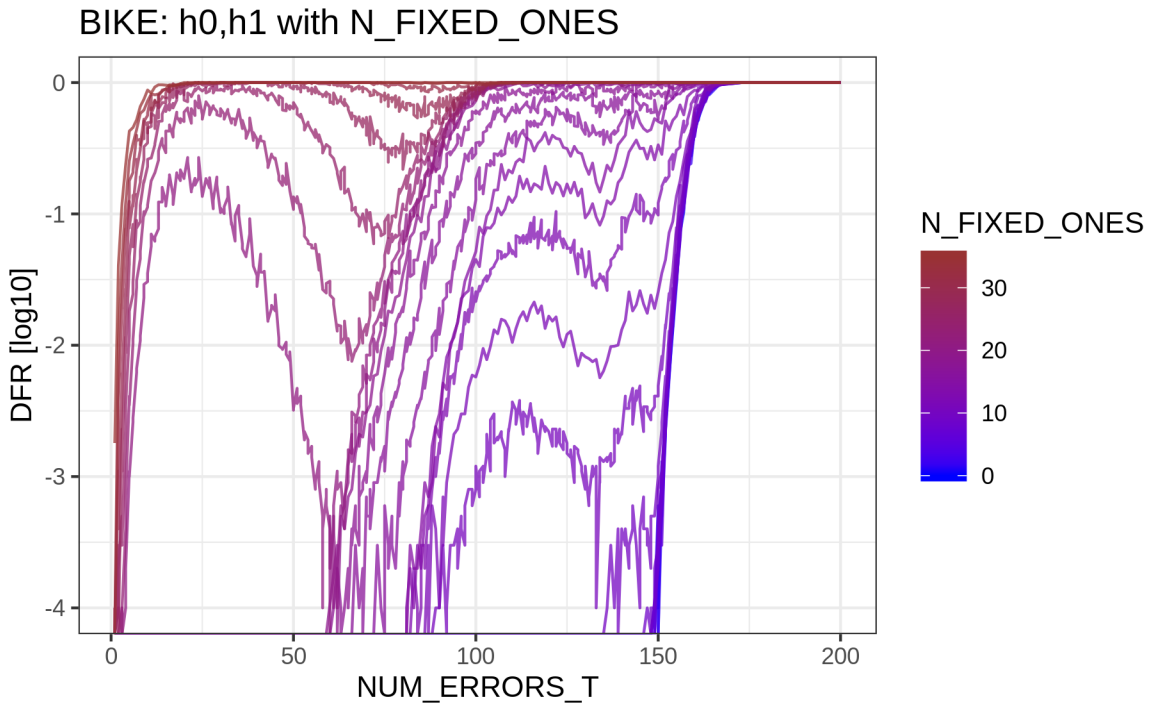

Figura 27: Andamento della DFR in funzione di NUM_ERRORS_T, per diversi valori di N_FIXED_ONES.

L'andamento di questi risultati può sembrare prima vista anomalo. Per valori di t compresi tra 20 e 25 le curve non hanno il caratteristico andamento *waterfall*. Il motivo di ciò è legato al funzionamento delle soglie del decodificatore usato, le quali sono state approfondite nel capitolo 4. Essendo soglie scelte appositamente per essere ottimizzate i particolari valori di t proposti per l'implementazione, si osserva un progressivo peggioramento delle prestazioni quando il valore di t in esame non è prossimo a questi ultimi.

Le soglie sono scelte per imitare il comportamento medio delle sindromi (e del loro peso in particolare) a `t = 134`. Modificando sensibilmente `t`, ma lasciando fisse le soglie, il decoder lavora male. Infatti, a `t` basso e con un alto numero di 1 consecutivi, quel che accade è che la sindrome ha un peso che, in media, è notevolmente basso. Questo porta ad una scelta della soglia bassissima, con la conseguenza che numerosissimi bit vengono flippati, anche se sono giusti.

A parte questo fenomeno le curve mostrano che, per un dato valore di t, all'aumentare di N_FIXED_ONES le chiavi diventano rapidamente "deboli" e portano ad un aumento della DFR. L'effetto è anche più pronunciato di quanto non fosse in LEDAcrypt.

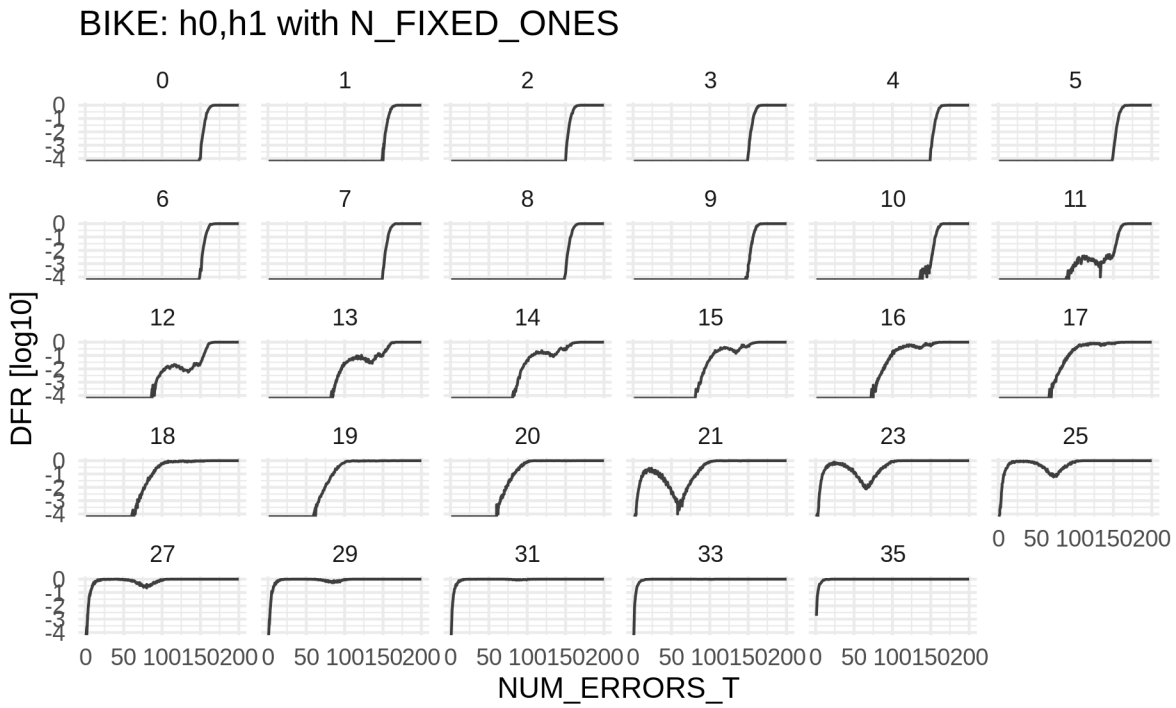

Figura 28: Andamento della DFR in funzione di NUM_ERRORS_T. Grafici presentati per diversi valori di N_FIXED_ONES.

Il parametro `t` è un elemento che viene attentamente analizzato in fase di progettazione dei crittosistemi di questo tipo. Di default è pari a 134 per SL = 1 ; già con sequenze lunghe 11 o più si osserva una DFR sicuramente troppo elevata per garantire la sicurezza.

E' opportuno per avere una migliore comprensione graficare le stesse curve dopo averle adeguatamente smussate, dato che il grafico appare abbastanza frastagliato, in conseguenza della casualità introdotta dall'aver svolto 10000 test per ogni coppia di parametri analizzati.

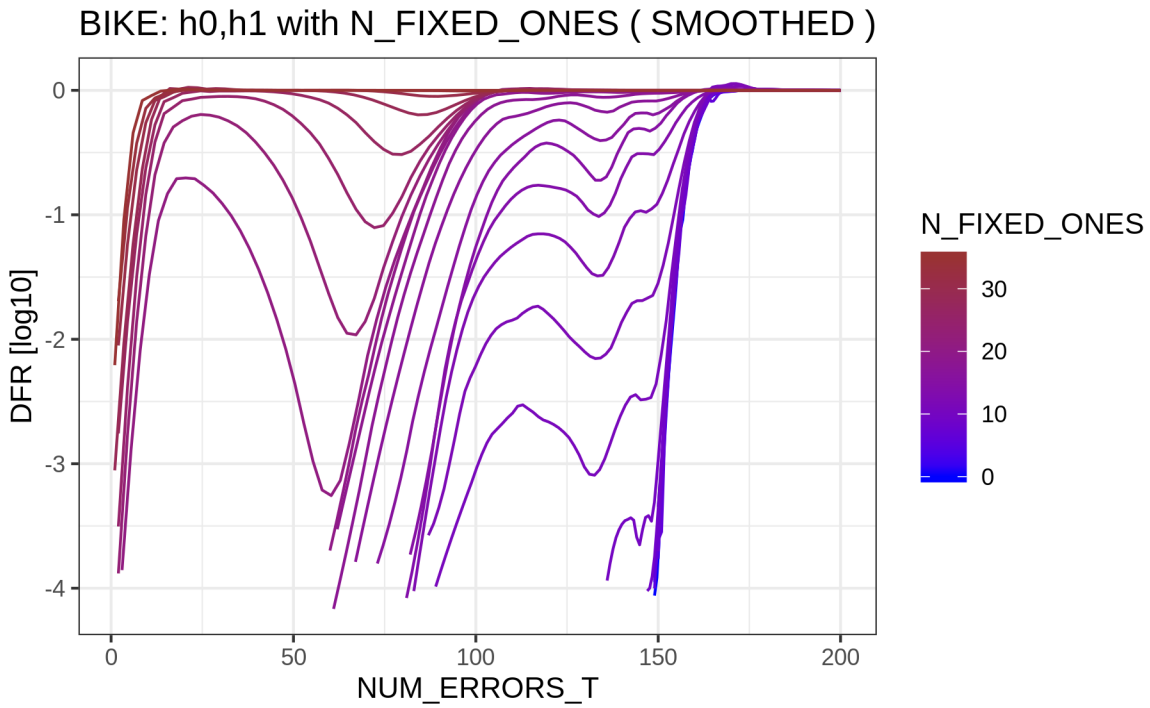

Figura 29: Andamento della DFR in funzione di NUM_ERRORS_T per diversi valori di N_FIXED_ONES. Curve smussate.

Si può evincere dalle curve tracciate che valori bassi di N_FIXED_ONES non hanno un effetto significativo. D'altronde la probabilità che essi si verifichino è elevata.

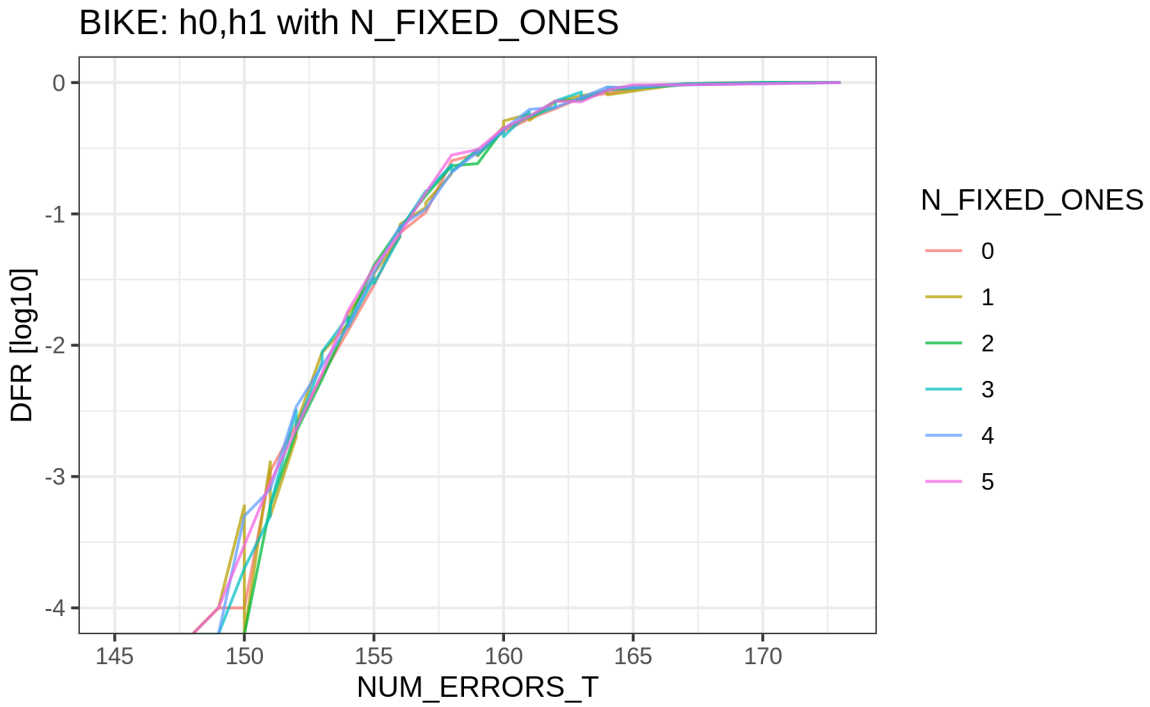

Figura 30: Andamento della DFR in funzione di NUM_ERRORS_T. Il colore rappresentata alcuni valori di N_FIXED_ONES, presi a campione.

