Università degli Studi di Milano
Facoltà di Scienze e Tecnologie
*Corso di Laurea in Informatica*

# ANALISI CRITICA DEI CIFRARI SIMON E SPECK

**Relatore:** Prof. Andrea VISCONTI

Tesi di:
Lorenzo ROSSI
Matricola: 792211

Anno Accademico 2014-2015

# Prefazione

Lo scopo di questa tesi è analizzare due famiglie di cifrari a blocchi di recente presentazione, Simon e Speck. Entrambi i cifrari sono stati già stati analizzati sotto diversi punti di vista da parte della comunità degli studiosi, tuttavia rimangono ancora ampi margini da esaminare prima di poter esprimere un giudizio sulla loro sicurezza.

Nello specifico, in questa tesi, una volta studiato lo stato dell'arte, ci si è occupati di sottoporre una primitiva per ognuna delle due famiglie ad un attacco puramente algebrico, utilizzando come punto di partenza l'attacco descritto da Courtois et al. in [18]. In particolare, sono stati analizzati Simon64/128 (Simon su blocchi da 64 bit e 128 bit di chiave) e Speck64/128 (Speck su blocchi da 64 bit e 128 bit di chiave). Sono stati attaccati i primi quattro round di Simon64/128 e un solo round di Speck64/128. I diversi risultati mettono in luce le profonde differenze tra le due famiglie di cifrari.

## Organizzazione della tesi

La tesi è organizzata come segue:

- nel Capitolo 1 vengono presentate le famiglie di cifrari Simon e Speck e l'attuale stato della loro analisi da parte della comunità scientifica
- nel Capitolo 2 vengono presentati gli attacchi algebrici ai cifrari Simon64/128 e Speck64/128
- nel Capitolo 3 vengono discussi i risultati e le loro conclusioni

# Ringraziamenti

Innanzitutto vorrei ringraziare il mio relatore, Prof. Andrea Visconti, per i preziosi spunti e le correzioni necessarie durante tutto questo lavoro e per l'ottimismo con cui ha risposto ai problemi che ci siamo trovati ad affrontare.

Grazie alla mia famiglia per avermi supportato in questi anni di studi, per avermi sostenuto quando gli esami non andavano bene e per avermi sopportato negli ultimi, intensi, mesi di questo lavoro. In particolare grazie a mio padre per tutto il tempo che ha dedicato a rileggere questa tesi e a correggere il mio italiano.

Poi, e qui la lista si fa lunga, vorrei ringraziare Agnese, AF, AL, AM, ADS, Ambra, Dario, FC, FS, FeBa, FisioF, GG, GV, Giulia, PS, SP, SB, VA, VS e tutti gli altri nomi, sigle e soprannomi raccolti durante quella meravigliosa avventura iniziata sotto la pioggia di piazza Piola l'aprile 2013, tutti quelli che ci hanno seguito, piazza per piazza, conferenza per conferenza, articolo per articolo.

Tutti i miei compagni di università: l'AulettaComputerClub, Biondo, Elena, Giulio, Ivano, Leonardo Michele, Mura e Pive. Anche se non abbiamo mai dato un singolo esame assieme, grazie anche a Keru per il prezioso aiuto con LaTeX.

E per ultimo, ma solo perché so che, ora che sei arrivato a fondo pagina, sarai offeso per non essere ancora stato citato, un grazie anche a Stefo, senza il cui fondamentale contributo non mi sarei mai nemmeno interessato all'informatica, figuriamoci laurearmici.

**Lorenzo Rossi**
Milano, Luglio 2015

# Indice

**Prefazione** **ii**

**Ringraziamenti** **iii**

**1 Le famiglie di cifrari Simon e Speck** **1**
1.1 Crittografia lightweight . . . . . 1
1.2 Simon e Speck . . . . . 2
1.2.1 Simon . . . . . 3
1.2.2 Speck . . . . . 4
1.3 Attacchi noti a Simon e Speck . . . . . 7
1.3.1 Crittoanalisi differenziale . . . . . 8
1.3.2 Crittoanalisi algebrica . . . . . 10
1.3.3 Attacchi side-channel . . . . . 11

**2 Attacco algebrico di Simon64/128 e Speck64/128** **13**
2.1 Strumenti utilizzati . . . . . 13
2.2 L'attacco algebrico a Simon64/128 . . . . . 14
2.2.1 Rappresentazione algebrica di Simon64/128 . . . . . 15
2.2.2 Risultati dell'attacco . . . . . 18
2.3 L'attacco algebrico a Speck64/128 . . . . . 22
2.3.1 Rappresentazione algebrica di Speck64/128 . . . . . 23
2.3.2 Risultati dell'attacco . . . . . 27

**3 Analisi dei risultati, conclusioni e sviluppi futuri** **28**
3.1 Analisi dei risultati . . . . . 28
3.2 Conclusioni . . . . . 30
3.2.1 Sviluppi futuri . . . . . 32

# Elenco delle tabelle


1 I parametri di Simon . . . . . 3
2 I parametri di Speck . . . . . 4
3 Risultati degli attacchi differenziali su Simon. . . . . 9
4 Risultati degli attacchi differenziali su Speck. . . . . 10
5 Risultati degli attacchi differenziali impossibili su Simon . . . . . 11
6 Numero di monomi per equazione nel sistema di Simon64/128 . . . 17
7 Tempi di attacco medi e occupazione di memoria per Simon64/128 . 21
8 Divisione dei tempi di attacco di Simon64/128 . . . . . 28

# Elenco delle figure


1 Schema del round di Simon . . . . . 5
2 Schema del round di Speck . . . . . 6
3 Distribuzione dei tempi di analisi algebrica del primo round di Simon64/128 . . . . . 18
4 Distribuzione dei tempi totali di attacco al primo round di Simon64/128 19
5 Distribuzione dei tempi di analisi algebrica del secondo round di Simon64/128 . . . . . 20
6 Distribuzione dei tempi totali di attacco per il secondo round Simon64/128 . . . . . 21
7 Distribuzione dei tempi di analisi algebrica del terzo round di Simon64/128 . . . . . 22
8 Distribuzione dei tempi totali di attacco del terzo round di Simon64/128 23
9 Distribuzione dei tempi di analisi algebrica del quarto round di Simon64/128 . . . . . 24
10 Distribuzione dei tempi complessivi di analisi del quarto round di Simon64/128 . . . . . 25
11 Tempi medi di attacco per Simon64/128 . . . . . 29

# Capitolo 1

# Le famiglie di cifrari Simon e Speck

## 1.1 Crittografia lightweight

Gli algoritmi crittografici attualmente in uso sono nati per venire incontro alle necessità dell'era desktop e mal si adattano alle esigenze delle nuove tecnologie, più pervasive, in cui è uno scenario normale trovare numerose periferiche dalle risorse estremamente limitate che si scambiano messaggi attraverso una rete. Questo tipo di periferiche sta diventando sempre più diffuso nella vita di tutti i giorni, si trova spesso a trattare i nostri dati personali e quindi, sebbene le loro risorse siano limitate, non è possibile trascurarne gli aspetti di sicurezza. Questa esigenza ha portato alla nascita di una nuova branca della crittografia, detta lightweight, che si occupa di realizzare primitive crittografiche leggere.

Negli ultimi anni il mondo della crittografia si è dedicato con interesse crescente allo sviluppo di nuove primitive crittografiche leggere - per citare due esempi abbiamo la famiglia di cifrari Salsa20 proposta da [9] e il cifrario PRESENT pubblicato da [13]. Nessuna delle primitive presentate sinora ha ottenuto il consenso necessario a elevarsi a standard per questa categoria. A pesare, almeno in parte, su questa freddezza sono le considerazioni sull'ottimizzazione, un punto cardine dello sviluppo di una primitiva crittografica lightweight: gli accorgimenti che possono contribuire a ridurre il peso di un'implementazione hardware hanno un effetto negativo sul peso di un'implementazione software.

Il concetto di peso o di efficienza di un'implementazione nell'ambito della crittografia lightweight richiede una trattazione più approfondita per l'importanza che ha avuto nelle scelte progettuali dietro a Simon e Speck. All'inizio di [7] vengono presentate alcune considerazioni che hanno influito sulla progettazione di entrambi i cifrari: riassumendole rapidamente, l'obiettivo degli autori non è stato tanto progettare due algoritmi che fossero estremamente efficienti rispetto a *una singola* applicazione, quanto ottenere degli algoritmi che potessero comportarti bene su un

numero di applicazioni *quanto più ampio possibile*. Per questo motivo la principale caratteristica di entrambe le famiglie è la flessibilità. Per ottenere questo risultato è stato messo in secondo piano il throughput, ritenuto un requisito meno pressante per le applicazioni lightweight, limitandosi a porre come vincolo che esso restasse almeno pari a 12 kbps su di un processore a 100 kHz. La scelta fatta in [7] di presentare due famiglie di cifrari distinte e parametrizzate è da vedersi proprio nell'ottica di ottenere la massima flessibilità e di fornire cifrari adeguati a quante più situazioni possibili.

Dal punto di vista della sicurezza, al momento non è ancora emerso nulla che assomigli a uno standard di sicurezza su cui misurare i cifrari lightweight. Alcuni attacchi che richiedono vaste quantità di dati cifrati e in chiaro potrebbero non essere possibili perché esiste la possibilità che un apparecchio che monta, per esempio, un cifrario con blocchi da 16 bit non produca dati sufficienti neanche nel corso di tutta la sua vita. D'altra parte, nell'era del *pervasive computing* è assai più probabile che un attaccante sia in grado di raggiungere fisicamente l'apparecchiatura sotto attacco: quindi gli attacchi side-channel potrebbero costituire una minaccia assai più grave e immediata di altre tipologie. Nonostante queste considerazioni siano già diffuse e, almeno in parte, intuitive, non è ancora emerso un preciso modello degli attacchi a cui una primitiva lightweight dovrebbe saper resistere.

## 1.2 Simon e Speck

Nel Giugno 2013 un gruppo di ricercatori affiliati alla National Security Agency statunitense ha presentato alla comunità crittografica due famiglie di cifrari a blocchi [7] pensate per andare incontro alle richieste di sicurezza in ambienti dalle risorse estremamente limitate. Sebbene entrambe le famiglie siano state ottimizzate quanto più possibile sia dal punto di vista hardware (numero di porte logiche necessarie per implementare il cifrario su un circuito dedicato) sia da quello software (dimensioni del codice e consumo di memoria) molte scelte che favoriscono l'ottimizzazione hardware penalizzano quella software e viceversa, portando così a produrre due famiglie estremamente leggere [1] ma specializzate: SIMON è stato ottimizzato per l'uso su hardware, mentre SPECK per quello software.

Entrambe le famiglie sono state pensate per un uso flessibile, indipendente da un'applicazione specifica, e sono quindi parametrizzate; complessivamente ogni famiglia contiene dieci cifrari a blocchi.

[1] In [7] vengono presentati costi inferiori ad altri algoritmi lightweight per tutte le dimensioni di plaintext e chiave, sia dal punto di vista hardware che da quello software, mentre uno studio dell'implementazione di entrambi i cifrari su microcontrollori è stato effettuato sempre dagli stessi autori in [8]

| Cifrario | $n$ | $m$ | $z$ | Round |
|---|---|---|---|---|
| SIMON32/64 | 16 | 4 | $z_0$ | 32 |
| SIMON48/72 | 24 | 3 | $z_0$ | 36 |
| SIMON48/96 | 24 | 4 | $z_1$ | 36 |
| SIMON64/96 | 32 | 3 | $z_2$ | 42 |
| SIMON64/128 | 32 | 4 | $z_3$ | 44 |
| SIMON96/96 | 48 | 2 | $z_2$ | 52 |
| SIMON96/144 | 48 | 3 | $z_3$ | 54 |
| SIMON128/128 | 64 | 2 | $z_2$ | 68 |
| SIMON128/192 | 64 | 3 | $z_3$ | 69 |
| SIMON128/256 | 64 | 4 | $z_4$ | 72 |

Tabella 1: I parametri di SIMON

### 1.2.1 Simon

SIMON è una famiglia di cifrari a blocchi costruiti sul modello Feistel. Ogni blocco di SIMON è costituito da 2 parole di n bit ciascuna, mentre la chiave è composta da m parole di lunghezza n, ogni cifrario della famiglia SIMON viene identificato come SIMON2N/NM, con n che può assumere i valori 16, 24, 32, 48, 64 e m che può assumere valori in dipendenza da n (vedi Tabella 1). In base a m e n, inoltre, variano il numero di round e il vettore di n bit z, utilizzato per generare le chiavi di round come indicato nella 4. Una rappresentazione del funzionamento di un singolo round di SIMON è data nella Figura 1.2.1.