Al contrario valori elevati rendono difficile la decodifica anche per valori di t molto piccoli.

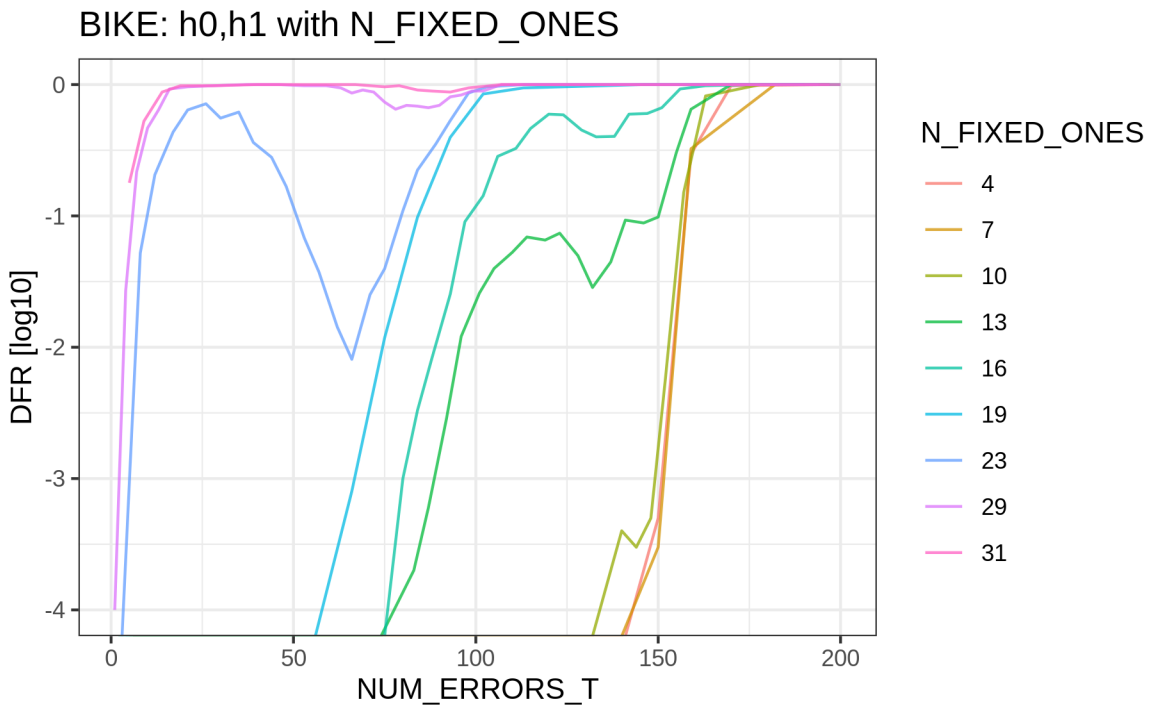

Figura 31: Andamento della DFR in funzione di NUM_ERRORS_T. Il colore rappresentata alcuni valori di N_FIXED_ONES, presi a campione.

Per NUM_ERRORS_T = 134 non si osserva un peggioramento sensibile per le sequenze lunghe 10 o meno. La degradazione è probabilmente comunque presente ma porta ad una DFR abbastanza bassa da non essere rilevata simulando 10^4 iterazioni. Aumentando t, come avviene per i Livelli di Sicurezza superiori, aumenta la robustezza rispetto all'insorgere di questo tipo di chiavi deboli.

Può essere utile riportare infine i valori in forma tabellare, per avere una visualizzazione pratica dei valori numerici esatti. Le tabelle di seguito sono state costruite come fatto in precedenza per LEDAcrypt.

N_FIXED_ONES	T:DFR==0.5	T:DFR==0
2	160	150
3	160	149
4	160	149
5	160	149
6	160	149
7	160	149
8	160	148
9	160	149
10	160	145
11	160	133

N_FIXED_ONES	T:DFR==0.5	T:DFR==0
12	159	90
13	158	85
14	153	83
15	151	81
16	139	75
17	108	69
18	98	65
19	94	62
20	95	62
21	92	58

Tabella 6: Tabella riassuntiva dei valori della DFR per diversi valori di N_FIXED_ONES.

In conclusione i dati confermano sperimentalmente come la generazione casuale di chiavi deboli di questo tipo porta ad un aumento sensibile e rapido della DFR. Questo fenomeno può essere ulteriormente studiato per valutarne esattamente l'impatto sul livello di sicurezza, o potrebbe essere potenzialmente alla base di un attacco qualora si riuscisse ad indurre in qualche maniera la generazione di chiavi di questo tipo.

Per questi scenari non si è giunti ad avere risultati numerici per il valore limite della DFR. Per completare l'analisi manca il calcolo esatto della probabilità di occorrenza per un particolare valore di `N_FIXED_ONES`. La formula per ricavarla non è semplice da elaborare. In questo lavoro si è preferito approfondire un altro tipo di elementi deboli, che verranno trattati nel successivo Capitolo 8.

7 CASO DI STUDIO 2 : ANALISI DI VETTORI DI ERRORE DEBOLI

Si vuole ora studiare la possibilità di attaccare BIKE sfruttando, per una data chiave, particolari tipi di errori "deboli", in quanto hanno proprietà particolari che portano ad un sensibile aumento della DFR.

In particolare di seguito si esaminerà un tipo di vettori di errore deboli per BIKE, la cui elaborazione concettuale è basata sullo studio del funzionamento delle soglie di decodifica studiate approfonditamente nel capitolo 4. Verrà analizzato il caso in cui la posizione degli elementi 1 del vettore **e** sia parzialmente coincidente con gli elementi del supporto di **H**. Lo scopo è quello di verificare che vettori **e** di questo tipo portino a difficoltà di decodifica, minando i presupposti di sicurezza e affidabilità dell'algoritmo.

Si tratta quindi di simulare la DFR che si ha quando viene generato un particolare tipo di vettori **e**, esaminati in funzione del parametro TX che indica il numero dei loro elementi 1 che sono presenti anche nel supporto di $\mathbf{h_0}$ (o di $\mathbf{h_1}$).

La rispettiva probabilità di occorrenza può essere calcolata matematicamente in modo esatto, dato che si suppone che **e** sia generato casualmente e uniformemente poiché deriva da una funzione di hash. Si riesce quindi a calcolare un valore limite per la DFR effettuando una media pesata tra la DFR stimata per ogni TX e la probabilità che venga generato un **e** che rientra in quella particolare casistica.

Si tratta di un valore limite perché per tutti i vettori che non sono stati considerati può accadere di avere ugualmente degli errori. E' sufficiente provare che tale valore per la DFR è superiore a $2^{-\lambda}$ per negare l'assunzione di sicurezza IND-CCA2. E' importante riuscire ad avere dei valori limite teorici, dato che per ottenere una stima tramite simulazioni occorrerebbero almeno 2^{128} prove, che sono troppe per essere simulate.

7.1 Analisi dell'occorrenza

Nel paragrafo 6.4 è stato spiegato che per calcolare il valore limite per la DFR si effettuerà una media pesata tra le diverse eventualità e la probabilità che queste si verifichino (figura 32). La probabilità di occorrenza può essere calcolata analiticamente.

La formula per valutare quanti siano i possibili vettori **e** con TX elementi 1 in comune con la prima colonna di **h** è:

$$CASI_{TX} = \binom{DV}{TX} \cdot \binom{N_{BITS} - DV}{T - TX}$$

Ovvero il prodotto tra tutte le possibili combinazioni che si possono ottenere prendendo TX elementi tra i dv elementi che costituiscono la prima colonna di **h** e tutte le altre possibilità per i restanti 1, che sono in numero pari a T - TX e possono occupare N_BITS posizioni, ad eccezione delle dv che fanno parte della prima colonna **h**.

Il numero di possibili vettori è dato semplicemente dai possibili modi di distribuire i T elementi non nulli tra le posizioni possibili, che sono N_BITS, quindi:

$$CASI_{TOT} = \binom{N_{BITS}}{T}$$

Quindi la probabilità che sia generato un vettore **e** con uno specifico valore TX è:

$$PROB_{TX} = \frac{\binom{DV}{TX} \cdot \binom{N_{BITS} - DV}{T - TX}}{\binom{N_{BITS}}{T}}$$

Di seguito viene graficata questa formula. I valori numerici sono relativi a BIKE-1 CCA con SL=1. La stessa curva graficata con valori presi dagli altri livelli di sicurezza esibisce un andamento generale analogo.

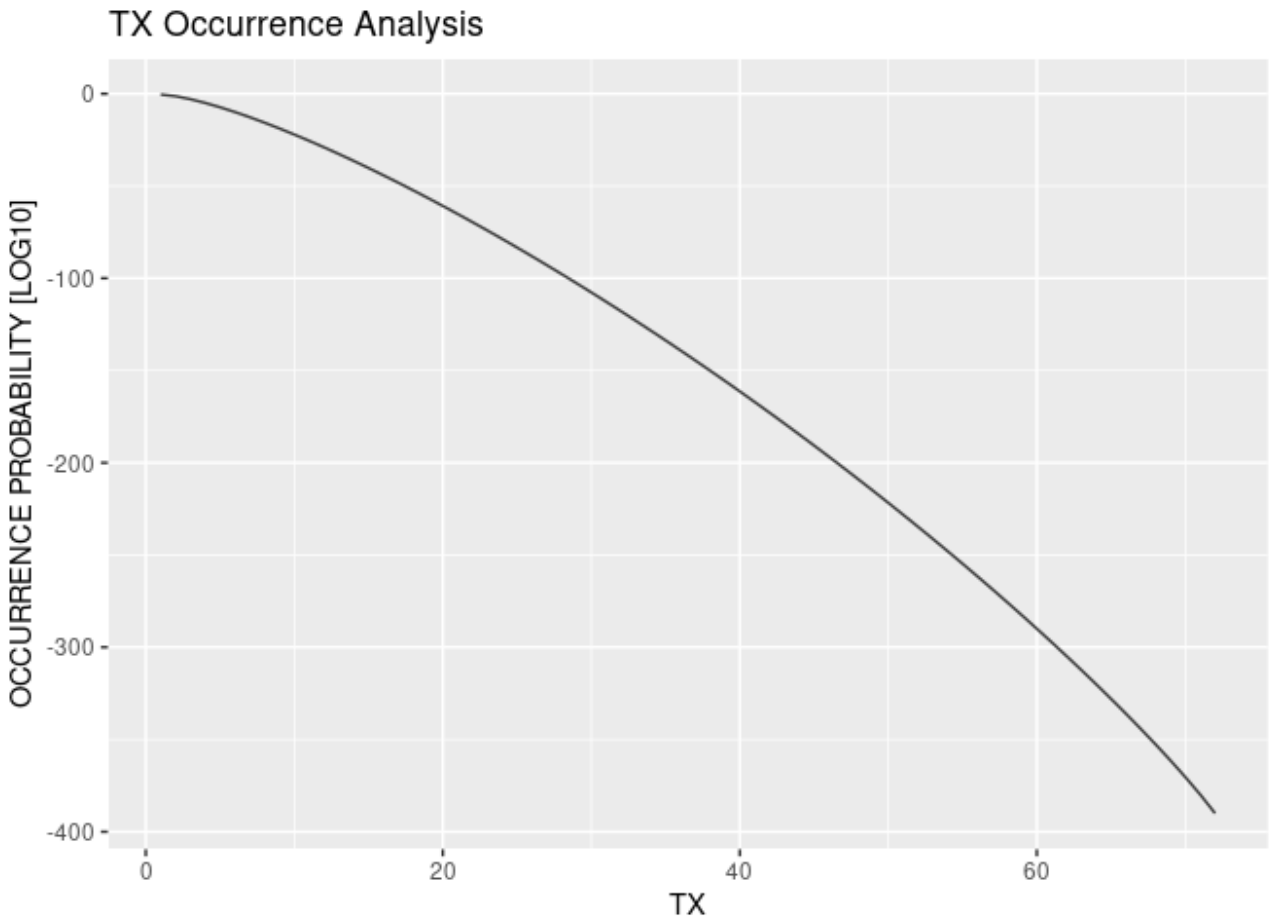

*Figura 32: Probabilità di occorrenza di un vettore **e** con TX elementi 1 in comune con **h0**. I valori numerici sono relativi a BIKE-1 CCA con SL=1.*

7.2 Calcolo del valore limite per la DFR

Effettuando una media pesata tra la DFR precedentemente simulata su tutti i possibili valori di TX (da un minimo di 0 ad un massimo di dv) e la probabilità che quel particolare valori di TX si verifichi, si può ottenere una stima finale della DFR:

$$DFR_{strimata} = \frac{\sum_{TX=0}^{DV} \binom{DV}{TX} \cdot \binom{N_{BITS} - DV}{T - TX} \cdot DFR(TX)}{\binom{N_{BITS}}{T}}$$

E' importante notare che il risultato finale ha valore generale, e non dipende da TX, in quanto comprende tutte le possibilità: è cioè una valutazione indiretta della DFR ottenuta analizzando dei risultati in casi particolari ma che nella loro totalità coprono la casistica completa. Per come sono stati effettuati i singoli test, questa stima può essere considerata una buona approssimazione per difetto. E' una stima per difetto perché per ogni TX sono stati effettuati solo 10000 test, per cui nei casi con DFR simulata < 10^{-4} , questa è stata approssimata a 0.

$$DFR_{reale} > DFR_{stimata}$$

Come si è visto in precedenza, la probabilità di occorrenza decresce molto rapidamente all'aumentare di TX. Questo vuol dire che per valori bassi di TX la probabilità di occorrenza è relativamente alta e che è possibile che l'impatto sulla stima totale sia significativo anche se la DFR è molto minore di 10^{-4}, dato che un minore ordine di grandezza della DFR può essere compensato dal fatto che la probabilità di verificarsi è di svariati ordini di grandezza più grande.

Sarebbe possibile quindi effettuare simulazioni più estese in modo da calcolare limiti più accurati.

Dato che i calcoli da effettuare riguardano numeri grandi, molti programmi hanno difficoltà a gestirli correttamente e ad elaborare un valore numerico. E'

necessario perciò ricorrere a programmi appositamente pensati per l'analisi numerica. In questo lavoro ci si è avvalsi di PARI-GP.

Il codice PARI-GP che implementa le formule precedentemente descritte è riportato in Figura 33.

```
tx = 0;
PROB = 0;

while(DFR = fileread(FILEDFR), print(DFR); print(tx);
PROB += binomial(DV,{tx})*binomial(N_BITS-DV,{T-tx})*eval(DFR);
tx = tx + 1)

\p 192
print(PROB)
print(binomial(N_BITS,T))
PROB = PROB/binomial(N_BITS,T) + 0.
PROB_BASE2 = log(PROB)/log(2)
```

Figura 33: Codice scritto in linguaggio PARI-GP per calcolare il valore limite per la DFR.

La variabile FILEDFR è un semplice file .txt in cui ogni riga corrisponde al valore di DFR per quel particolare TX, e le altre variabili devono essere assegnate al valore di interesse in base al Security Level.

7.3 Test

Per svolgere i test si è modificato il codice originario in modo che alcuni elementi non nulli di **e** venissero generati deterministicamente, scegliendo tra gli elementi 1 della prima colonna del supporto della matrice **h**.

Il numero di elementi generati in questo modo verrà indicato come TX ed è il parametro di riferimento per le analisi.

Ricordiamo che dv è il numero di elementi 1 per ogni riga/colonna delle matrici circolanti e T è il peso del vettore di errore. N_BITS indica la dimensione di **e**.

Per ogni valore di TX sono state effettuate 10000 prove di codifica/decodifica, divise in 1000 per 10 diverse chiavi, in modo da stimare la DFR con accuratezza sufficiente per i nostri scopi.

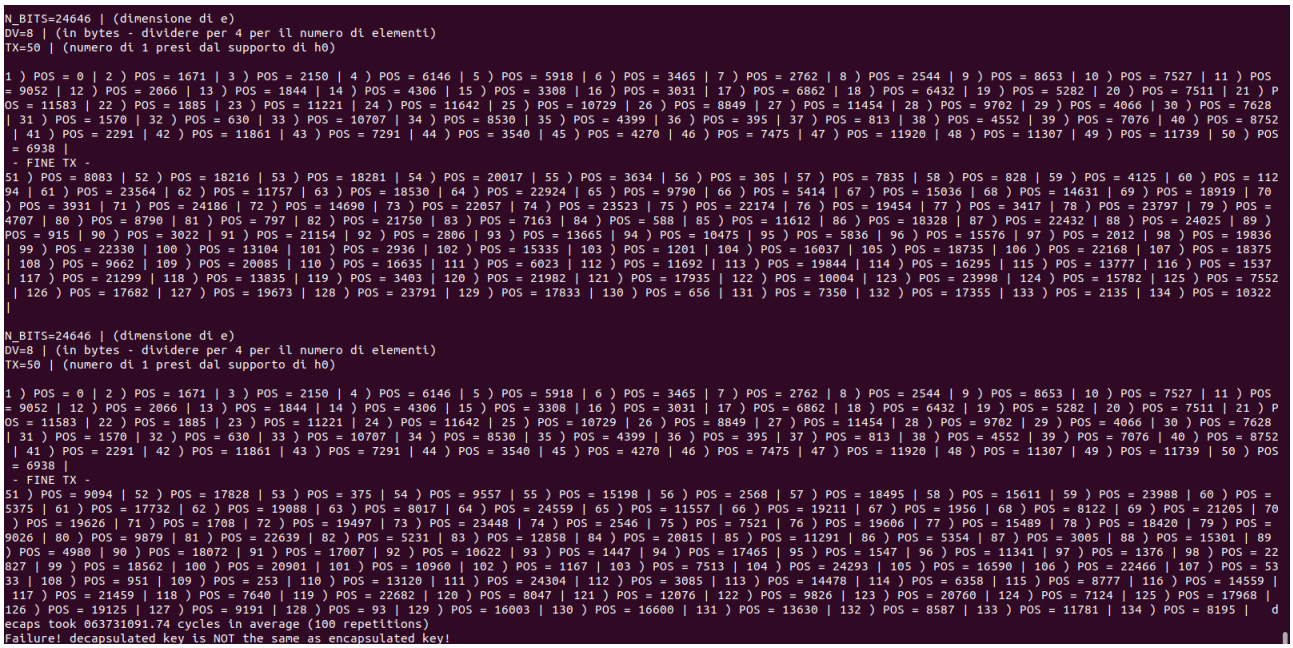

Figura 34: Finestra di comando durante l'esecuzione del codice di BIKE modificato..

Nella Figura 34 viene presentato un esempio di come appare la finestra di comando durante l'esecuzione del codice di questi test. Vengono stampati a video le posizione di tutti gli elementi non nulli durante la generazione di **e**; si può notare come i primi TX elementi siano sempre uguali, perché in tutte le generazioni essi sono presi da elementi in comune con il supporto di $\mathbf{h}_0$. Infatti l'immagine si riferisce a uno dei 1000 test che riguardano la stessa **h**. Le funzioni

di stampa sono state aggiunte per motivi espositivi e di verifica, ma sono state commentate nel codice finale in modo che fosse più performante.

7.4 BIKE V3

Verrà qui analizzato BIKE-1, come presentato nella sua versione v3. E' importante notare che l'implementazione di BIKE rilasciata dagli autori è la versione BIKE-1 CCA, che genera i vettori di errore usando come seme l'hash della chiave pubblica. Perciò indurre la generazione di vettori di errore con caratteristiche specifiche è molto difficile.

Tuttavia se riuscissimo a trovare facilmente degli errore in decodifica, allora lo schema non potrebbe più essere considerato IND-CCA2, perché ciò significherebbe che è facile produrre dei fallimenti.

<table>
<tr><th></th><th colspan="2">BIKE-1</th><th colspan="2">BIKE-1-CCA</th></tr>
<tr><td>SK</td><td colspan="2">(h_0, h_1)</td><td colspan="2">$(h_0, h_1, \sigma_0, \sigma_1)$</td></tr>
<tr><td></td><td colspan="4">with $|h_0| = |h_1| = w/2$</td></tr>
<tr><td>PK</td><td colspan="4">$(f_0, f_1) \leftarrow (gh_1, gh_0)$</td></tr>
<tr><td>Enc</td><td colspan="4">$m \xleftarrow{\$} \mathcal{R}$</td></tr>
<tr><td></td><td colspan="2">$(e_0, e_1) \xleftarrow{\$} \mathcal{R}^2$</td><td colspan="2">$(e_0, e_1) \leftarrow \mathbf{H}(mf_0, mf_1)$</td></tr>
<tr><td></td><td colspan="4">such that $|e_0| + |e_1| = t$</td></tr>
<tr><td></td><td colspan="4">$(c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$</td></tr>
<tr><td></td><td colspan="2">$K \leftarrow \mathbf{K}(e_0, e_1, c)$</td><td colspan="2">$K \leftarrow \mathbf{K}(mf_0, mf_1, c)$</td></tr>
<tr><td>Dec</td><td colspan="4">$s \leftarrow c_0 h_0 + c_1 h_1 \; ; u \leftarrow 0$</td></tr>
<tr><td></td><td colspan="4">$(e'_0, e'_1) \leftarrow \mathsf{Decode}(s, h_0, h_1, u)$</td></tr>
<tr><td></td><td colspan="2"></td><td colspan="2">$(e''_0, e''_1) \leftarrow \mathbf{H}(c_0 + e'_0, c_1 + e'_1)$</td></tr>
<tr><td></td><td colspan="2">$K \leftarrow \mathbf{K}(e'_0, e'_1, c')$</td><td>$K \leftarrow \mathbf{K}(\sigma_0, \sigma_1, c)$</td><td>$K \leftarrow \mathbf{K}(c_0 + e'_0, c_1 + e'_1, c)$</td></tr>
</table>

Tabella 7: Funzionamento di BIKE-1 e BIKE-1 CCA.

7.4.1 Implementazione

E' stato modificato il codice di generazione dell'errore. Invece di generate_sparse_rep() (vedi sezione 5.2) è stata scritta una nuova funzione, la quale prende in ingresso anche la chiave segreta sk.

```
ret_t  my_generate_err(OUT uint64_t *    a,
                       OUT idx_t         wlist[],
                       IN const uint32_t weight,
                       IN const uint32_t len,
                       IN const uint32_t padded_len,
                       IN OUT aes_ctr_prf_state_t *prf_state,
                       IN const sk_t *sk,  //H, per prendere 1 dal suo supporto
                      uint64_t TX)
{
  assert(padded_len % 64 == 0);
  // Bits comparison
  assert((padded_len * 8) >= len);

  uint64_t ctr = 0;
  struct sk_s H;
  H = *sk;

  int DVtest = sizeof(H.wlist[0].val);
  printf("\n \nN_BITS=%u | (dimensione di e) \n", len);
  printf("DV=%d | (in bytes - dividere per 4 per il numero di elementi) \n", DVtest);
  printf("TX=%lu | (numero di 1 presi dal supporto di h0) \n \n", TX);
  do
  {
    wlist[ctr] = H.wlist[0].val[ctr];

    if(is_new(wlist, ctr)){
        printf("%lu ) POS 1 = %d | ", ctr, wlist[ctr]);
    ctr += is_new(wlist, ctr);
    };

  } while(ctr < TX); //weight

  printf("\n - END TX -> \n");

  // Generate weight rand numbers
  do
  {
    GUARD(get_rand_mod_len(&wlist[ctr], len, prf_state));

    if(is_new(wlist, ctr)){
        printf("%lu ) POS 1 = %d | ", ctr, wlist[ctr]);
        ctr += is_new(wlist, ctr);
    };

  } while(ctr < weight); //weight

  printf("\n \n");
  // Initialize to zero
  memset(a, 0, (len + 7) >> 3);

  // Assign values to "a"
  secure_set_bits(a, wlist, padded_len, weight);
```

Figura 35: Codice originale scritto per effettuare le elaborazioni. Le funzioni che stampano a video possono essere commentate per un'esecuzione più veloce.