Lo stato del round i-esimo di SIMON è rappresentato da due parole di n bit ciascuna e sarà indicato con $L_i||R_i$. La funzione Feistel che genera lo stato successivo utilizza tre operazioni:

- XOR bit a bit $\oplus$
- AND bit a bit $\wedge$
- Shift circolare a sinistra $S^j$ di j bit

Lo stato del round i-esimo è calcolato a partire da $L_{i-1}||R_{i-1}$ e dalla chiave di round $k_i$:

$$L_i = R_{i-1} \oplus F(L_{i-1}) \oplus k_i \tag{1}$$

$$R_i = L_{i-1} \tag{2}$$

Mentre la funzione F usata nel calcolo di $L_i$ è definita come:

| **Cifrario** | **n** | **m** | $\alpha$ | $\beta$ | **Round** |
|---|---|---|---|---|---|
| SPECK32/64 | 16 | 4 | 7 | 2 | 22 |
| SPECK48/72 | 24 | 3 | 8 | 3 | 22 |
| SPECK48/96 | 24 | 4 | 8 | 3 | 23 |
| SPECK64/96 | 32 | 3 | 8 | 3 | 26 |
| SPECK64/128 | 32 | 4 | 8 | 3 | 27 |
| SPECK96/96 | 48 | 2 | 8 | 3 | 28 |
| SPECK96/144 | 48 | 3 | 8 | 3 | 29 |
| SPECK128/128 | 64 | 2 | 8 | 3 | 32 |
| SPECK128/192 | 64 | 3 | 8 | 3 | 33 |
| SPECK128/256 | 64 | 4 | 8 | 3 | 34 |

Tabella 2: I parametri di SPECK

$$F(x) = S^1(x) \wedge S^8(x) \oplus S^2(x) \tag{3}$$

Per i primi m round di cifratura vengono usate le m parole della chiave iniziale come chiavi di round. Le restanti chiavi vengono generate a partire da queste, utilizzando funzioni diverse a seconda del numero di parole della chiave iniziale:

$$K_{\mathrm{i+m}} = K_{\mathrm{i}} \oplus Y \oplus S^{-1}(Y) \oplus c \oplus (z_{\mathrm{j}})_{\mathrm{i}} \tag{4}$$

$$Y = \begin{cases} S^{-3}(K_{\mathrm{i+1}}) & \text{se } m = 2 \\ K_{\mathrm{i+1}} \oplus S^{-3}(K_{i+2}) & \text{se } m = 3 \\ K_{\mathrm{i+1}} \oplus S^{-3}(K_{i+3}) & \text{se } m = 4 \end{cases} \tag{5}$$

Dove i valori c e z sono due array di bit dai valori costanti utilizzati per prevenire attacchi basati sulla rotazione delle chiavi. Mentre c è una costante di valore `0xfff...fc`, z varia a seconda della versione di SIMON utilizzata: le possibilità sono elencate nella Tabella 1 (maggiori dettagli sui valori di z e su come vengono generati possono essere trovati in [7]).

### 1.2.2 Speck

SPECK è una famiglia di cifrari progettata per fornire prestazioni eccellenti sia in un'implementazione hardware che in una software, ma ottimizzata espressamente per l'uso su microcontrollori. Ogni membro della famiglia SPECK utilizza due parole di lunghezza n bit e una master key di lunghezza $m * n$ bit.La notazione utilizzata per indicare le diverse versioni di SPECK è la stessa utilizzata per SIMON.

La funzione di round di SPECK utilizza le seguenti operazioni:

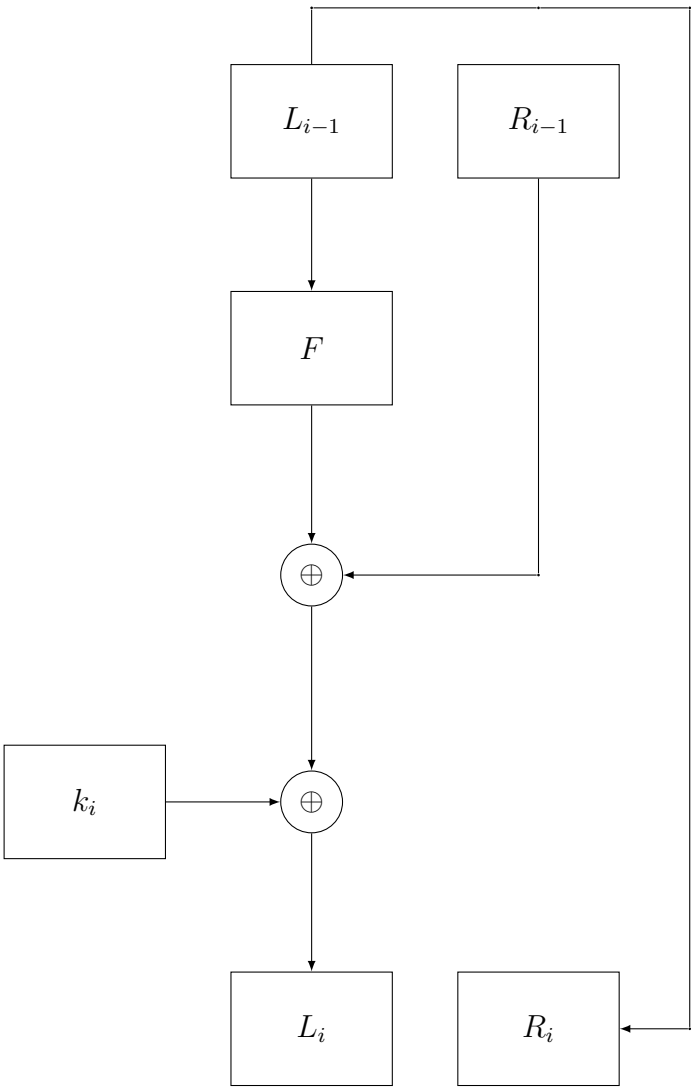


Figura 1: Schema del round di SIMON

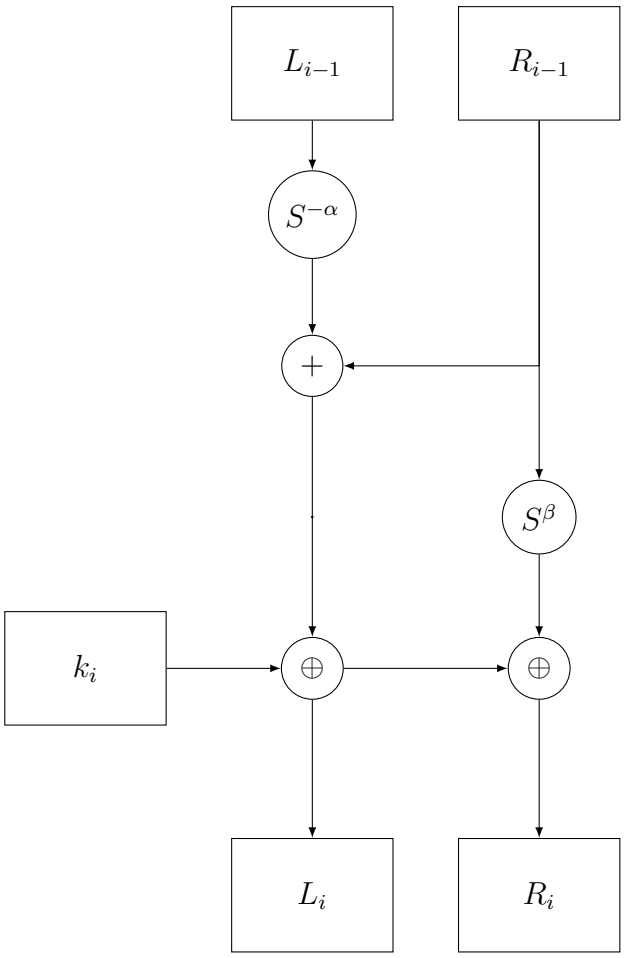


Figura 2: Schema del round di Speck

- XOR bit a bit, $\oplus$
- somma modulo $2^n$, $+$
- Shift circolare a sinistra $S^j$ di j bit

La funzione di round R di SPECK, al contrario di quella di SIMON, opera su entrambe le parole dello stato contemporaneamente:

$$R_k(x, y) = (((S^{-\alpha}(x) + y) \oplus k), (S^{\beta}(y) \oplus (S^{-\alpha}(x) + y) \oplus k)) \tag{6}$$

Dove $\alpha$ e $\beta$ assumono i valori indicati nella Tabella 2.La funzione inversa, necessaria per decrittare, utilizza la sottrazione modulo $2^n$ invece della somma ed è definita come:

$$R_k^{-1}(x, y) = ((S^{\alpha}((x \oplus y) - S^{-\beta}(x \oplus y)), S^{-\beta}(x \oplus y)) \tag{7}$$

Per la generazione delle chiavi di round SPECK fa uso di un vettore di appoggio $\ell$, composto da parole di n bit e così definito:

$$\ell_{i+m-1} = (k_i + S^{-\alpha}(\ell_i)) \oplus i \tag{8}$$

Mentre la chiave di round $k_i$ viene calcolata come segue

$$k_{i+1} = S^{\beta}(k_i) \oplus \ell_{i+m-1} \tag{9}$$

Al contrario di quanto detto per SIMON, tutte le varianti di SPECK calcolano le chiavi di round allo stesso modo. I valori iniziali di $\ell$ e k sono dati dalla master key **K**, le cui m-1 parole costituiscono gli elementi iniziali di $\ell$, mentre l'ultima parola è $k_0$. In questo senso è possibile vedere **K** come un array di m elementi: $[\ell_{m-2} \dots \ell_0, k_0]$

## 1.3 Attacchi noti a Simon e Speck

Poiché entrambi i cifrari sono stati presentati alla comunità solo di recente, non sono attualmente impiegati in nessuna applicazione pratica né ne esiste, almeno al momento della stesura di questa tesi, una implementazione di riferimento per gli studiosi. Gli attacchi noti finora, quindi, sono costruiti a partire da uno studio teorico e basati su implementazioni *ad hoc* dei cifrari. A titolo di esempio si riportano gli indirizzi di due repository: in `https://github.com/GSongHashrate/SimonSpeck/` si trova un'implementazione in C++ del solo SIMON utilizzata durante l'attacco di [18], in `https://github.com/bozhu/NSA-ciphers` si trovano i sorgenti Python sia di SIMON sia di SPECK utilizzati in questa tesi (mentre una terza implementazione, per microcontrollori a 8 bit, è stata discussa in [8]). È importante ricordare che tutte

queste implementazioni sono state realizzate a scopo di studio e nessuna di essere è da considerarsi sicura o adatta a un uso pratico.

La maggior parte degli attacchi portati è basata su tecniche di crittoanalisi differenziale, uno solo si basa sulla crittoanalisi algebrica, così come è stato presentato un solo attacco side-channel, su una prima prova di implementazione hardware di SIMON. La maggior parte delle tipologie di attacco portata su una delle due famiglie ha mostrato di essere efficace anche quando portata sull'altra.