7.4.2 Analisi della DFR

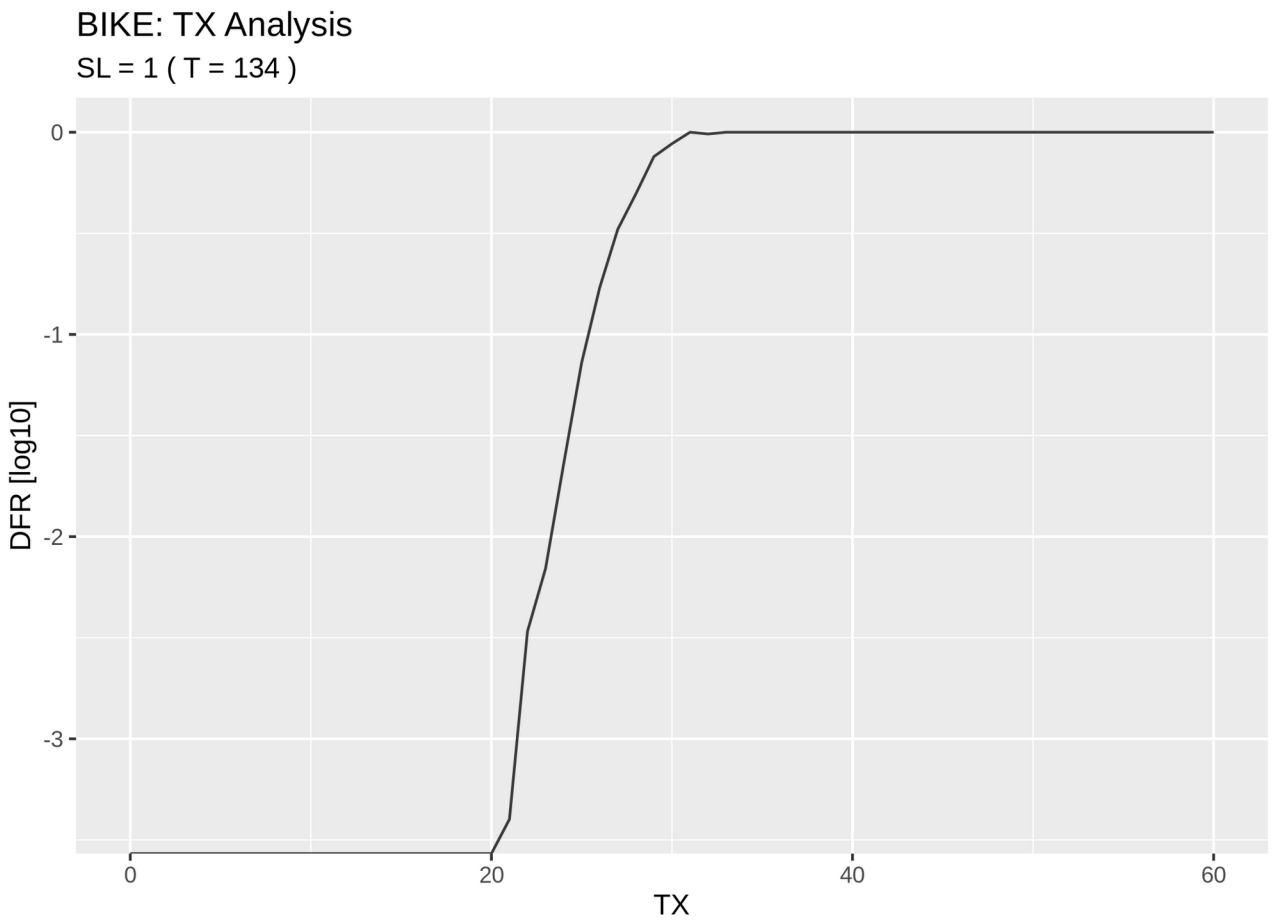

Figura 36: Analisi della DFR di BIKE-1 CCA (v3), quando SL=1, al variare di TX, numero di elementi 1 in comune tra ***e*** *ed* $\boldsymbol{h_0}$.

Per il livello di sicurezza 1, dv = 71 , T1 = 134 e la dimensione di **e** è 23558. La DFR è inferiore a 10^{-4} per TX minore di 21, ed aumenta al crescere di esso fino a che, per TX = 33, la DFR diventa $\simeq 1$ (50 errori su 50 tentativi di decodifica).

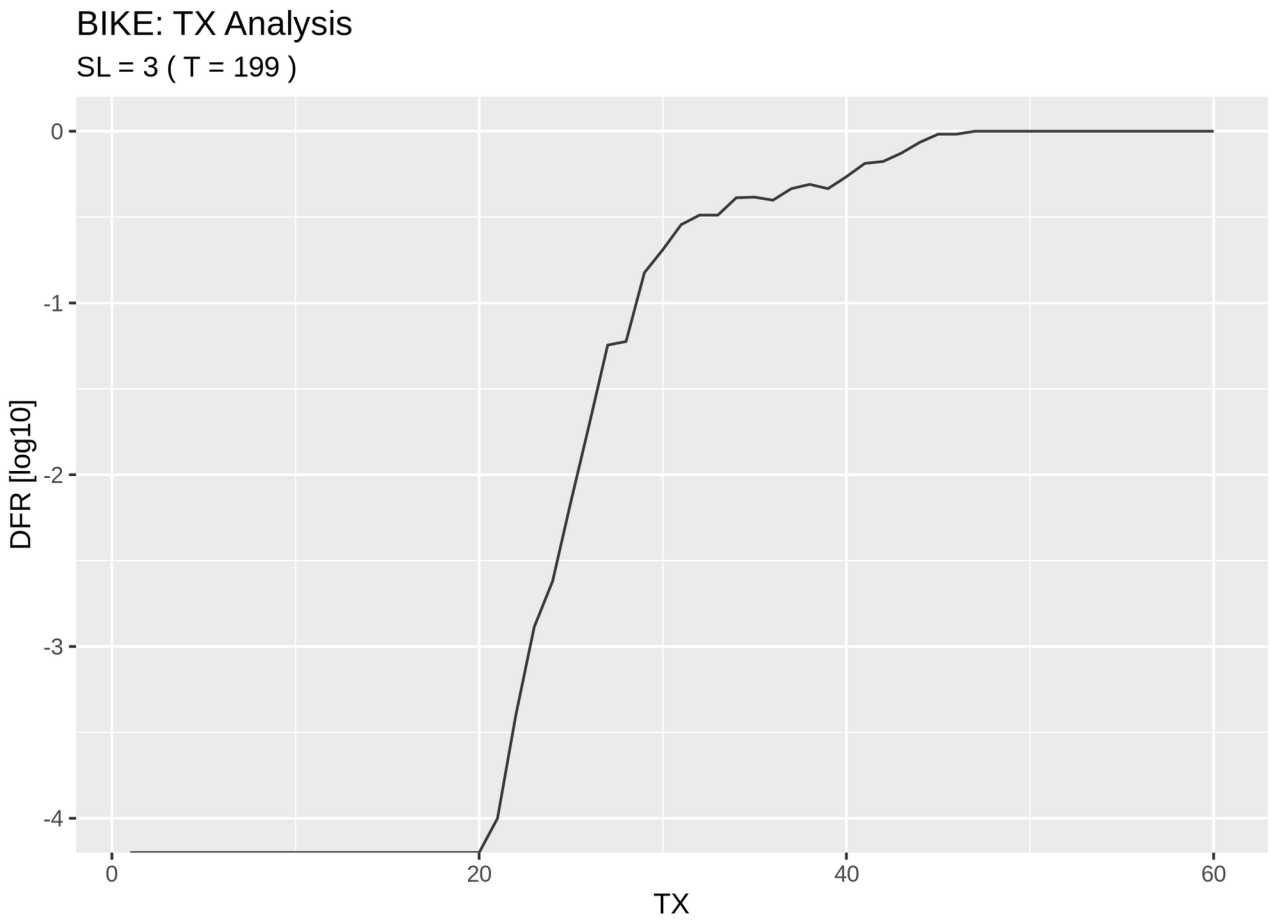

Figura 37: Analisi della DFR di BIKE-1 CCA (v3), quando SL=3, al variare di TX, numero di elementi 1 in comune tra ***e*** *ed* $\boldsymbol{h_0}$.

La figura 37 si riferisce invece al livello di sicurezza 3, per cui dv = 103 e T1 = 199. La lunghezza di **e** è 49642. Per TX minore di 21 la DFR è inferiore a 10^{-4}, poi aumenta al crescere di TX e per TX= 47 diventa $\simeq 1$.

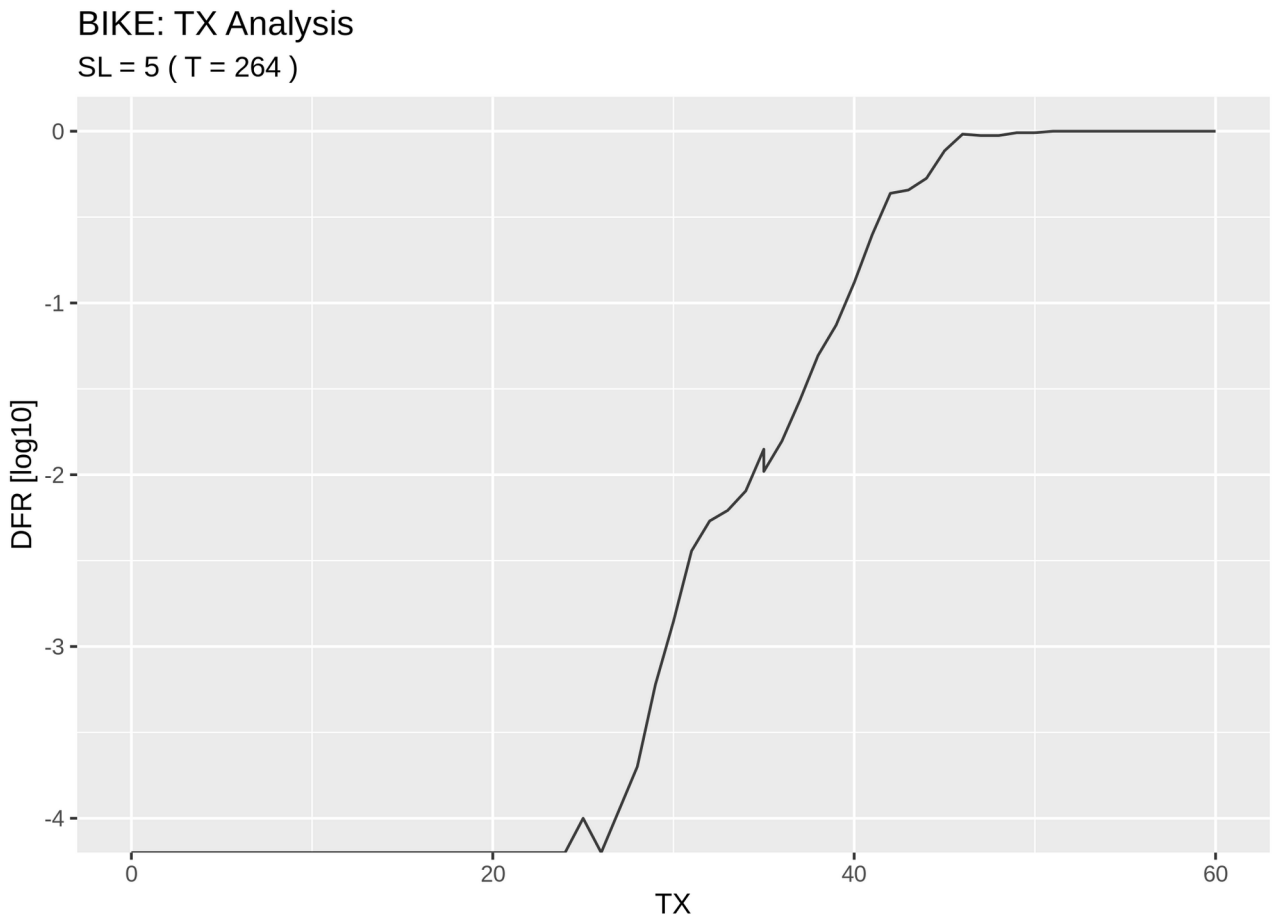

Figura 38: Analisi della DFR di BIKE-1 CCA (v3), quando SL=5, al variare di TX, numero di elementi 1 in comune tra ***e*** *ed* $\boldsymbol{h}_0$.

Infine per il livello di sicurezza 5, dv = 137 e T1 = 264. La lunghezza di **e** è 81194. La DFR è inferiore a 10^{-4} per TX minore di 28 e diventa $\simeq$ 1 per TX = 51.

Si può concludere che le prestazioni del decodificatore peggiorano progressivamente all'aumentare di TX. Le prestazioni diventano non sufficientemente buone quando il vettore di errore ha queste particolari caratteristiche, e ciò può essere sfruttato per minare l'assunzione di IND-CCA-2. Ciò è dovuto al fatto che vettori di errori con questa configurazione portano ad un uso non previsto delle soglie del decodificatore, che quindi opera in maniera molto poco efficiente.