In [7] gli autori affermano di aver progettato sia SIMON sia SPECK seguendo il modello di minaccia tradizionale per lo sviluppo di primitive crittografiche, cioè assumendo che l'attaccante sia in grado di cifrare e decifrare arbitrariamente vaste quantità di dati. Sono state prese precauzioni per proteggersi nel caso di un avversario capace di cambiare singoli bit della chiave e gli autori dichiarano di aver puntato a realizzare due cifrari a prova di attacchi *related-key*. Non sono stati presi in considerazione, per ammissione degli stessi autori, scenari in cui i due cifrari possono essere utilizzati come funzioni di hash, né attacchi *open-key*. I risultati delle analisi effettuate sui cifrari dai progettisti stessi prima della pubblicazione di [7] non sono stati pubblicati.

Al momento nessuno studio ha ancora trovato una vulnerabilità che permetta di attaccare tutti i round di SIMON o di SPECK.

### 1.3.1 Crittoanalisi differenziale

La crittoanalisi differenziale è un tipo di attacco a *chosen plaintext* identificato per la prima volta durante la progettazione di DES [15] e poi pubblicata da Biham e Shamir negli anni 90 [11]. Da allora è stata impiegata con successo nell'attacco di numerose primitive crittografiche differenti. L'idea alla base di tutti gli attacchi differenziali è quella di tracciare la propagazione di differenze tra input attraverso un certo numero di round e di individuare differenze prevedibili tra gli output con una certa probabilità. Queste differenze, note come proprietà differenziali, possono essere utilizzate per ottenere parti di una chiave di round (normalmente la prima o l'ultima). Usando numerose coppie di plaintext, scelti perché rispettino la differenza in input, e tentando contemporaneamente tutti i candidati per la chiave di round sotto attacco ci si aspetta che la sotto-chiave corretta emerga più spesso di quelle errate, permettendo all'attaccante di riconoscerla.

I due studi fondamentali per quanto riguarda la crittoanalisi differenziale di SIMON sono quelli presentati da Alkhzaimi e Lauridsen [6] e, parallelamente, da Abed et al. [1]. Entrambi gli studi si sono concentrati sull'analisi di SIMON e hanno analizzato tutti e dieci i cifrari che compongono la famiglia arrivando a risultati estremamente simili. Un ulteriore risultato, pubblicato nel maggio 2015 da Mourouzis et al.. [23]

mostra come sia possibile attaccare fino a 26 round di Simon64/128 utilizzando delle proprietà differenziali troncate.

Di ogni cifrario sono stati attaccati, con questa tecnica, circa la metà dei round con percentuali di successo comprese tra il 60% e il 90% circa. Nella tabella 3 sono riportati i round attaccati da [1] e [6], per ulteriori dettagli sulle percentuali di successo si rimanda agli articoli citati.

| **Cifrario** | **Round** | | |
|---|---|---|---|
| | **Totali** | **Abed** | **Alkhzaimi** |
| Simon32/64 | 32 | 18 | 16 |
| Simon48/72 | 36 | 19 | 18 |
| Simon48/96 | 36 | 19 | 18 |
| Simon64/96 | 42 | 26 | 24 |
| Simon64/128 | 44 | 26 | 24 |
| Simon96/96 | 52 | 35 | 29 |
| Simon96/144 | 54 | 35 | 29 |
| Simon128/128 | 68 | 46 | 40 |
| Simon128/192 | 69 | 46 | 40 |
| Simon128/256 | 72 | 46 | 40 |

Tabella 3: Risultati degli attacchi differenziali su Simon.

Per quanto riguarda il tempo e la memoria necessari a eseguire gli attacchi presentati, l'occupazione di memoria è sostanzialmente identica in entrambi i casi, mentre i tempi di attacco presentati da Alkhzaimi e Lauridsen sono nettamente inferiori a quelli presentati da Abed et al., soprattutto al crescere di m e n arrivando a richiedere poco più del 50% dei tempi di Abel et al.. per attaccare Simon128/256. Per comprendere l'origine di queste differenze è importante notare che i due studi seguono approcci molto diversi all'analisi differenziale, soprattutto per quanto riguarda la tecnica di individuazione delle proprietà differenziali.

Per quanto riguarda invece la crittoanalisi di Speck, al momento sono state presentate analisi da Abed et al. [2], I. Dinur [19] e da Biryukov et al. [12] con risultati paragonabili a quelli ottenuti su Simon. Mentre [2] e [19] hanno attaccato tutti e dieci i cifrari della famiglia Speck, [12] ha attaccato soltanto i primi 5, al momento i risultati migliori sono stati presentati in [19] che, anche quando non supera il numero di round attaccati negli altri due studi, ottiene complessità di tempo e di memoria molto inferiori (per i dettagli degli attacchi si vedano gli studi citati). Nella Tabella 4 sono riportati i risultati ottenuti nei tre attacchi.

| Cifrario | Round | | | |
|---|---|---|---|---|
| | **Totali** | **Abed** | **Dinur** | **Biryukov** |
| Speck32/64 | 22 | 10 | 14 | 11 |
| Speck48/72 | 22 | 12 | 14 | 12 |
| Speck48/96 | 23 | 12 | 15 | 12 |
| Speck64/96 | 26 | 15 | 18 | 16 |
| Speck64/128 | 27 | 15 | 19 | 16 |
| Speck96/96 | 28 | 16 | 16 | n.a. |
| Speck96/144 | 29 | 16 | 17 | n.a. |
| Speck128/128 | 32 | 16 | 17 | n.a. |
| Speck128/192 | 33 | 18 | 18 | n.a. |
| Speck128/256 | 34 | 18 | 19 | n.a. |

Tabella 4: Risultati degli attacchi differenziali su Speck.

**Differenziale impossibile**

Un attacco con differenziale impossibile punta, al contrario di un attacco differenziale, a trovare due differenze $\Delta_{\text{in}}$ e $\Delta_{\text{out}}$ non compatibili tra di loro, a ricostruire quali bit della chiave influenzano le due differenze e a utilizzare questa informazione per filtrare lo spazio delle chiavi.

Anche in questo caso i due studi più completi sugli attacchi a Simon sono [6] e [1]. Entrambi hanno ottenuto una probabilità di successo molto vicina a 1, sebbene su un numero di round estremamente ridotto. Assieme ai risultati questa volta vengono riportate nella Tabella 5 anche le complessità di tempo, molto più elevate e, questa volta, estremamente significative per quanto riguarda l'efficacia dell'attacco (i casi che presentano tempi di attacco superiori a quelli di una ricerca per esaustione sono indicati con †).

In questo caso le prestazioni sono molto meno positive dell'attacco differenziale, al punto che si può dire che non esiste un attacco con differenziale impossibile né per Simon96/96 né per Simon128/128. Bisogna tuttavia tenere conto del fatto che entrambi i gruppi di studiosi hanno costruito questo attacco per lasciare un margine d'errore inferiore all'1% e che uno studio più approfondito delle proprietà differenziali di Simon porterà con molta probabilità a una variazione di questi risultati.

Speck non è ancora stato analizzato sotto questo punto di vista.

### 1.3.2 Crittoanalisi algebrica

Basata su una frase comunemente attribuita a C. Shannon, "rompere un buon cifrario dovrebbe richiedere tanto sforzo quanto risolvere un sistema di equazioni simultanee in un vasto numero di variabili di un tipo complesso", la crittoanalisi algebrica utilizza

| Cifrario | Round | | | Tempi | |
|---|---|---|---|---|---|
| | Totali | Abed | Alkhzaimi | Abed | Alkhzaimi |
| Simon32/64 | 32 | 13 | 14 | $2^{50.1}$ | $2^{44.183}$ |
| Simon48/72 | 36 | n.a. | 15 | n.a. | $2^{69.079}$ |
| Simon48/96 | 36 | 15 | 15 | $2^{53.0}$ | $2^{69.079}$ |
| Simon64/96 | 42 | n.a. | 16 | n.a. | $2^{91.986}$ |
| Simon64/128 | 44 | 17 | 16 | $2^{71.95}$ | $2^{91.986}$ |
| Simon96/96 | 52 | n.a. | 19 | n.a. | $2^{129.780}$† |
| Simon96/144 | 54 | 20 | 19 | $2^{111.0}$ | $2^{129.780}$ |
| Simon128/128 | 68 | n.a. | 22 | n.a. | $2^{187.527}$† |
| Simon128/192 | 69 | n.a. | 22 | n.a. | $2^{187.527}$ |
| Simon128/256 | 72 | 25 | 22 | $2^{195.0}$ | $2^{187.527}$ |

Tabella 5: Risultati degli attacchi differenziali impossibili su Simon

un modello matematico del cifrario sotto attacco e software matematico (tipicamente SAT-solver) per cercare di ottenere la chiave. Una prima applicazione pratica di questa tecnica è stata data da Courtois e Bard nel loro attacco a DES [16] ed è stata ripresa più volte su numerosi cifrari a blocchi.

Attualmente l'unica analisi algebrica di Simon è quella presentata da Courtois et al.. in [18], che sono riusciti ad attaccare fino a 10 round di Simon64/128 utilizzando una combinazione di attacco algebrico e differenziale e indovinando alcuni bit della chiave. Un elemento interessante di questo tipo di attacchi è il numero di coppie plaintext-ciphertext necessarie a eseguire l'attacco: durante l'attacco di [18] sono state utilizzate un massimo di 10 coppie. Nello stesso articolo sono stati presentati risultati che si fermavano all'ottavo (dalle 2 alle 6 coppie utilizzate) e al nono round (6 o 7 coppie utilizzate).

Al momento Speck non è stato sottoposto a crittoanalisi algebrica.

### 1.3.3 Attacchi side-channel

Gli attacchi side-channel utilizzano debolezze di una particolare implementazione del cifrario per ottenere informazioni sulla chiave utilizzata. Non esistendo, attualmente, implementazioni ufficiali né di Simon né di Speck non bisogna aspettarsi di trovare molti attacchi di questo tipo: l'unico presentato, ad oggi, alla comunità mostra la vulnerabilità di una implementazione FPGA di Simon che segua pedissequamente le specifiche presentate in [7]. L'attacco presentato in [10], nello specifico, mostra come si possa monitorare il consumo di energia di un'implementazione hardware di Simon per ottenere la differenza di Hamming tra i diversi stati del cifrario e, conoscendo

questa e il plaintext, effettuare ipotesi sulla chiave. Assieme all'attacco vengono presentate delle contromisure per neutralizzarlo nelle prossime implementazioni.

# Capitolo 2

# Attacco algebrico di Simon64/128 e Speck64/128

Un attacco algebrico è costituito da due passaggi distinti [16]:

1. Rappresentazione del cifrario con un sistema di equazioni in $GF(2)$
2. Soluzione del sistema

Normalmente, si considera la prima parte quella più critica e che richiede maggiore sforzo da parte dell'analista, mentre la seconda può essere automatizzata (come viene fatto solitamente) utilizzando un SAT-solver. Una buona rappresentazione è critica per la riuscita dell'attacco e questo passo non sempre è banale. In [16] viene suggerito, come metodo per ottenere una buona rappresentazione del cifrario, di seguirne fedelmente un'implementazione circuitale.

Diversamente dall'attacco presentato di seguito, l'attacco descritto in [18] aggiunge un passaggio intermedio tra il primo e il secondo: il passaggio del sistema di equazioni generato inizialmente a ElimLin [17]. Nell'attacco presentato in questa tesi ElimLin non è stato adottato per due motivi, uno di ordine progettuale - si è scelto di mantenere l'attacco il più semplice possibile per avere una chiara idea di quali elementi dei due cifrari concorressero al risultato - e uno di ordine pratico: al momento non esistono implementazioni open source e/o accuratamente documentate di ElimLin, rendendo estremamente difficile l'utilizzo corretto dell'algoritmo e la comprensione dei suoi risultati. L'impatto di questa scelta sui risultati dell'attacco presentato sarà discusso nel Capitolo 3.