Effettuando una media pesata tra la DFR per ogni TX e la relativà probabilità di occorrenza, che si può calcolare con le formule del paragrafo 7.2, si può finalmente ottenere un valore minimo per la DFR.

7.4.3 Risultati

Di seguito verranno i riportati i dati numerici ottenuti con le simulazioni e i mezzi descritti precedentemente.

Livello di Sicurezza 1

Le simulazioni qui effettuate consentono di affermare per il Security Level 1 che:

$DFR_{SL=1} > 2.6013 \cdot 10^{-34} \simeq 2^{-111.57}$

Mentre era stata assunta pari a 2^{-128}.

Quindi in questo caso l'analisi condotta è sufficiente a mettere in dubbio le assunzioni su cui si basa la sicurezza del sistema.

Livello di Sicurezza 3

Per il Security Level 3:

$DFR_{SL=3} > 4.9659 \cdot 10^{-34} \simeq 2^{-110.63}$

Mentre era stata assunta pari a 2^{-192}.

Il fatto che la DFR limite sia maggiore rispetto al precedente livello di sicurezza è da riscontrarsi nel modo in cui i test sono stati effettuati. Se si fossero eseguite prove in grado di rivelare per ogni TX delle DFR anche minori di 10^{-4} la stima sarebbe più precisa. L'accuratezza raggiunta è ad ogni modo sufficiente ad affermare che anche in questo caso gli obiettivi per la sicurezza non sono raggiunti.

Livello di Sicurezza 5

Infine per Security Level 5 la DFR è significativamente maggiore di quella necessaria, in quanto

$DFR_{SL=5} > 2.3163 \cdot 10^{-40} \simeq 2^{-131.67}$

Mentre era necessaria minore o uguale a 2^{-256}.

Vengono riportati i valori dichiarati dagli autori di BIKE in modo da poter avere un confronto e trarre le conclusioni circa la sicurezza di BIKE v3.

	DFR dichiarata [log2]	DFR minima [log2]
Security Level 1	-128	-111.57
Security Level 3	-192	-110.63
Security Level 5	-256	-131.67

Tabella 8: Risultati relativi a BIKE v3.

Per la versione di BIKE Round 2 (v3) i risultati dimostrano che l'assunzione nel paper di riferimento [2] di poter elaborare una stima della DFR per estrapolazione non è del tutto accurata. Si può concludere da questa analisi che la DFR è sicuramente superiore a quella dichiarata, necessaria affinché si possa affermare che il sistema è IND-CCA2.

7.4.4 Conclusioni

Nel caso qui studiato di BIKE-1 CCA (v3), l'analisi di vettori **e** in cui il numero TX di elementi 1 coincidenti con quelli del supporto di $\mathbf{h}_0$ sia particolarmente alto riesce a mettere in mostra un caso particolare che ha effetto significativo sulla DFR. Valutare l'impatto di questo tipo di vettori è sufficiente a porre un limite inferiore sul valore della DFR che è di per sé abbastanza stringente da minare i presupporti per la sicurezza IND-CCA2. Il limite inferiore calcolato con questa analisi è più alto di quello stimato dagli autori, la cui analisi è quindi errata.

Questa conclusione è un risultato importante, perché fornisce un esempio pratico di come la metodologia proposta in questo lavoro, spiegata per esteso nel paragrafo 6.4, possa essere usata in modo proficuo per avere una valutazione della sicurezza di un crittosistema effettuando un numero di simulazioni contenuto. In questo caso una sola analisi di questo tipo è sufficiente a violare le assunzioni di sicurezza avanzate dagli autori.

7.5 BIKE V4

Nel periodo della stesura di questo elaborato (primo semestre 2020) è stata rilasciata la versione v4 di BIKE. Dato che le analisi precedenti hanno portato ad un risultato significativo, si è esteso il test anche alla versione più recente di BIKE. In BIKE v4 i parametri sono stati scelti in modo da essere più cautelativi e perciò dovrebbero garantire una maggiore sicurezza; in questo paragrafo si vuole studiare l'effetto dei vettore di errore deboli, similmente a quanto con la versione precedente, per vedere se si riescono ad ottenere gli stessi risultati o meno.

E' interessante notare che in questa versione gli autori non dichiarano BIKE come IND-CCA; viene invece fornita una dimostrazione formale che tale garanzia di sicurezza è raggiunta se la DFR del decoder è sufficientemente bassa (2^{-128} e 2^{-192} per i rispettivi livelli di sicurezza), specificando però che non è possibile provare che tali valori della DFR siano effettivamente raggiunti.

7.5.1 Implementazione

E' stata modificata la funzione status_t generate_sparse_rep() (più informazioni in appendice) in modo tale che creasse chiavi deboli con le caratteristiche che ci interessa studiare. Sono stati svolti test come fatti con 10000 iterazioni, divise in modo che fossero 1000 per 10 diverse chiavi ad ogni valore di TX.

```
status_t generate_error_tx(OUT uint8_t * r,
        IN  const uint32_t weight,
        IN  const uint32_t len,
        IN OUT aes_ctr_prf_state_t *prf_state,
        IN const unsigned char *sk,
        uint64_t TX)
{
    uint32_t rand_pos = 0;
    status_t res = SUCCESS;
    uint64_t ctr = 0;

    //position of 1 in secret key
    const sk_t* l_sk = (sk_t*)sk;

    uint32_t h0_compact[DV] = {0};
    uint32_t h1_compact[DV] = {0};

    convert2compact_gme(h0_compact, l_sk->val0);
    convert2compact_gme(h1_compact, l_sk->val1);

    uint64_t pos;

    //Ensure r is zero.
    setZero(r, DIVIDE_AND_CEIL(len, 8ULL));

   while(ctr < TX)
   {
        res = get_rand_mod_len(&rand_pos, DV, prf_state);
        pos = h0_compact[rand_pos];
        //pos = h1_compact[rand_pos] + R_BITS;

        if (!CHECK_BIT(r, pos))
        {
            ctr++;
            //No collision set the bit
            SET_BIT(r, pos);
            //printf("%lu) POS = %ld | ", ctr, pos);
        }

  }  //weight

  //printf("\n - END TX - \n");
```

Figura 39: Codice usato per i test di questo paragrafo. Per una migliore resa grafica, viene riportato solo la parte di codice che è stata modificata.

7.5.2 Analisi della DFR

I grafici presentati in questa sezione mostrano i risultati delle simulazioni per l'analisi della DFR in BIKE v4.

I grafici sono riportati fino al valore di TX=60 ma le elaborazioni sono state svolte fino al valore massimo possibile, pari al parametro t del crittosistema. Le curve possono essere estese in maniera ovvia (raggiunta DFR=1 non si discostano da tale valore).

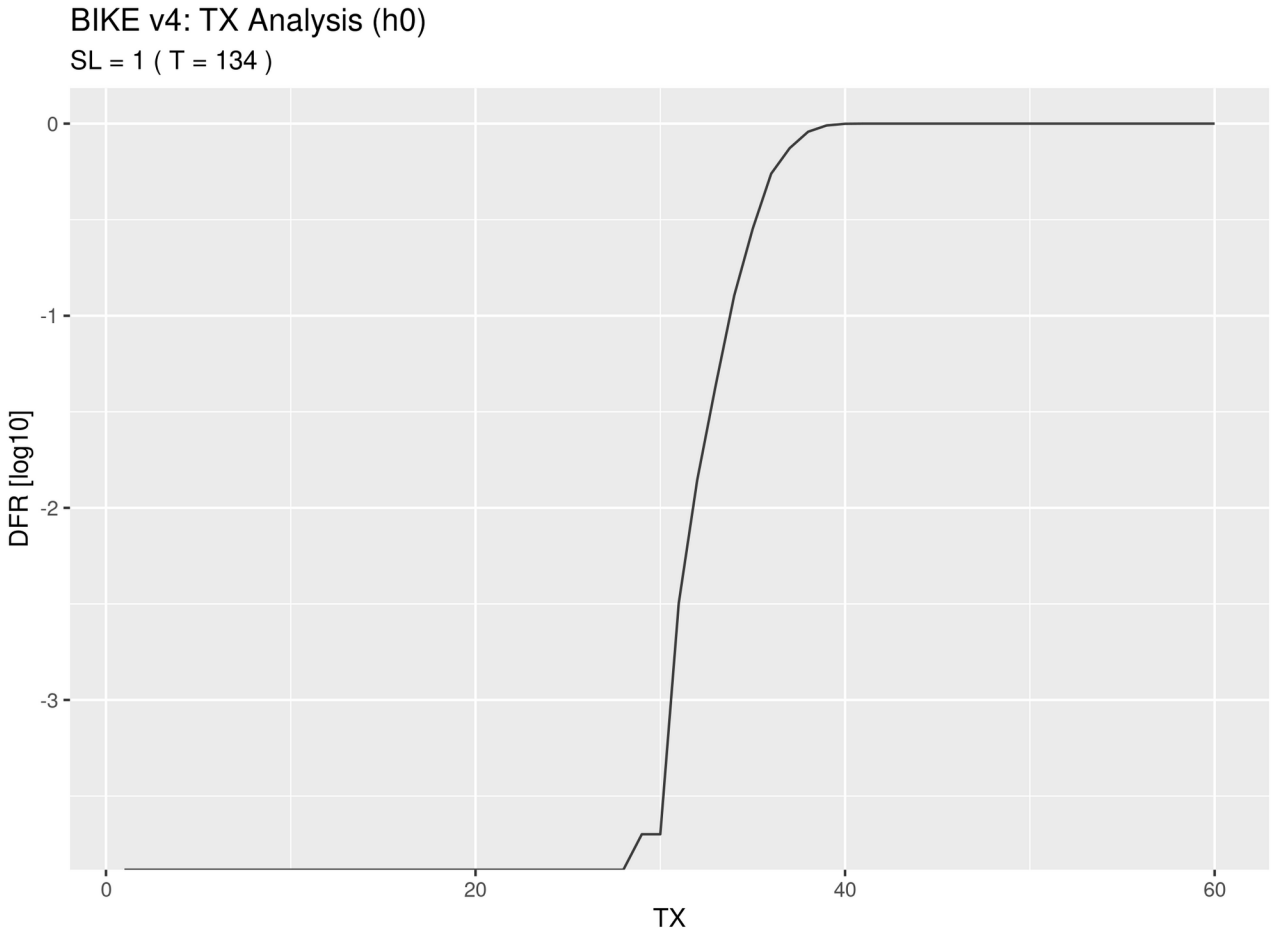

Figura 40: DFR di BIKE (v4) al variare di TX, riferito ad h_0. SL=1.

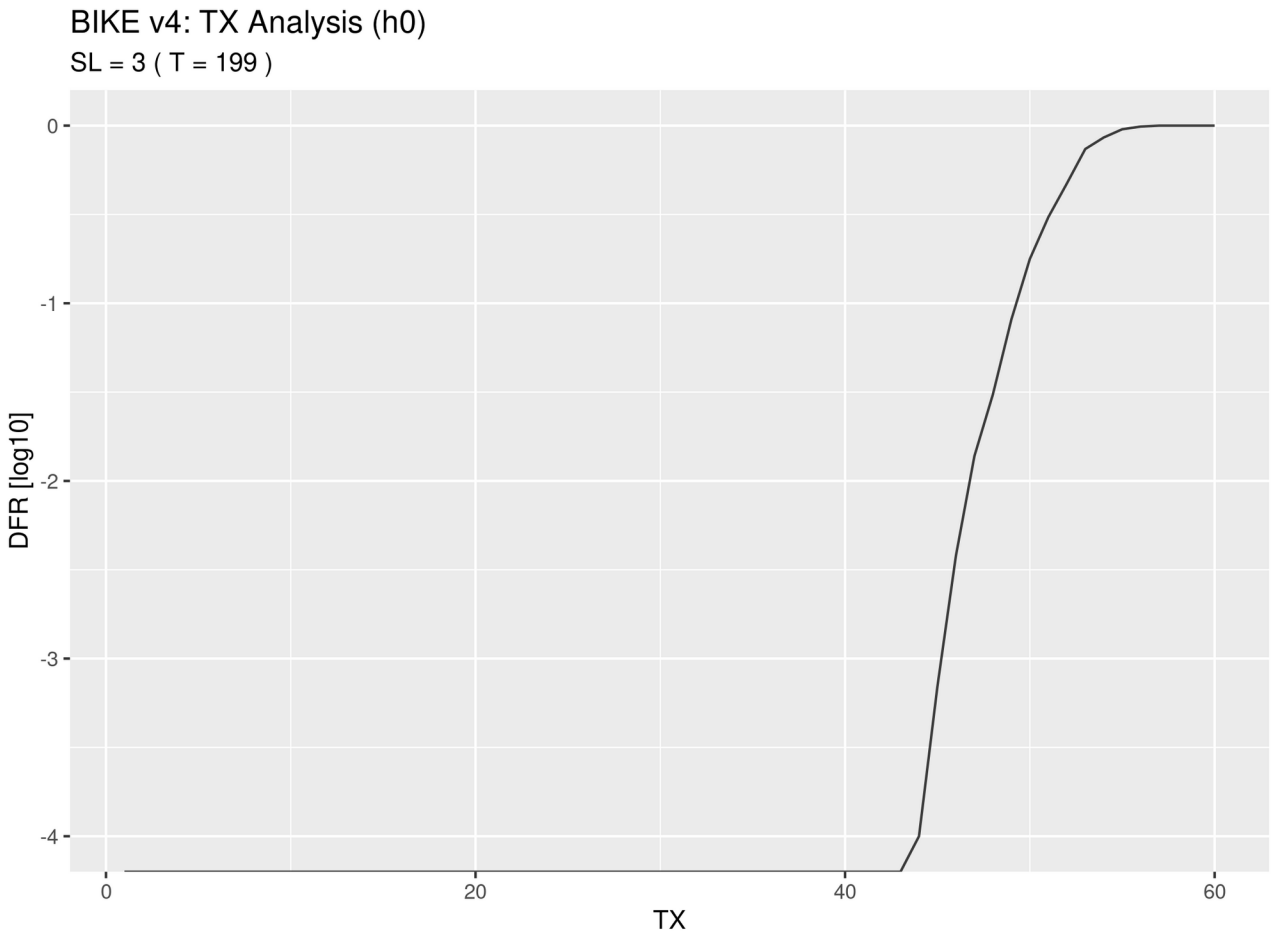

Figura 41: DFR di BIKE (v4) al variare di TX, riferito ad h_0. SL=3.

BIKE v4 ha un funzionamento simile alla versione precedente (coincide in pratica con BIKE-2 v3). Le curve sono infatti qualitativamente simili a quelle di BIKE v3, ma la DFR cresce significativamente più lentamente al crescere di TX. Ad esempio, per SL=1 il primo valore di TX per cui è avvenuto un errore in decodifica è 29, mentre in BIKE v3 ciò è successo per TX pari a 21.

Quando gli stessi esperimenti vengono effettuati riferendosi ad $\mathbf{h_1}$ invece che ad $\mathbf{h_0}$, i risultati sono praticamente uguali. Le due matrici pur essendo diverse svolgono una funzione analoga ai fini del funzionamento dell'algoritmo, quindi tale risultato coincide con quanto atteso.

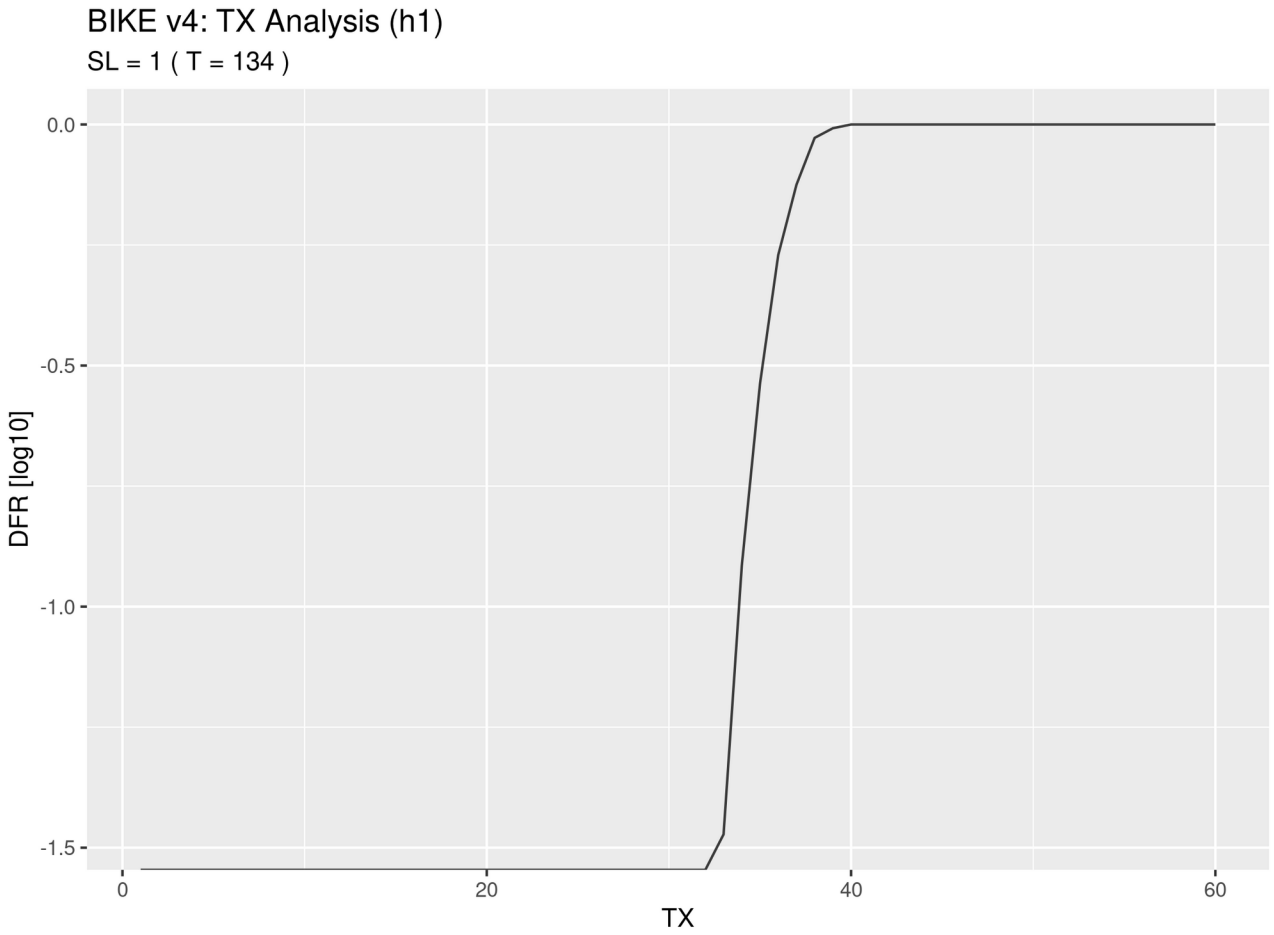

Figura 42: DFR di BIKE (v4) al variare di TX, riferito ad h_1. SL=1.

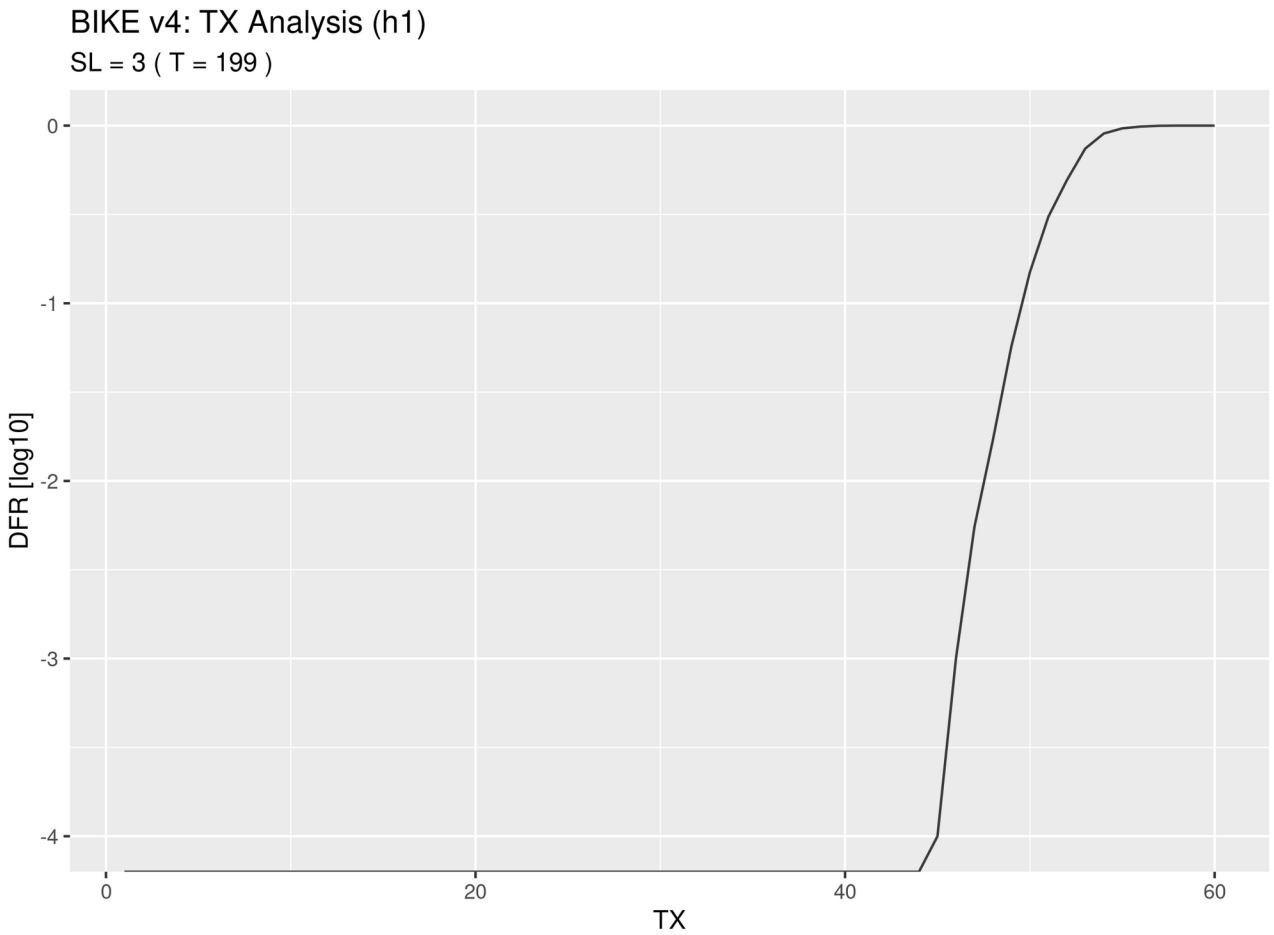

Figura 43: DFR di BIKE (v4) al variare di TX, riferito ad h_1. SL=3.

7.5.3 Analisi dell'occorrenza

Il calcolo della probabilità che si manifesti un vettore **e** con un particolare TX è del tutto analogo a quello enunciato in precedenza per BIKE v3, in quanto si tratta di un processo completamente casuale.

7.5.4 Risultati per h_0

Livello di Sicurezza 1

Le simulazioni qui effettuate consentono di affermare per il Security Level 1 che:

$DFR_{SL=1} > 8.0518 \cdot 10^{-51} \simeq 2^{-166.412}$

Livello di Sicurezza 3

Per il Security Level 3:

$DFR_{SL=3} > 1.0144 \cdot 10^{-80} \simeq 2^{-265.733}$

Gli esperimenti condotti non pongono di per sé stessi in dubbio l'assunzione che il requisito sulla DFR sia soddisfatto. Il limite per la DFR evidenziato dalla nostra analisi non è sufficiente a poter concludere se gli obiettivi per la sicurezza siano raggiunti o meno.

	DFR dichiarata [log2]	DFR minima [log2]
Security Level 1	-128	-166.412
Security Level 3	-192	-265.733

Tabella 9: Risultati relativi a BIKE v4, riferiti ad h0.

7.5.5 Risultati per h_1

Livello di Sicurezza 1

La DFR stimata per il Security Level 1 è circa la stessa di quella ottenuta nel caso precedente:

$DFR_{SL=1} > 1.4648 \cdot 10^{-49} \simeq 2^{-162.224}$

Livello di Sicurezza 3

Per il Security Level 3 vale lo stesso:

$DFR_{SL=3} > 1.4340 \cdot 10^{-80} \simeq 2^{-265.235}$

	DFR dichiarata [log2]	DFR minima [log2]
Security Level 1	-128	-162.224
Security Level 3	-192	-265.235

Tabella 10: Risultati relativi a BIKE v4, riferiti ad h1.

I risultati riferiti all'analisi di $\mathbf{h_1}$ sono dello stesso ordine di grandezza. Di conseguenza si possono trarre le stesse conclusioni tratte in precedenza.

7.5.6 Conclusioni

Gli esperimenti condotti in questo caso non bastano a minare l'ipotesi che il crittosistema abbia DFR abbastanza bassa da essere sicuro.