## 2.1 Strumenti utilizzati

Il software che esegue entrambi gli attacchi è stato scritto in Python. La scelta è stata fatta durante la preparazione dell'attacco a Simon64/128 perché l'utilizzo di

Python permetteva di appoggiarsi agli strumenti forniti da SageMath [27].

Come risolutore SAT, dopo aver valutato diverse possibilità, si è scelto di utilizzare CryptoMiniSAT 2.9.7 [26], un'estensione di MiniSAT [20] pensata per affrontare i problemi SAT che emergono tipicamente in ambito crittografico (come viene illustrato in [26], affrontando i problemi SAT di stampo crittografico con tecniche risolutive apposite è possibile ottenere risultati migliori rispetto a quelli ottenuti utilizzando strategie generali). Sia MiniSAT che CryptoMiniSAT hanno riportato risultati molto positivi quando confrontati con altri risolutori (si veda, ad esempio, il risultato ottenuto da MiniSAT durante la SAT Competition 2005 [21] o quello ottenuto da CryptoMiniSAT durante la SAT-Race 2010 [24]). Al momento della stesura di questa tesi sono disponibili due versioni di CryptoMiniSAT, la 2 e la 4, mentre è stata annunciata l'uscita, nel prossimo futuro, di CryptoMiniSAT 5. Sebbene CryptoMiniSAT 4 fosse disponibile fin dall'inizio di questo lavoro, si è optato per CryptoMiniSAT 2 perché era già stato integrato all'interno di SageMath. Sia CryptoMiniSAT che MiniSAT sono completamente open source.

SageMath [27] è un framework per lo studio della matematica open source rilasciato per la prima volta nel 2005. La comunità degli sviluppatori è composta da gruppi di studiosi che mantengono e scrivono i moduli che utilizzano nel loro lavoro. Molti dei moduli sono costituiti da wrapper che permettono di utilizzare software esterno all'interno di SageMath. Nello specifico per gli attacchi algebrici si è fatto ricorso al modulo Boolean Polynomials il cui cuore è costituito da un wrapper attorno a PolyBoRi [14] a cui sono stati aggiunti un'interfaccia per CryptoMiniSAT e un convertitore di equazioni da forma algebrica a CNF, a cura di M. Albrecht.

PolyBoRi è uno strumento di analisi di anelli di polinomi booleani basato sullo studio delle basi di Gröbner. Il cuore di PolyBoRi è scritto in C++ e si basa su due librerie, CUDD [25] e M4RI [4] (M4RI viene anche utilizzata da sola all'interno di SageMath).

Questi strumenti sono già stati utilizzati con successo per attacchi algebrici da M. Albrecht in [3, 5].

Tutti gli attacchi descritti qui di seguito sono stati eseguiti su un computer con processore AMD Athlon II 630 quad-core a 2.8 GHz e 8.0 GB di memoria RAM.

## 2.2 L'attacco algebrico a Simon64/128

SIMON64/128 è già stato analizzato tramite crittoanalisi algebrica in [18] con risultati molto diversi da quelli presentati: è quindi necessario mettere in evidenza le differenze tra i due attacchi. Innanzitutto l'attacco qui presentato si basa esclusivamente sulla crittoanalisi algebrica, mentre quello presentato in [18] è stato costruito combinando tecniche di crittoanalisi algebrica con tecniche differenziali e tentando diversi bit della chiave. Questo ha permesso a Courtois et al. di ottenere la chiave fino al decimo

round di SIMON64/128 ma, d'altra parte, non ha mostrato quale sia la resistenza del cifrario alla sola crittoanalisi algebrica né permette un confronto immediato tra SIMON e SPECK su questo aspetto, per via della composizione degli attacchi (come viene mostrato più avanti, SPECK risponde agli attacchi algebrici in maniera molto diversa da SIMON). Perciò è probabile che, sebbene l'attacco di [18] abbia avuto su SIMON un successo molto maggiore di quello presentato qui, quest'ultimo possa dare un contributo importante all'analisi di *entrambi* i cifrari.

Sia l'attacco portato a SIMON64/128 che quello portato a SPECK64/128 utilizza una sola coppia P-C, scelta casualmente, e non cerca di indovinare alcun bit della chiave.

### 2.2.1 Rappresentazione algebrica di Simon64/128

Come indicato in [16] un buon metodo per rappresentare un cifrario per questo tipo di attacco è seguire pedissequamente una sua implementazione hardware. Mancando, tuttavia, un'implementazione di riferimento, si è scelta come punto di partenza - come d'altra parte era già stato fatto in [18] - la descrizione data in [7]. Modificando quindi le formule 1 e 2 perché si riferiscano ai singoli bit, si ottiene che il bit j-esimo della parte sinistra del round i-esimo vale:

$$L_{\mathrm{i}}^{\mathrm{j}} = R_{\mathrm{i-1}}^{\mathrm{j}} \oplus L_{\mathrm{i-1}}^{\mathrm{j+1}} \wedge L_{\mathrm{i-1}}^{\mathrm{j+8}} \oplus L_{\mathrm{i-1}}^{\mathrm{j+2}} \oplus k_{\mathrm{i}}^{\mathrm{j}} \tag{10}$$

Mentre per la parte destra dello stato è sufficiente applicare la formula 2 un bit alla volta per ottenere:

$$R_{\mathrm{i}}^{\mathrm{j}} = L_{\mathrm{i-1}}^{\mathrm{j}} \tag{11}$$

A partire dalle formule 10 e 11 viene ottenuto il sistema che rappresenta tutti i round di SIMON64/128. Nelle descrizioni dei round (Formule 12, 14 e 15) viene utilizzata la seguente notazione:

- $x_i$ indica il bit i-esimo del plaintext
- $t_{i,j}$ indica il bit i-esimo dell'output del round j
- $k_i$ indica il bit i-esimo della chiave

Il primo round viene quindi rappresentato come segue:

$$\begin{cases} t_{0,1} = x_{32} \oplus x_1 \wedge x_8 \oplus x_2 \oplus k_0 \\ t_{1,1} = x_{33} \oplus x_2 \wedge x_9 \oplus x_3 \oplus k_1 \\ \dots \\ t_{31,1} = x_{63} \oplus x_0 \wedge x_7 \oplus x_1 \oplus k_{31} \\ t_{32,1} = x_0 \\ t_{33,1} = x_1 \\ \dots \\ t_{63,1} = x_{31} \end{cases} \tag{12}$$

Sostituendo ai bit del plaintext l'output del round precendente, il secondo round può venire, inizialmente, descritto così:

$$\begin{cases} t_{0,2} = t_{32,1} \oplus t_{1,1} \wedge t_{8,1} \oplus t_{2,1} \oplus k_{32} \\ t_{1,2} = t_{33,1} \oplus t_{2,1} \wedge t_{9,1} \oplus t_{3,1} \oplus k_{33} \\ \dots \\ t_{31,2} = t_{63,1} \oplus t_{0,1} \wedge t_{7,1} \oplus t_{1,1} \oplus k_{63} \\ t_{32,2} = t_{0,1} \\ \dots \\ t_{63,2} = t_{31,1} \end{cases} \tag{13}$$

Tuttavia, come già è stato detto sopra, non è possibile effettuare alcuna ipotesi sui valori di $t_{0,1} \dots t_{63,1}$, perché tutto ciò che sappiamo su di loro è che sono polinomi definiti nella Formula 12. Sostituendo quindi $t_{0,1} \dots t_{63,1}$ con i polinomi che li descrivono il secondo round di SIMON64/128 può essere così descritto:

$$\begin{cases} t_{0,2} = (x_0) \oplus (x_{33} \oplus x_2 \wedge x_9 \oplus x_3 \oplus k_1) \wedge (x_{40} \oplus x_9 \wedge x_{16} \oplus x_{10} \oplus k_8) \oplus \\ \qquad \oplus (x_{34} \oplus x_3 \wedge x_{10} \oplus x_4 \oplus k_2) \oplus k_{32} \\ \dots \\ t_{31,2} = (x_{31}) \oplus (x_{32} \oplus x_1 \wedge x_8 \oplus x_2 \oplus k_0) \wedge (x_{39} \oplus x_8 \wedge x_15 \oplus x_9 \oplus k_7) \oplus \\ \qquad \oplus (x_{33} \oplus x_2 \wedge x_9 \oplus x_3 \oplus k_1) \oplus k_{63} \\ t_{32,2} = (x_{32} \oplus x_1 \wedge x_8 \oplus x_2 \oplus k_0) \\ \dots \\ t_{63,2} = (x_{63} \oplus x_0 \wedge x_7 \oplus x_1 \oplus k_{31}) \end{cases} \tag{14}$$

Le parentesi tonde raccolgono quello che prima era un singolo monomio. Possiamo notare come i 5 monomi che descrivono ogni bit del primo round siano diventati 17 al secondo round.

| Round | L | R |
|---|---|---|
| 1 | 5 | 1 |
| 2 | 17 | 5 |
| 3 | 57 | 17 |
| 4 | 189 | 57 |

Tabella 6: Numero di monomi per equazione nel sistema di SIMON64/128

È possibile rappresentare anche il terzo round come è stato fatto nella Formula 13, sostituendo $t_{0,1} \dots t_{63,1}$ con $t_{0,2} \dots t_{63,2}$ e $k_{32} \dots k_{63}$ con $k_{64} \dots k_{96}$, tuttavia questo passaggio viene omesso perché ripetitivo e si dà immediatamente la descrizione del terzo round di SIMON64/128 utilizzando tutti i polinomi.

$$\begin{cases} t_{0,3} = (x_{32} \oplus x_1 \wedge x_8 \oplus x_2 \oplus k_0) \oplus (x_1 \oplus (x_{34} \oplus x_3 \wedge x_{10} \oplus x_4 \oplus k_2) \wedge \\ \wedge (x_{41} \oplus x_{10} \wedge x_{17} \oplus x_{11} \oplus k_9) \oplus (x_{35} \oplus x_4 \wedge x_{11} \oplus x5 \oplus k_3) \oplus \\ \oplus k_{33}) \wedge (x_8 \oplus (x_{41} \oplus x_{10} \wedge x_{17} \oplus x_{11} \oplus k_9) \wedge (x_{48} \oplus x_{17} \wedge x_{24} \oplus \\ \oplus x_{18} \oplus k_{16}) \oplus (x_{42} \oplus x_{11} \wedge x_{18} \oplus x_{12} \oplus k_{10}) \oplus k_{40}) \oplus (x_2 \oplus (x_{35} \oplus \\ \oplus x_4 \wedge x_{11} \oplus x_5 \oplus k_3) \wedge (x_{42} \oplus x_{11} \wedge x_{18} \oplus x_{12} \oplus k_{10}) \oplus (x_{36} \oplus \\ \oplus x_5 \wedge x_{12} \oplus x_6 \oplus k_4) \oplus k_{34}) \oplus k \\ \dots \\ t_{32,3} = (x_0) \oplus (x_{33} \oplus x_2 \wedge x_9 \oplus x_3 \oplus k_1) \wedge (x_{40} \oplus x_9 \wedge x_{16} \oplus x_{10} \oplus k_8) \oplus \\ \oplus (x_{34} \oplus x_3 \wedge x_{10} \oplus x_4 \oplus k_2) \oplus k_{32} \\ \dots \\ t_{63,3} = (x_{31}) \oplus (x_{32} \oplus x_1 \wedge x_8 \oplus x_2 \oplus k_0) \wedge (x_{39} \oplus x_8 \wedge x_15 \oplus x_9 \oplus k_7) \oplus \\ \oplus (x_{33} \oplus x_2 \wedge x_9 \oplus x_3 \oplus k_1) \oplus k_{63} \end{cases} \tag{15}$$

Per motivi di leggibilità non viene riportato il sistema che descrive il quarto round. Vengono invece riportate, nella Tabella 6, le lunghezze dei due tipi di polinomio, L e R, che fanno parte del sistema di ogni round.

Applicando gli stessi criteri utilizzati per ottenere le Formule 10 e 11 anche alla Formula 4, utilizzando direttamente la formula per $m = 4$, si è ottenuta la formula per calcolare i bit della chiave di round:

$$k_i^j = k_{i-4}^j \oplus k_{i-3}^j \oplus k_{j-3}^{i-1} \oplus k_{i-3}^{j-1} \oplus k_{j-3}^{i-2} \oplus c^j \oplus z_3^j \tag{16}$$

Prima di proseguire, è importante effettuare due annotazioni. La prima è che tutte le somme e le sottrazioni agli apici, ovvero quelle che indicano la posizione del bit, sono da considerarsi *modulo*32, sebbene non sia stato espressamente indicato per

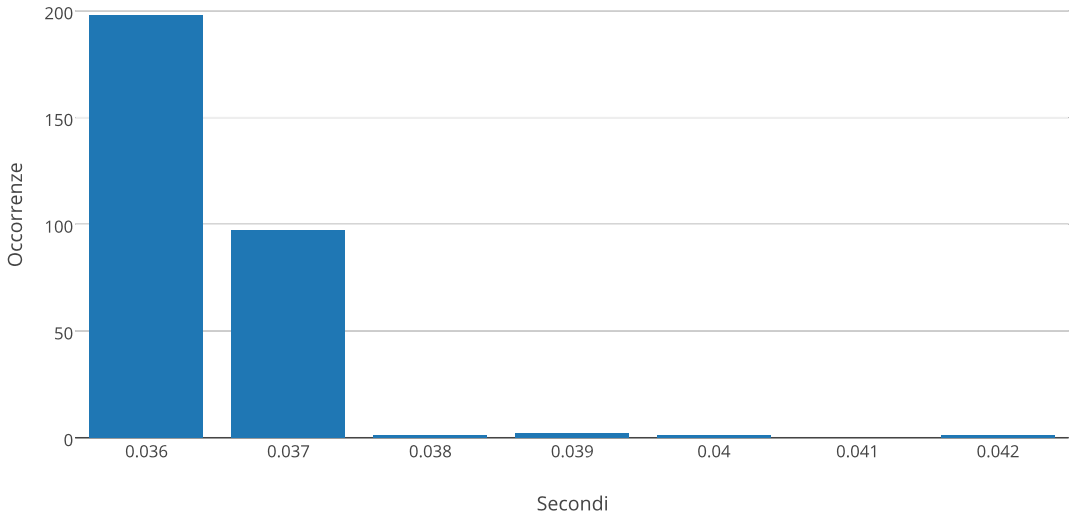


Figura 3: Distribuzione dei tempi di analisi algebrica del primo round di SIMON64/128

motivi di leggibilità. È altrettanto importante notare che la rappresentazione interna degli stati del cifrario, per motivi di comodità di implementazione, ha invertito l'endianess degli stati stessi e, di conseguenza, anche il verso degli shift. Sebbene quest'ultima variazione possa sembrare controintuitiva e fonte di confusione, ha permesso di sfruttare al meglio gli strumenti messi a disposizione da SageMath e dal suo interprete Python.

### 2.2.2 Risultati dell'attacco

L'attacco è riuscito con una percentuale di successo del 100% a recuperare le prime 4 chiavi di round di SIMON64/128. Per ogni round, i dati sono stati ottenuti su un campione di 300 plaintext casuali e i relativi ciphertext, cifrati utilizzando una chiave casuale.

Nella Figura 3 viene mostrata la distribuzione dei tempi necessari a CryptoMiniSAT per analizzare il sistema che descrive SIMON64/128 e ricavarne la chiave. Osservando la figura, si può notare come 295 coppie delle 300 testate siano state analizzate in un tempo compreso tra 0.036 e 0.037 secondi. Il tempo massimo impiegato per la sola analisi è stato di 0.042 secondi. La stessa distribuzione, relativa ai tempi

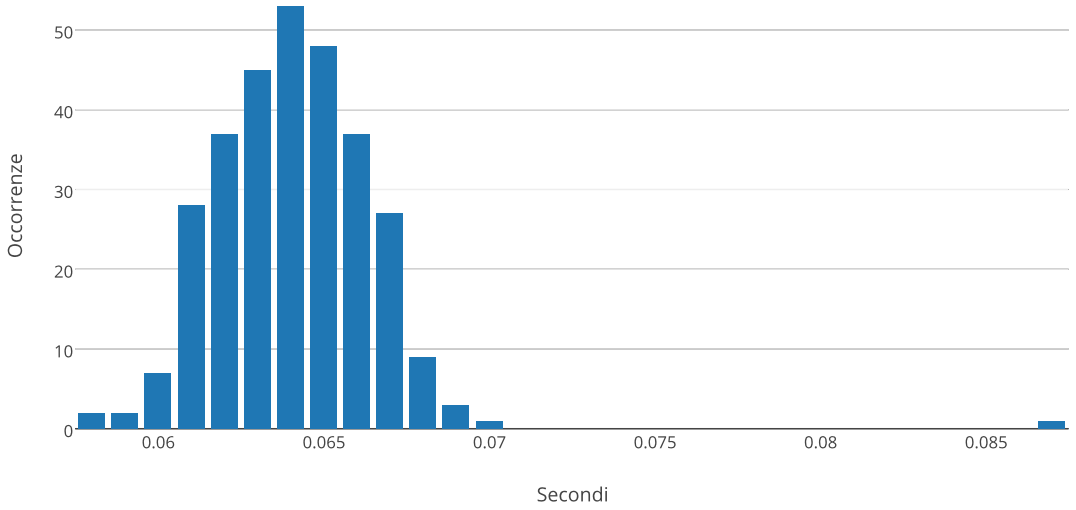


Figura 4: Distribuzione dei tempi totali di attacco al primo round di SIMON64/128

complessivi di attacco, può essere vista nella Figura 4: in questo caso la distribuzione è meno netta, rivelando un picco di occorrenze attorno a 0.063 secondi, ma la distribuzione ricorda molto più una Gaussiana.

Passando dal primo al secondo round i tempi di attacco e di analisi algebrica crescono di pochi centesimi di secondo e cambiano nuovamente distribuzione. Possiamo tuttavia notare, confrontando tra loro le Figure 5 e 6 come le due distribuzioni si assomiglino molto di più: entrambe presentano due picchi molto elevati verso i tempi più ridotti, attorno a cui si raccoglie la quasi totalità dei risultati. Si può supporre che questo dipenda dal fatto che, mentre i tempi di analisi richiesti da CryptoMiniSAT sono cresciuti, i tempi necessari a generare il sistema di equazioni abbiano avuto un incremento nettamente inferiore e, di conseguenza, un minor peso nel determinare i tempi complessivi. Oltre 200 coppie hanno richiesto sia come tempo di analisi sia come tempo complessivo, poco meno di 0.05 secondi.

Analizzando le Figure 7 e 8 che si riferiscono, rispettivamente, ai tempi di analisi e ai tempi complessivi di attacco del terzo round, è possibile vedere come, sebbene i tempi di analisi seguano una distribuzione coerente con quanto apparso in precedenza, i tempi di attacco si dividano su due gruppi distanti tra loro, uno compreso tra 0.2 e 0.25 secondi (188 casi) e uno tra 0.33 e 0.356 secondi (54 casi). Questo comportamento, all'apparenza anomalo, viene analizzato nel Capitolo 3.

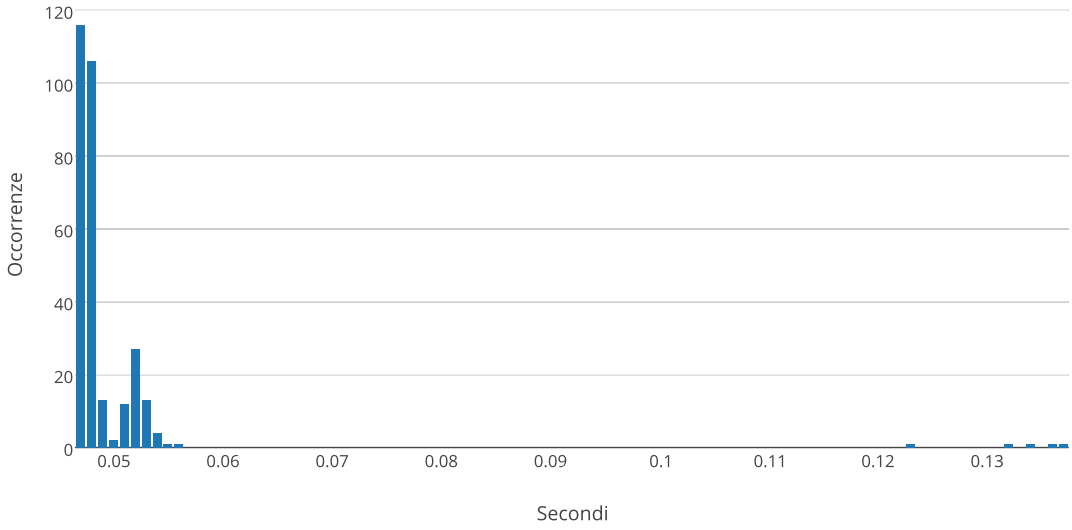


Figura 5: Distribuzione dei tempi di analisi algebrica del secondo round di SIMON64/128

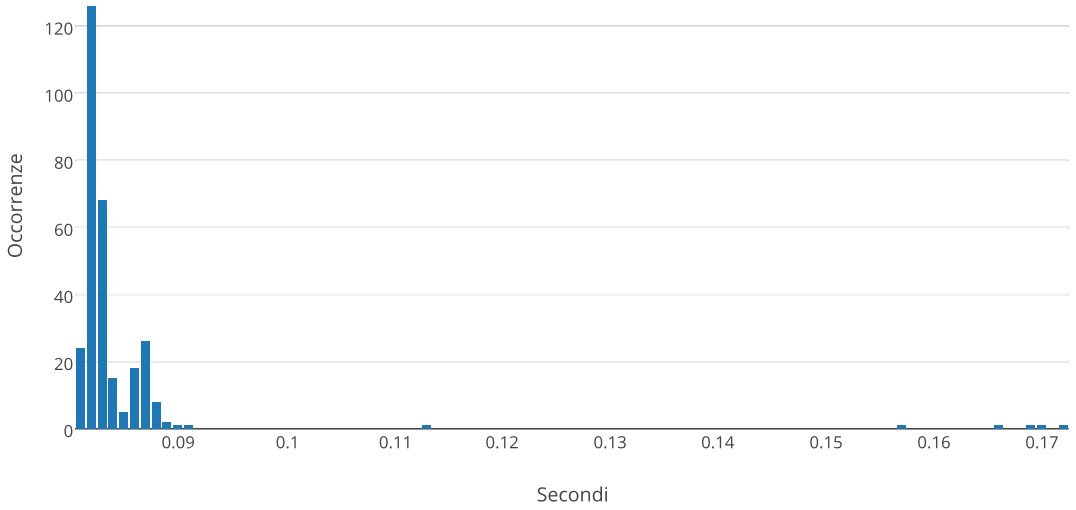


Figura 6: Distribuzione dei tempi totali di attacco per il secondo round SIMON64/128