I risultati relativi ad $\mathbf{h_0}$ e quelli relativi ad $\mathbf{h_1}$ possono essere considerati complementari. Infatti essendo la generazione del vettore di errore casuale ed indipendente dalle caratteristiche di $\mathbf{h_0}$ ed $\mathbf{h_1}$, le due casistiche studiate possono essere considerate statisticamente indipendenti e quindi esaminate congiuntamente. Ad ogni modo, non è necessario svolgere calcoli precisi per sapere che la DFR stimata in questo modo sarebbe di al massimo un ordine di grandezza maggiore. Questo perché le due eventualità sono equiprobabili, perciò essendo incorrelate, l'effetto dell'una congiunto a quello dell'altra porta circa ad un raddoppio della DFR, dato che per ogni valore di TX la DFR è circa la stessa. Quindi non cambiano di fatto le considerazioni della sicurezza sin qui fatte.

Si può concludere che la specifica v4 è meglio progettata e potenzialmente sicura, sebbene le analisi qui svolte evidenzino in ogni caso delle criticità che sarebbe opportuno analizzare con mezzi più adeguati, in quanto ci sono dei segnali che portano a pensare che svolgendo test più accurati si potrebbe arrivare a dimostrare che il crittosistema non è sicuro, come è avvenuto con l'analisi di BIKE v3.

8 CONCLUSIONI

In questo lavoro è stata proposta una nuova metodologia per stimare la DFR di un crittosistema basati su codici QC-MDPC e quindi valutarne la sicurezza. Il metodo serve a stimare un limite inferiore alla DFR, e riesce a risolvere il problema di fornire un valore attendibile per grandezze che altrimenti non sarebbero stimabili tramite simulazioni numeriche. Questa metodologia è legata allo studio, individuazione e simulazione di particolari chiavi "deboli", ma ha validità generale.

Dopo aver spiegato teoricamente questo approccio e fornito le nozioni preliminari per rendere la trattazione consistente, sono stati forniti i risultati di due particolari casi di studio. Si è scelto di analizzare i crittosistemi LEDAcrypt e BIKE, entrambi proposti per la gara Post-Quantum Cryptography Standardization del NIST e basati su codici sparsi, in quanto di grande interesse e potenziale.

Si è studiato prima l'effetto di chiavi deboli con lunghe sequenze di elementi 1. I risultati sono stati graficati ed analizzati ed hanno permesso una più profonda comprensione della sicurezza di questi crittosistemi. Non si è giunti a calcolare dei risultati numerici, ma ritiene che questo tipo di chiavi deboli non siano un problema per la sicurezza. Ciononostante non si può escludere che esistano altri tipi di chiavi deboli che invece lo siano.

Quindi si è testato l'impatto di chiavi con elementi in comune con la matrice **H** nel caso di BIKE, studiandone due diverse versioni.

Il risultato probabilmente più interessante è stato riuscire a dimostrare che gli assunti di sicurezza di BIKE nella sua versione v3 non sono rispettati e che il sistema è potenzialmente vulnerabile. Lo stesso non è vero per BIKE v4.

Come sviluppi futuri si potrebbe elaborare una descrizione teorica più rigida e completa di tale metodologia, ed applicarla ad un vasto numero di crittosistemi la cui sicurezza sia legata alla DFR. L'auspicio è che tale tecnica possa affiancare i metodi di simulazione classica in modo da consentire di avere delle stime affidabili anche per valori numerici che renderebbero altrimenti il problema intrattabile.

9 APPENDICE : CODICE C

Si andrà ad analizzare brevemente la struttura ed il funzionamento delle implementazioni dei crittosistemi analizzati in questo lavoro: LEDAcrypt e BIKE. Entrambe sono scritte in linguaggio C/C++. Una panoramica sul funzionamento dei codici è utile, sia per motivi espositivi e didattici, sia perché fornisce al lettore interessato la possibilità di replicare o approfondire i test, ed in ultimo perché permette di avere una comprensione più chiara dei crittosistemi e quindi delle qualità delle analisi svolte in questo lavoro.

9.1 LEDAcrypt: Codice C

Il codice qui usato per le simulazione è scritto in C ed è quello corrispondente alla versione di LEDAcrypt del 24 luglio 2019 proposta per il Round 2; è reperibile su internet [14].

In particolare i test sono stati effettuati usando gli script messi a disposizione, i quali sono situati in

.../LEDAcrypt-master/Additional_Implementations/KEM/Benchmarking-KEM/bin

Per effettuare il test desiderato è necessario compilare il file .bin con i comandi

make clean

SL=1 N0=2 make

Dove la variabile `SL` indica il livello di sicurezza (Security Level) desiderato e la variabile `N0` indica invece il valore n_0 come descritto in precedenza. I valori particolari del codice precedente sono a titolo di esempio e corrispondono con i valori standard dell'implementazione.

Per esaminare e modificare il codice si deve operare sul file

.../LEDAcrypt-master/Additional_Implementations/KEM/Benchmarking-KEM/library/main_test.c

La funzione principale che viene richiamata è

void test_KEM_niederreiter_code(int ac, char *av[], long unsigned int NumTests)

la quale è definita all'interno del file test_niederreiter.c all'interno dello stesso folder.

I valori numerici dei parametri del crittosistema sono definiti in:

.../LEDAcrypt-master/Reference_Implementation/KEM/include/qc_ldpc_parameters.h

9.1.1 LEDAcrypt: Codice C per la generazione delle chiavi

Dato che in questo lavoro ci interessa studiare l'effetto di determinati tipi di chiavi, sono di particolare importanza le funzioni che servono a generarle. In particolare

key_gen_niederreiter(pk, sk,&niederreiter_keys_expander);

che si trova in .../LEDAcrypt-master/Reference_Implementation/KEM/library/niederreiter_keygen.c

la quale richiama la funzione

generateHPosOnes_HtrPosOnes(HPosOnes, HtrPosOnes, keys_expander);

la quale è definita in H_Q_matrices_generation.c nello stesso folder.

Una variabile di particolare interesse è HtrPosOnes , la quale indica la posizione degli elementi 1 nella matrice $\mathbf{H}^T$: è infatti una matrice binaria di dimensione [N_0 x dv] , dove ricordiamo che dv indica il peso di una riga della matrice.

```
void generateHPosOnes_HtrPosOnes(POSITION_T HPosOnes[N0][DV],
                         POSITION_T HtrPosOnes[N0][DV],
                         AES_XOF_struct *keys_expander
                        )
{
   for (int i = 0; i < N0; i++) {
      /* Generate a random block of Htr */
      rand_circulant_sparse_block(&HtrPosOnes[i][0],
                                  DV,
                                  keys_expander);
   }
   for (int i = 0; i < N0; i++) {
      /* Obtain directly the sparse representation of the block of H */
      for (int k = 0; k < DV; k++) {
         HPosOnes[i][k] = (P - HtrPosOnes[i][k])  % P; /* transposes indexes */
      }// end for k
   }
} // end generateHtr_HtrPosOnes
```

Figura 44: Generazioni degli elementi non-nulli casuali di H in LEDAcrypt. a

La principale funzione richiamata è

void rand_circulant_sparse_block(POSITION_T *pos_ones, const int countOnes, AES_XOF_struct *seed_expander_ctx)

in .../LEDAcrypt-master/Reference_Implementation/KEM/library/gf2x_arith_mod_xPplusOne.c

```
/*---------------------------------------------------------------------------*/
/* Obtains fresh randomness and seed-expands it until all the required positions
 * for the '1's in the circulant block are obtained */

void rand_circulant_sparse_block(POSITION_T *pos_ones,
                                 const int countOnes,
                                 AES_XOF_struct *seed_expander_ctx)
{

   int duplicated, placedOnes = 0;

   while (placedOnes < countOnes) {
      int p = rand_range(NUM_BITS_GF2X_ELEMENT,
                         BITS_TO_REPRESENT(P),
                         seed_expander_ctx);
      duplicated = 0;
      for (int j = 0; j < placedOnes; j++) if (pos_ones[j] == p) duplicated = 1;
      if (duplicated == 0) {
         pos_ones[placedOnes] = p;
         placedOnes++;
      }
   }
} // rand_circulant_sparse_block

/*---------------------------------------------------------------------------*/
```

Figura 45: Figura 19: Generazioni di sequenze casuali in LEDAcrypt. E' la funzione che verrà modificata per svolgere i test qui riportati.

Il funzionamento è abbastanza semplice: vengono scelti randomicamente dei valori che rappresentano la posizione degli elementi 1, controllando che non ci siano ripetizioni. Il numero di elementi 1 corrisponde alla variabile countOnes. Per modificare quindi la generazione delle chiavi in modo da far sì che queste siano forzate ad avere le caratteristiche particolari che interessa studiare si dovrà andare a modificare questa funzione.

Al fine di capire come vengono generate le posizione degli elementi non-nulli, è opportuno analizzare infine la funzione

static int rand_range(const int n, const int logn, AES_XOF_struct *seed_expander_ctx)

di chiarata nello stesso file. Quello che serve sapere è che essa restitusice un valore uniformemente distribuito tra 0 ed n-1 (incluso).

seed_expander_ctx in ingresso è un seme che serve a far sì che dandone uno particolare in ingresso si ri-ottenga deterministicamente dei particolari valori e si possa quindi ricostruire la matrice.

Tramite questo procedimento è generata la chiave pubblica pk e la chiave privata sk,che corrisponde alla coppia di matrici {**H** , **Q**}.

Q è generata in particolare tramite

void generateQsparse(POSITION_T pos_ones[N0][M], AES_XOF_struct *keys_expander)

che utilizza un meccanismo simile per generare la posizione casuale degli elementi 1. Tuttavia ricordiamo che in questa implementazione la matrice **Q** è stata definita come una matrice identità (M = 1) e non è di particolare importanza.

```
-------------------------------------------------------
| LEDAkem (QC-LDPC based key encapsulation mechanism) |
-------------------------------------------------------
|                                                     |
|                CATEGORY:......     1                |
|                      N0:......     2                |
|                       P:...... 14939(b)             |
|               Rate(K/N):......     0.5              |
|                                                     |
| H circ. block weight DV:......    77                |
| Q circ. block weight M0:......     1                |
|                      M1:......     0                |
|                                                     |
|      number of errors T:......   196                |
|                                                     |
|        private key size:......    24(B) =  0.0(KiB) |
|        public  key size:......  1872(B) =  1.0(KiB) |
|   encapsulated key size:......  1872(B) =  1.8(KiB) |
|                                                     |
-------------------------------------------------------
```

Figura 46: Parametri di LEDAcrypt come impostati per le simulazioni, riportati nella finestra di comando durante l'esecuzione del codice.

In figura 46 viene mostrato come si presenta l'esecuzione del codice di LEDAcrypt nella finestra di comando, e sono mostrati anche i parametri come sono stati impostati per le simulazioni i cui risultati sono mostrati in questo lavoro. Si può vedere in particolare che M0=1 ed M1=1.

9.1.2 LEDAcrypt: Codice C per la generazione del vettore di errore

Un'altra funzione di fondamentale importanza è

rand_circulant_blocks_sequence(err,NUM_ERRORS_T, &niederreiter_encap_key_expander);

in
.../LEDAcrypt-master/Reference_Implementation/KEM/library/gf2x_arith_mod_xPplusOne.c

Sebbene sia implementata in maniera leggermente diversa, essa opera in maniera simile a rand_circulant_sparse_block e si basa allo stesso modo su rand_range().

Nel codice in esame il compito svolto da questa funzione è quello di definire il vettore di errore **e,** indicato dalla variabile err. Il peso di **e** è invece indicato dalla variabile NUM_ERRORS_T .

9.1.3 LEDAcrypt: Codice C per la cifratura

La cifratura viene eseguita tramite la funzione

encrypt_niederreiter(syndrome, pk, err);

```
void encrypt_niederreiter(DIGIT syndrome[],                 // 1  polynomial
                          const publicKeyNiederreiter_t *const pk,
                          const DIGIT err[])                // N0  polynomials
{
   int i;
   DIGIT saux[NUM_DIGITS_GF2X_ELEMENT];

   memset(syndrome, 0x00, NUM_DIGITS_GF2X_ELEMENT*DIGIT_SIZE_B);

   for (i = 0; i < N0-1; i++) {
      gf2x_mod_mul(saux,
                   pk->Mtr+i*NUM_DIGITS_GF2X_ELEMENT,
                   err+i*NUM_DIGITS_GF2X_ELEMENT
                  );
      gf2x_mod_add(syndrome, syndrome, saux);
   }  // end for
   gf2x_mod_add(syndrome, syndrome, err+(N0-1)*NUM_DIGITS_GF2X_ELEMENT);
} // end encrypt_niederreiter
```

Figura 47: Codice C per la cifratura usato in LEDAcrypt.