Nelle Figure 9 e 10 vengono riportati, con una precisione inferiore ai casi precedenti, i tempi ottenuti per l'analisi del quarto round di SIMON64/128. Si è scelto di ridurre la precisione della rappresentazione dai millesimi di secondo ai secondi per adeguarsi all'ordine di grandezza dei risultati (mantenendo la precisione ai millesimi di secondo, così come sono stati rappresentati i dati 298 coppie per grafico utilizzavano tempi diversi e soltanto in un caso per categoria i tempi coincidevano). È possibile notare come, in entrambi i grafici, la distribuzione sia molto simile: è facile notare, osservando la Tabella 7, come il tempo medio impiegato dal SAT-Solver per l'analisi del quarto round sia estremamente vicino al tempo complessivo di attacco; questo dato, e le sue conseguenze, sono trattati con maggior precisione nel Capitolo 3.

| Round | Tempo medio | | Memoria (KB) |
|---|---|---|---|
| | SAT-Solver | Totale | |
| 1 | 0.036 | 0.064 | 1080.0 |
| 2 | 0.050 | 0.085 | 7400.0 |
| 3 | 0.084 | 0.256 | 13544.0 |
| 4 | 26.108 | 26.470 | 19688.0 |

Tabella 7: Tempi di attacco medi e occupazione di memoria per SIMON64/128

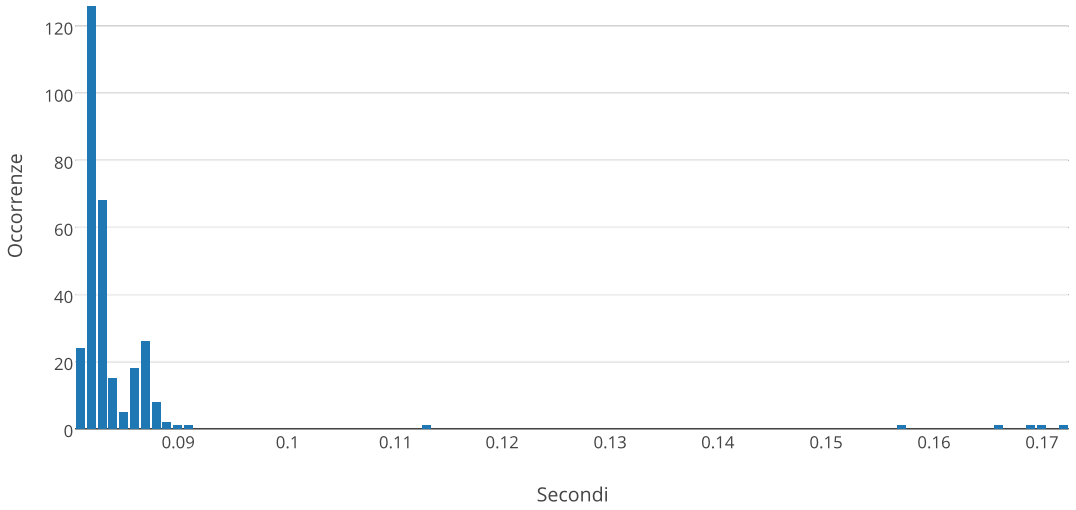


Figura 7: Distribuzione dei tempi di analisi algebrica del terzo round di SIMON64/128

Il consumo di memoria necessario a rappresentare il cifrario, illustrato nella Tabella 7, rimane costante all'interno del round indipendentemente dalle coppie utilizzate. L'occupazione di memoria dell'intero attacco, comprensiva cioè di quanto richiesto sa SageMath e CryptoMiniSAT, è in realtà molto più elevata e difficile da tracciare, dovendo tenere conto dei numerosi processi coinvolti. Sono richiesti, comunque, diversi GB di memoria disponibile per portare a termine l'attacco con successo.

Si è tentato di estendere l'attacco oltre il quarto round, tuttavia le equazioni superano in lunghezza il limite imposto da CryptoMiniSAT, causandone l'arresto. Le cause di questo blocco e le conseguenze che ne sono state tratte, soprattutto a seguito di un confronto con i risultati ottenuti su SPECK, sono discusse più approfonditamente nel Capitolo 3.

## 2.3 L'attacco algebrico a Speck64/128

Mentre SIMON64/128 è stato scelto perché già analizzato con criteri simili a quelli adottati, nessun altro si è ancora occupato di effettuare l'analisi algebrica di un qualsiasi cifrario della famiglia SPECK. Per poter effettuare un parallelo con l'altro

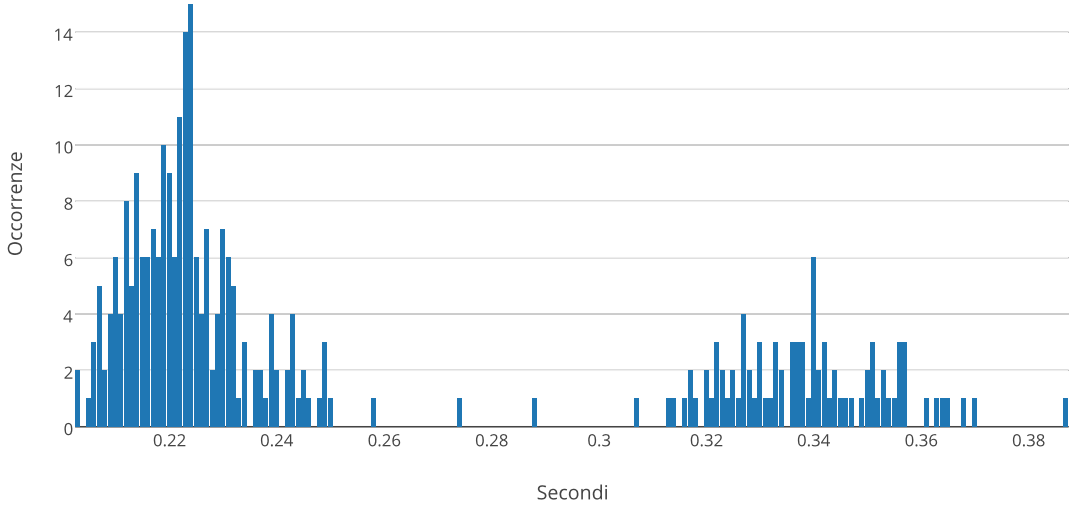


Figura 8: Distribuzione dei tempi totali di attacco del terzo round di SIMON64/128

attacco studiato si è scelto di utilizzare la variante di SPECK con $m = 4$ e $n = 32$, in modo da ottenere blocchi e chiave delle stesse dimensioni.

### 2.3.1 Rappresentazione algebrica di Speck64/128

Come detto sopra, di SPECK non è stata finora effettuata alcuna analisi dal punto di vista algebrico e, come già indicato, non ne esistono implementazioni hardware. Di conseguenza, la rappresentazione presentata sotto è il primo tentativo di esprimere SPECK come sistema di equazioni. Come viene rilevato in [18], il modello adottato per il cifrario influenza grandemente i risultati di un attacco algebrico: è quindi possibile che i risultati qui presentati possano cambiare, anche in maniera drastica, qualora dovesse emergere un metodo più efficace di rappresentare SPECK64/128.

Al contrario di SIMON, SPECK utilizza poche operazioni esprimibili immediatamente sui singoli bit e agisce su entrambe le metà dello stato a ogni round. Si è reso quindi necessario realizzare una rappresentazione più complessa della funzione di round $R(x, y)$ che non una semplice trasposizione ai bit della formula 6 (non essendo la decrittazione di alcun interesse per l'attacco in questione, non è stata effettuata nessuna conversione della 7). Per comodità di lettura e implementazione $R(x, y)$ è stata

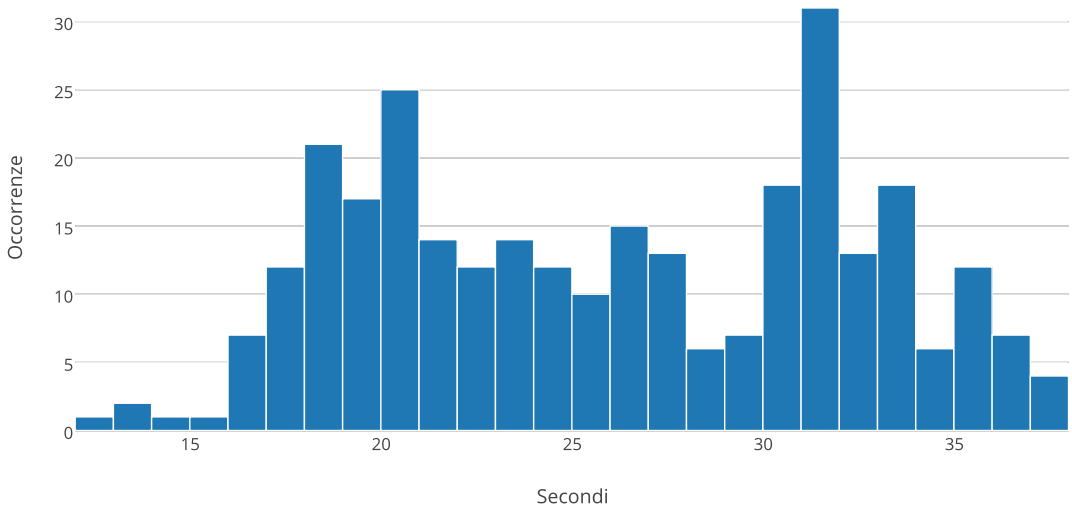


Figura 9: Distribuzione dei tempi di analisi algebrica del quarto round di SIMON64/128

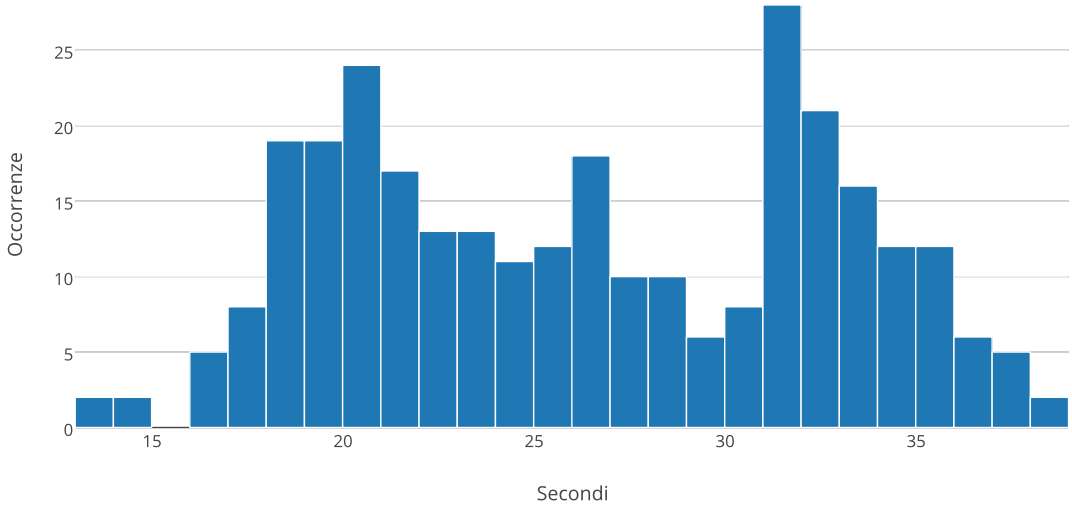


Figura 10: Distribuzione dei tempi complessivi di analisi del quarto round di SIMON64/128

spezzata in due funzioni $RL(x, y)$ e $RR(x, y)$, tali che $R(x, y) = (RL(x, y), RR(x, y))$, definite come segue:

$$RL(x^j, y^j) = x^{j-\alpha} \oplus y^j \oplus r^j \oplus k^j \tag{17}$$

$$RR(x^j, y^j) = RL(x^j, y^j) \oplus y^{j+\beta} \tag{18}$$

Dove $k^j$ è il bit j-esimo della chiave di round e $r^j$ è il riporto della somma modulo 32 tra le due metà dello stato e che viene definito bit per bit come indicato in [22]:

$$r^j = \begin{cases} 0 & \text{se } j = 0 \\ (x^j \wedge y^j) \oplus (x^j \wedge r^{j-1}) \oplus (y^j \wedge r^{j-1}) & \text{altrimenti} \end{cases} \tag{19}$$

Il sistema che descrive il primo round di SPECK64/128, quindi, può essere descritto come segue (tenendo a mente la notazione usata nelle Formule 12, 14 e 15).

$$\begin{cases} t_{0,1} = x_{24} \oplus x_{32} \oplus k_0 \\ t_{1,1} = x_{25} \oplus x_{33} \oplus x_{25} \wedge x_{33} \oplus k_1 \\ \dots \\ t_{32,1} = x_{24} \oplus x_{32} \oplus k_0 \oplus x_{35} \\ t_{1,1} = x_{25} \oplus x_{33} \oplus x_{25} \wedge x_{33} \oplus k_1 \oplus x_{36} \\ \dots \end{cases} \tag{20}$$

A causa della lunghezza del resto, la principale fonte di complessità nella descrizione di SPECK64/128 non vengono riportati i bit più significativi delle due metà dello stato. Infatti, mentre calcolando il riporto della somma un bit alla volta richiede un numero costante di operazioni come definito nella Formula 19, rappresentare quella stessa Formula usando un polinomio richiede un numero di monomi che, per il trentaduesimo bit, può essere formato da miliardi di monomi, come può essere calcolato partendo dalla seguente Formula:

$$lr_i = \begin{cases} 0 & \text{se } i = 0 \\ 2 & \text{se } i = 1 \\ 2 + 2 * (1 + lr_{i-1}) & \text{altrimenti} \end{cases} \tag{21}$$

Come esposto in 1.2.2, la master key $\mathbf{K}$ è composta da $[\ell_2, \ell_1, \ell_0, k_0]$, di conseguenza è necessario calcolare le chiavi di round anche per attaccare round inferiori a m. Nel caso di SPECK oltre alla chiave occorre generare il vettore $\ell$ seguendo la formula 8; come è facile comprendere osservando la 8 e la 9, la generazione del vettore k delle chiavi di round e quella di $\ell$ devono andare di pari passo.

$$\ell_{i-m+1}^{j} = k_i^j \oplus \ell_i^{j-\alpha} \oplus r^j \oplus i^j \tag{22}$$

$$k_i^j = k_i^{j+\beta} \oplus \ell_{i+m-1}^{j} \tag{23}$$

Dove $r^j$ è definito come nella formula 19 e i è il numero del round. È importante notare fin da questo punto la differenza nel rapporto tra gli AND e gli XOR che occorrono per definire un singolo bit dello stato di SPECK rispetto a quelli necessari per un bit di stato di SIMON: il fatto che sia coinvolto un numero elevato (e variabile: gli AND coinvolti a generare $r^j$ aumentano al crescere di j) ha un impatto estremamente negativo sulle prestazioni del SAT-solver.

### 2.3.2 Risultati dell'attacco

Sebbene l'analisi algebrica del primo round di SPECK64/128 sia eseguibile anche con carta e penna, non è stato possibile estendere questo tipo di attacco ai round successivi: come per il quinto round di SIMON64/128 le equazioni che rappresentano lo stato del secondo round di SPECK64/128 superano il limite di lunghezza ammesso da CryptoMiniSAT, bloccando l'esecuzione del programma.

Come già accennato sopra, un cambiamento della rappresentazione di SPECK64/128 potrebbe alzare il numero di round attaccabili esclusivamente tramite crittoanalisi algebrica, così come l'uso di un SAT-solver differente (anche se, visto lo scopo di CryptoMiniSAT, non è da dare per scontato che il cambio di SAT-solver non porti con sé nuove problematiche). Rimane tuttavia certo che la complessità dal punto di vista algebrico di SPECK sia, a parità di m e n, molto più elevata di quella di SIMON.

# Capitolo 3

# Analisi dei risultati, conclusioni e sviluppi futuri

## 3.1 Analisi dei risultati

Come è possibile vedere nella Tabella 7 i tempi di attacco per Simon64/128, pur subendo un incremento massiccio da un round all'altro, si mantengono estremamente contenuti. Sono necessari circa 30 secondi per attaccare quattro round di Simon, senza bisogno di fare ricorso a hardware dedicato. È interessante notare come mentre i tempi di risoluzione da parte del SAT-solver sono in continua crescita, il tempo necessario a eseguire il resto dell'attacco, vale a dire le operazioni di codifica del cifrario, si mantengano pressoché costanti. Inizialmente i tempi sono divisi quasi equamente, ma al crescere dei round la quasi totalità del tempo necessario all'attacco è costituita dal tempo necessario a CryptoMiniSAT per risolvere il sistema.

La divisione dei tempi può essere osservata meglio nella Tabella 8. Nella divisione dei tempi è particolarmente interessante notare il terzo e il quarto round: nel passaggio dal secondo al terzo round i tempi di codifica mostrano un incremento notevole, assai maggiore rispetto ai tempi di analisi, mentre dal terzo al quarto round l'incremento maggiore è quello dei tempi di calcolo da parte CryptoMiniSAT. Questo comportamento è probabilmente causato dal notevole aumento del numero di monomi

| Round | Codifica | SAT-Solver | Totale |
|---|---|---|---|
| 1 | 0.028 | 0.036 | 0.064 |
| 2 | 0.035 | 0.050 | 0.085 |
| 3 | 0.172 | 0.084 | 0.256 |
| 4 | 0.362 | 26.108 | 26.470 |

Tabella 8: Divisione dei tempi di attacco di Simon64/128

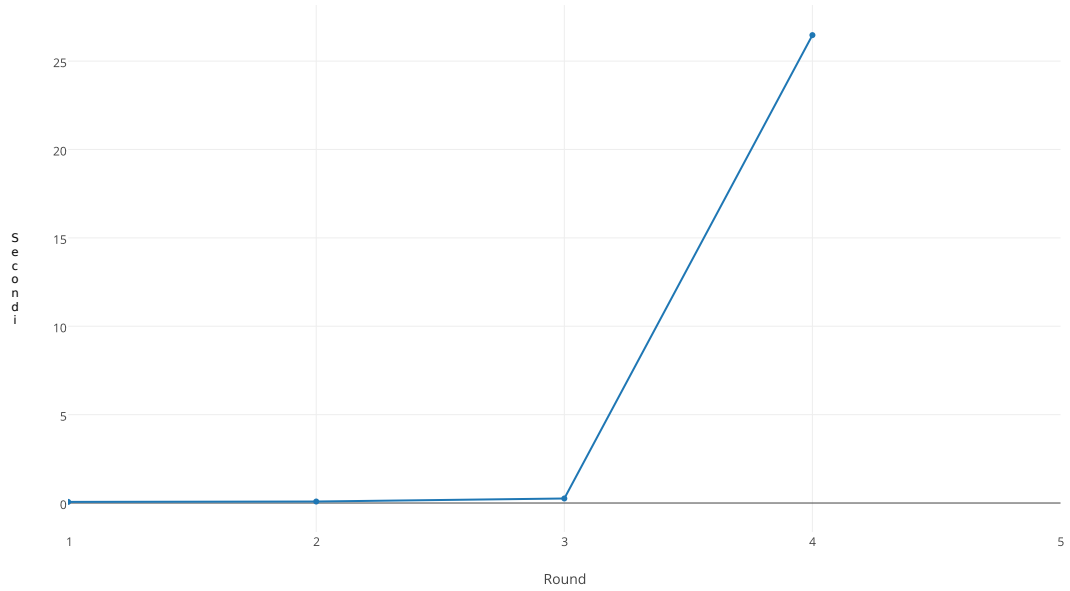


Figura 11: Tempi medi di attacco per Simon64/128

presente nelle equazioni che compongono il sistema che stiamo analizzando: secondo e il terzo round il numero di monomi necessari per calcolare un singolo bit passa da 17 a 57. Nel sistema che stiamo analizzando avremo quindi 4 operazioni AND e 53 operazioni XOR. Ricordiamo al lettore che l'elevato numero di monomi, in generale, richiede lunghi tempi di codifica, ma ad aumentare significativamente i tempi di calcolo sono solo le operazioni AND. Il passaggio dal terzo al quarto round, invece, porta ad aumentare il numero di operazioni per singolo bit a 189, alzando il numero di AND a 13. L'aumento così massiccio di operazioni non lineari richiede molto più sforzo computazionale a CryptoMiniSAT, molto più di quanto non ne sia richiesto a SageMath e PolyBoRi per codificare quelle stesse equazioni.

La situazione per quanto riguarda Speck64/128 è, invece, estremamente diversa. La complessità iniziale è estremamente ridotta, così come per Simon64/128, ma cresce seguendo una curva molto più ripida, passando immediatamente dall'essere risolvibile anche senza aiuto di CryptoMiniSAT all'essere impossibile da risolvere persino per CryptoMiniSAT.

Sebbene il numero di round attaccati sia differente, entrambi gli attacchi terminano allo stesso modo e, in un certo senso, nello stesso momento. In entrambi i casi

la complessità aumenta fino al punto da impedire la risoluzione automatica nel momento in cui le chiavi di round vengono calcolate e non sono più tratte direttamente dalla master key. Non bisogna però farsi trarre in inganno da questa coincidenza. Se è vero che, in entrambi i casi, la derivazione della chiave aumenta la complessità delle equazioni (quello che prima era un monomio viene sostituito da un polinomio composto quasi esclusivamente da valori incogniti), è altrettanto vero che a causare il blocco di CryptoMiniSAT non è il numero di operazioni generiche, ma il numero di operazioni non lineari. Osservando le formule 16 e 23 (tenendo conto, in quest'ultima, anche delle formule per calcolare $\ell$ e il riporto, rispettivamente 22 e 19) è evidente come la derivazione della chiave in Speck64/128 aumenti il numero di operazioni non lineari, mentre in Simon64/128 vengono introdotte solo operazioni lineari. Nel caso di Simon64/128 è il semplice accumulo di complessità dato dall'aumentare dei round (ovvero di stati intermedi non noti all'attaccante e su cui non si possono fare previsioni) a determinare, a un certo punto, il fallimento dell'attacco. In questo senso possiamo comprendere meglio l'apporto della crittoanalisi differenziale all'attacco presentato in [18]: la possibilità di effettuare previsioni sugli stati intermedi, conoscendo le proprietà differenziali e la probabilità che esse siano verificate, contribuisce in maniera significativa a ridurre questo livello di complessità.

## 3.2 Conclusioni

Il risultato più incoraggiate ottenuto durante questo lavoro è stato, senza dubbio, quello relativo all'attacco a Simon64/128. Sebbene quattro round siano, effettivamente, molto pochi, sono esattamente m round consecutivi. Per come è stata pensata l'intera famiglia Simon questo ha due conseguenze fondamentali:

- È possibile trascurare i rapporti fra i bit delle chiavi per un massimo di m round consecutivi.
- Conoscendo $K_{i+m} \dots K_{i+1}$ è possibile ricavare $K_i$ partendo dalla Formula 4 e risolvendola per $K_i$ come segue:

$$K_i = K_{i+m} \oplus Y \oplus S^{-1}(Y) \oplus c \oplus (z_j)_i \tag{24}$$

  dove Y è definito seguendo la Formula 5. Iterando questo processo è possibile arrivare fino ad ottenere $[K_0 \dots K_m - 1]$ ovvero la master key $\mathbf{K}$.

Inoltre, per via del primo punto esposto, un attaccante che sia in grado di leggere due stati intermedi qualsiasi a 4 round di distanza è in grado di ripetere questo stesso attacco. Non avendo, come già detto in precedenza, a disposizione alcuna implementazione reale su cui effettuare questo tipo di analisi non si può dire, al momento,

quanto sia probabile che si verifichi uno scenario del genere. Certamente questa possibilità non va trascurata nel momento in cui si progetta un'implementazione di SIMON orientata a un uso diverso dall'analisi.