Il codice calcola il prodotto tra la chiave pubblica pk in ingresso e la versione trasposta del vettore di errore **e** rappresentato dalla variabile err.

9.1.4 LEDAcrypt: Codice C per la decifratura

La riga che esegue la decifratura e restituisce una variabile booleana che indica se questa è andata a buon fine o meno è :

ok = decrypt_niederreiter(decrypted_err,sk,syndrome);

Il codice che viene eseguito quando la funzione è richiamata è abbastanza complesso, e implementa l'algoritmo di decifratura come descritto precedentemente alla sezione 3.1.

Vengono generate nuovamente le matrici **H** e **Q**, ma questo avviene utilizzando lo stesso seme usato in precedenza, e quindi le matrici saranno le medesime.

In decrypt_niederreiter() la verifica che la decifratura sia avvenuta con successo è effettuata tramite il comando

decryptOk = bf_decoding(err, (const POSITION_T (*)[dv]) HtrPosOnes, (const POSITION_T (*)[M]) QtrPosOnes, privateSyndrome);

In cui viene richiamata una funzione che ricopre un ruolo centrale ed implementa il decoding tramite l'algoritmo di Bit-Flipping ("bf") .

int bf_decoding(DIGIT out[], const POSITION_T HtrPosOnes[N0][dv], const POSITION_T QtrPosOnes[N0][M], DIGIT privateSyndrome[])

si trova in

.../LEDAcrypt-master/Reference_Implementation/KEM/library/bf_decoding.c

Riassumendo, si può dire che essa comprende tre fasi principali :

- Calcolo del peso della sindrome e determinazione della soglia.
- Calcolo della correlazione con una matrice **Q** completa.
- Bit-flipping basato sulla correlazione.

Il processo viene iterato per un certo numero di volte, indicato dalla variabile ITERATIONS_MAX, il cui valore è definito in .../LEDAcrypt-master/Reference_Implementation/KEM/include/bf_decoding.h

e di default è pari a

#define ITERATIONS_MAX (15)

9.2 BIKE-1 v3: CODICE C

Il codice qui usato è la versione chiamata BIKE_Additional_Implementation.2020.02.09 [3].

La funzione principale che esegue il test di BIKE è

.../BIKE-master/tests/fixed_seed_test.c

Esso esegue un certo numero di prove (in questo lavoro 10000) di Incapsulamento e Decapsulamento della chiave e resistuisce il numero di errori avvenuti, dai quali si può quindi calcolare la DFR.

Le definizione delle variabili più importanti che definiscono l'algoritmo sono dei DEFINE e si trovano nel file:

.../BIKE_Additional_Implementation.2020.02.09/BIKE-master/common/
bike_defs.h

```
#elif(LEVEL == 3)
#  define R_BITS 24821
#  define dv     103
#  define T1     199
#  define THRESHOLD_COEFF0 15.932
#  define THRESHOLD_COEFF1 0.0052936
```

Dove dv è il numero di elementi 1 per ogni riga/colonna delle matrici circolanti e T1 è il peso del vettore di errore, mentre `THRESHOLD_COEFF` sono definiti come descritto nel capitolo 4.

Le funzioni principali per eseguire il test si trovano in .../BIKE-master/kem.c

9.2.1 BIKE v3: Codice C per la generazione delle chiavi

int crypto_kem_keypair(OUT unsigned char *pk, OUT unsigned char *sk, uint64_t N_FIXED_ONES)

Richiama generate_sparse_rep() per generare sia $\mathbf{e_0}$ ed $\mathbf{e_1}$ che $\mathbf{H_0}$ ed $\mathbf{H_1}$.

9.2.2 BIKE v3: Codice C per la cifratura

Viene usata per la cifratura la funzione

int crypto_kem_enc(OUT unsigned char * ct, OUT unsigned char * ss, IN const unsigned char *pk)

che richiama encrypt(), la quale richiama function_h(). Quest'ultima funzione è di fondamentale importanza, perché genera i vettori $\mathbf{e_0}$ ed $\mathbf{e_1}$.

INLINE ret_t function_h(OUT split_e_t *splitted_e, IN const r_t *in0, IN const r_t *in1)

Usa il paradigma *extract-then-expand* , basato su SHA384 ed AES256-CTR PRNG, per generare e a partire da $(m \cdot f_0, m \cdot f_1)$.Si basa su

ret_t generate_sparse_rep(OUT uint64_t * a, OUT idx_t wlist[], IN const uint32_t weight, IN const uint32_t len, IN const uint32_t padded_len,IN OUT aes_ctr_prf_state_t *prf_state)

in

.../BIKE-master/prf/sampling.c

Essa genera dei valori casuali invocando get_rand_mod_len() finché non sono stati estratti tanti elementi 1 quanto è il peso desiderato della matrice. Sarà quindi questa la funzione ad essere modificata in modo da forzare la creazione in modo che non sia del tutto casuale ma generi elementi con le caratteristiche che ci interessa analizzare.

La funzione encrypt() implementata poi in maniera efficiente le operazioni vettoriali binarie come descritte necessarie per la cifratura, come già descritte nel paragrafo 3.2.

9.2.3 BIKE v3: Codice C per il calcolo delle soglie

La funzione che calcola le soglie di decodifica è get_threshold() che si trova in .../BIKE-master/decode/decode.c

```
_INLINE_ uint8_t
get_threshold(IN const syndrome_t *s)
{
  bike_static_assert(sizeof(*s) >= sizeof(r_t), syndrome_is_large_enough);

  const uint32_t syndrome_weight = r_bits_vector_weight((const r_t *)s->qw);

  // The equations below are defined in BIKE's specification:
  // https://bikesuite.org/files/round2/spec/BIKE-Spec-Round2.2019.03.30.pdf
  // Page 20 Section 2.4.2
  const uint8_t threshold =
      THRESHOLD_COEFF0 + (THRESHOLD_COEFF1 * syndrome_weight);

  DMSG("    Thresold: %d\n", threshold);
  return threshold;
}
```

Figura 48: BIKE v3: Codice C per il calcolo delle soglie di decodifica.

Dove le variabili THRESHOLD_COEFF0 e THRESHOLD_COEFF1 sono dichiarati come DEFINE in .../BIKE-master/common/bike_defs.h

9.2.4 BIKE v3: Codice C per la decifratura

int crypto_kem_dec(OUT unsigned char * ss, IN const unsigned char *ct, IN const unsigned char *sk)

Probabilmente la decifratura è la parte più complessa e sostanziosa di BIKE. L'implementazione delle funzioni necessarie si trova in

.../BIKE_Additional_Implementation.2020.02.09/BIKE-master/decode/decode.c

In questo lavoro la decodifica non sarà modificata rispetto a quella proposta dagli autori di BIKE e perciò non è necessario approfondire il suo funzionamento più di quanto fatto in precedenza.

9.3 BIKE V4: Codice C

Di BIKE v4 è stata presa la Reference Implementation. Il codice è genralmente semplice e leggibile nonostante l'alto grado di ottimizzazione.

Le funzioni prinicipali

- int crypto_kem_keypair(OUT unsigned char *pk, OUT unsigned char *sk)
- int crypto_kem_enc(OUT unsigned char *ct,OUT unsigned char *ss,IN const unsigned char *pk,IN const unsigned char *sk)
- int crypto_kem_dec(OUT unsigned char *ss, IN const unsigned char *ct, IN const unsigned char *sk)

Sono definite in kem.c

La variabile pk rappresenta la chiave pubblica, ct il testo cifrato (ciphertext), ed ss il segreto condiviso (shared secret).

La generazione del vettore di errore avviene attraverso una funzione di hash SHA384 e quindi AES256-CTR PRNG per generare il vettore di errore e a partire dal messaggio m; questa funzione è implementata in

INLINE status_t functionH(OUT uint8_t * e, IN const uint8_t * m)

Perciò è stata quest'ultima funzione ad essere modificata in modo che generasse vettori con il tipo di composizione da analizzare.

La funzione invocata per generare sequenze sparse è

status_t generate_sparse_rep(OUT uint8_t * r, IN const uint32_t weight, IN const uint32_t len,IN OUT aes_ctr_prf_state_t *prf_state)

definita in sampling.c, che per la generazione casuale invoca

status_t get_rand_mod_len(OUT uint32_t* rand_pos, IN const uint32_t len, IN OUT aes_ctr_prf_state_t* prf_state)

che si trova nello stesso file e che genera un valore casuale compreso tra 0 e len-1, in base ad un seme in ingresso.

10 BIBLIOGRAFIA

[1] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi e P. Santini, LEDAcrypt: Low-dEnsity parity-check - Specification revision 2.5 – March, 2020. [Online]. Available: https://www.ledacrypt.org/documents/LEDAcrypt_spec_latest.pdf

[2] Nicolas Aragon et al. , BIKE: Bit Flipping Key Encapsulation - Round 2 Submission. [Online]. Available: https://bikesuite.org/files/round2/spec/BIKE-Spec-Round2.2019.03.30.pdf

[3] Nicolas Aragon et al. , BIKE:Bit Flipping Key Encapsulation - Submission for Round 3 Consideration. [Online]. Available: https://bikesuite.org/files/v4.0/BIKE_Spec.2020.05.03.1.pdf

[4] R. G. Gallager, Low-Density Parity-Check Codes. PhD thesis, M.I.T., 1963.

[5] McEliece, Robert J. (1978), A Public-Key Cryptosystem Based On Algebraic Coding Theory. DSN Progress Report. 44: 114–116.

[6] H. Niederreiter (1986), Knapsack-type cryptosystems and algebraic coding theory. Problems of Control and Information Theory. Problemy Upravlenija i Teorii Informacii. 15: 159–166.

[7] Nicolas Sendrier and Valentin Vasseur. About low DFR for QC-MDPC decoding. In Jintai Ding and Jean-Pierre Tillich, editors, PQCrypto 2020, volume 12100 of LNCS, pages 20–34. Springer, 2020.

[8] Nir Drucker, Shay Gueron, and Dusan Kostic. QC-MDPC decoders with several shades of gray. In Jintai Ding and Jean-Pierre Tillich, editors, PQCrypto 2020, volume 12100 of LNCS, pages 35–50. Springer, 2020.

[9] Nicolas Sendrier and Valentin Vasseur. On the decoding failure rate of QC-MDPC bit-flipping decoders. In Jintai Ding and Rainer Steinwandt, editors, PQCrypto 2019, volume 11505 of LNCS, pages 404–416, Chongquing, China, May 2019. Springer.

[10] Julia Chaulet, Étude de cryptosystèmes à clé publique basés sur les codes MDPC quasi-cycliques . Thèse de doctorat, University Pierre et Marie Curie, March 2017.

[11] Nir Drucker, Shay Gueron, Dusan Kostic, and Edoardo Persichetti. On the applicability of the Fujisaki-Okamoto transformation to the BIKE KEM. IACR Cryptology ePrint Archive, 2020.

[12] Aiden Price and Joanne Hall, A Survey on Trapping Sets and Stopping Sets, CoRR, abs/1705.05996, 2017. [Online]. Available: https://arxiv.org/pdf/1705.05996.pdf

[13] S. Laendner and O. Milenkovic, “Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes”. 2005 Intern. Conf. on Wireless Networks, Commun. and Mobile Computing, vol. 1, pp. 630–635, 2005

[14] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi e P. Santini, «LEDAcrypt: Low-dEnsity parity-check coDe-bAsed cryptographic systems». 30 March 2019. [Online]. Available: https://www.ledacrypt.org/official/comments/2019/07/24/Latest_Round2_Implementation.html

[15] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, LEDAcrypt-KEM and LEDAcrypt-PKC website. https://www.ledacrypt.org/.

[16] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, Ledakem: A post-quantum key encapsulation mechanism based on QC-LDPC codes. In T. Lange andR. Steinwandt, editors,Post-Quantum Cryptography, LNCS, pages 3–24. SpringerInternational Publishing, Cham, 2018.

[17] M. Baldi, M. Bodrato, and F. Chiaraluce, A new analysis of the McEliece cryp-tosystem based on QC-LDPC codes. InSecurity and Cryptography for Networks,volume 5229 ofLNCS, pages 246–262. Springer Verlag, 2008

[18] Christof Zalka, Grover’s quantum searching algorithm is optimal. Phys. Rev. A, 60:2746–2751, October 1999.

[19] M. Finiasz and N. Sendrier, “Security bounds for the design of code-based cryptosystems”. in Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings, 2009, pp. 88–105.

[20] Computer Security Resource Center, https://csrc.nist.gov/Projects/Post-Quantum-Cryptography