Dai risultati ottenuti su SPECK64/128 o, meglio, dall'impossibilità di ottenere risultati positivi si possono trarre conclusioni estremamente interessanti sulla progettazione di SPECK: come già accennato sopra, infatti, è l'elevato numero di operazioni non lineari contenute sia nella funzione di round che nel calcolo della chiave a rendere impossibile un attacco di questo tipo. Sebbene future rappresentazioni che siano in grado di trovare delle correlazioni lineari non immediatamente visibili possano, successivamente, modificare l'efficacia di un attacco algebrico, è estremamente improbabile che un'analisi algebrica di SPECK come quella presentata su SIMON in [18] possa mai ottenerne lo stesso grado di successo. Se analizziamo le funzioni che descrivono SIMON e quelle che descrivono SPECK così come sono state riportate, rispettivamente, in 1.2.1 e 1.2.2 possiamo notare come la sola fonte di non linearità in SIMON sia all'interno della funzione di round (e quindi sia possibile, utilizzando più coppie P-C legate da una proprietà differenziale, ridurre la non-linearità come fatto in [18]), mentre in SPECK le operazioni non lineari vengono aggiunte tanto dalla funzione di round che da quella di calcolo della chiave.

Un altro fattore molto importante, per quanto riguarda la non linearità in SPECK è dato dal calcolo del riporto utilizzato per le somme. La complessità introdotta dal riporto non aumenta solo in funzione del round ma anche in funzione del bit preso in considerazione, crescendo man mano che ci si muove dal bit meno significativo a quello più significativo. La molteplicità di fonti di non linearità in SPECK rende molto difficile individuare una soluzione unica per risolvere i problemi emersi durante l'attacco esposto, di conseguenza è estremamente poco probabile che un attacco come quello riportato in [18] possa avere successo anche su SPECK.

Nelle considerazioni sulla progettazione di SIMON e SPECK che sono state presentate in [8] non è stato fatto alcun accenno alla crittoanalisi algebrica. L'unica spiegazione presentata per la scelta di differenti di operazioni coinvolte in SIMON e SPECK sta nell'ottimizzazione: la somma modulo n, l'unica ma più che sufficiente fonte di non linearità in SPECK, è più performante per un'implementazione software di quanto non lo sia la combinazione di AND e XOR utilizzati in SIMON, per il quale valgono le considerazioni opposte. Indubbiamente, le conseguenze di queste scelte non si limiteranno soltanto all'attacco presentato nel Capitolo 2, ma avranno per lo meno conseguenze per qualsiasi altra analisi che faccia ricorso a una rappresentazione algebrica delle due famiglie. Non è affatto da escludersi, alla luce di queste informazioni, che esistano anche altre tecniche di attacco le cui prestazioni varino molto da una famiglia all'altra.

Lo scopo dell'analisi di una primitiva crittografica è, innanzitutto, determinarne la sicurezza. Sia SIMON sia SPECK sono stati studiati ancora troppo poco per avere

certezze sulla loro sicurezza, tuttavia, alla luce di quanto già emerso, è possibile effettuare un primo bilancio di questi due anni di analisi:

- SIMON ha mostrato una certa resistenza iniziale all'analisi differenziale, tuttavia i progressi effettuati dagli studi successivi, riassunti nella Tabella 3, non lasciano escludere la possibilità che il numero di round attaccati cresca ulteriormente; a questo fattore bisogna aggiungere che, come mostrato in [18] e ribadito in questa tesi, combinare tecniche di analisi differenziale e algebrica può estendere significativamente il numero di round attaccati da quest'ultima.

- SPECK ha mostrato una resistenza all'analisi differenziale paragonabile a quella di SIMON, come viene illustrato nella Tabella 4, e una resistenza molto più elevata all'analisi algebrica; per i motivi esposti sopra, inoltre, è da considerarsi molto più difficile la riuscita di un attacco che combini le due tecniche. Sebbene sia prematuro assumere, per via dei risultati sopra esposti, che SPECK sia da considerarsi più sicuro di SIMON, è indubbio che, al momento, sono noti meno margini di vulnerabilità su cui costruire un futuro attacco.

### 3.2.1 Sviluppi futuri

Al momento della stesura di questa tesi SIMON e SPECK hanno fatto il loro ingresso nel mondo della crittografia da poco più di due anni ( [7] è stato pubblicato il 19 Giugno 2013): sebbene entrambi i cifrari siano già stati sottoposti a diverse analisi, si è ben lungi dall'avere un'idea precisa della loro effettiva forza. La quasi totalità degli attacchi mostrati finora si basa in qualche modo sull'analisi differenziale: [1, 2, 12, 19, 23] si basano interamente su questa tecnica, [18] utilizza anche l'analisi differenziale e, finora, solo [10] non ne ha fatto uso. Sebbene questa tecnica abbia già dimostrato la sua efficacia su numerosi cifrari a blocchi, un esempio fra tutti è [11], non è certo l'unico modo di mettere alla prova una primitiva crittografica. Ulteriore lavoro sull'analisi di entrambi i cifrari è sicuramente ancora necessario, quali che siano le tecniche utilizzate.

Dal punto di vista della crittoanalisi algebrica ci sono ancora numerosi passi da compiere verso una buona analisi sia di SIMON che di SPECK: innanzitutto, come già è stato detto più volte sopra, la ricerca di una diversa rappresentazione di SPECK potrebbe portare a risultati estremamente differenti per quanto riguarda la sua analisi. In secondo luogo, sono stati sottoposti a crittoanalisi differenziale tutti i cifrari di entrambe le famiglie, mentre soltanto due dei venti cifrari che complessivamente costituiscono SIMON e SPECK sono stati analizzati dal punto di vista algebrico. Né durante il lavoro di questa tesi né in [18] sono state fatte ipotesi sulla resistenza degli altri cifrari di queste famiglie. È sensato supporre che, riducendo la lunghezza dei blocchi, il problema di complessità che ha fermato gli attacchi descritti sopra si

verifichi più avanti nei round oppure che non si verifichi affatto, tuttavia questa supposizione richiede di essere verificata. Esiste, senza dubbio, ancora un certo margine di lavoro anche solo dedicandosi alla crittoanalisi algebrica di SIMON e SPECK, senza considerare altre tecniche.

# Bibliografia


[1] F. Abed, E. List, S. Lucks, J. Wenzel, "Differential and Linear Criptanalysis of Reduced-Round Simon", Cryptology ePrint Archive, Report 2013/526, 2013 http://eprint.iacr.org/2013/526

[2] F. Abed, E. List, S. Lucks, J. Wenzel, "Cryptanalysis of the SPECK Family of Block Ciphers", Cryptology ePrint Archive, Report 2013/568, 2013, http://eprint.iacr.org/2013/568

[3] M. Albrecht, "Algebraic Attacks on the Courtois Toy Cipher". Cryptologia, volume 32, numero 3, pp 220-276, 2008.

[4] M. Albrecht, G, Bard, The M4RI Library – Version 20090409, The M4RI Team, 2009

[5] M. Albrecht, "Algorithmic Algebraic Techniques and their Application to Block Cipher Cryptanalysis", PhD thesis, Royal Holloway, University of London, UK, 2010.

[6] H. A. Alkhzaimi e M. M. Lauridsen, "Cryptanalysis of the SIMON Family of Block Ciphers", Cryptology ePrint Archive, Report 2013/543, 2013 http://eprint.iacr.org/2013/543

[7] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, L. Wingers, "The Simon and Speck Families of Lightweight Block Ciphers", Cryptology ePrint Archive, Report 2013/404, 2013, http://eprint.iacr.org/2013/404

[8] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, L. Wingers, "The SIMON and SPECK Block Ciphers on AVR 8-bit Microcontrollers", Cryptology ePrint Archive, Report 2014/947, 2014, http://eprint.iacr.org/947

[9] D. J. Bernstein, "The Salsa20 Family of Stream Ciphers" in New Stream Cipher Designs—The eSTREAM Finalists, Lecture Notes in Compter Science, No. 4986, pp 84–97. Springer-Verlag, 2008.

[10] S. Bhasin, T. Graba, JL. Danger, Z. Najm, "A Look into SIMON from a Side-Channel Perspective" in 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp 56-59, 2014

[11] E. Biham, A. Shamir, "Differential cryptanalysis of des-like cryptosystem" in A.J. Menezes, S.A. Vanstone (edd.) Advances in Cryptology-CRYPT0'90, volume 537 of Lecture Notes in Computer Science, pp 2-21. Springer Berlin Heidelberg, 1991

[12] A. Biryukov, A. Roy, V. Velichkov, "Differential Analysis of Block Ciphers SIMON and SPECK", Cryptology ePrint Archive, Report 2014/922, 2014, `http://eprint.iacr.org/2014/922`

[13] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher" in CHES 2007, Lecture Notes in Computer Science, No. 4727, pp 450–66. Springer-Verlag, 2007

[14] M. Brickenstein, A. Dreyer, "PolyBoRi: A Gröbner basis framework for Boolean polynomials" in Journal of Symbolic Computation, volume 44, numero 9, pp 1326 - 1345, 2008

[15] D. Coppersmith, "The Data Encryption Standard (DES) and its strength against attacks", IBM Journal of Research and Development, 38(3),pp. 243-250, 1994

[16] N. Courtois, G. V. Bard, "Algebraic Cryptanalysis of the Data Encryption Standard", Cryptology ePrint Archive, Report 2006/402, 2006, `http://eprint.iacr.org/2006/402`

[17] N. Courtois, P. Sepehrdad, P. Susil, "S. Vaudenay, ElimLin Algorithm Revisited", in FSE 2012, Springer Verlag, 2012

[18] N. Courtois, T. Mourouzis, G. Song, P. Sepehrdad, P. Susil, "Combined Algebraic and Truncated Differential Cryptanalysis on Reduced-round Simon" in SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria, 28-30 August, pp 399-404, 2014

[19] I. Dinur, "Improved Differential Cryptanalysis of Round-Reduced Speck", Cryptology ePrint Archive, Report 2014/320, 2014, `http://eprint.iacr.org/2014/320`

[20] N. Eèn, N. Söresson, "An Extensible SAT-solver", in E. Giunchiglia, A. Tacchella (edd.), Theory and Applications of Satisfiability Testing, 6th International

Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers, pp 502-518, Spinger, 2004

[21] D. Le Berre, L. Simon (edd.), Journal on Satisfiability, Boolean Modeling and Computation, Volume 2, Special Volume on the SAT 2005 competitions and evaluations, 2006.

[22] H. Lipmaa, S. Moriai, "Efficient Algorithms for Computing Differential Properties of Addition", Cryptology ePrint Archive, Report 2001/001, 2001, `http://eprint.iacr.org/2001/001`

[23] T. Mourouzis, G. Song, N. Courtois, M. Christofii, "Advanced Differential Cryptanalysis of Reduced-Round SIMON64/128 Using Large-Round Statistical Distinguishers", Cryptology ePrint Archive, Report 2015/481, 2015 `http://eprint.iacr.org/2015/481`

[24] C. Sintz, A. Gupta, H. Jain, D. Le Berre, P. Manolios, Y. Novikov, Results of SAT-Race 2010, `http://baldur.iti.uka.de/sat-race-2010/results.html`

[25] F. Somenzi, CUDD: CU Decision Diagram Package Release 2.3.0. University of Colorado at Boulder, 1998.

[26] M. Soos, N. Karlsten, C. Castelluccia, "Extending SAT Solvers to Cryptographic Problems" in O. Kullman (ed.), Theory and Applications of Satisfiability Testing - SAT 2009, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009

[27] W. A. Stein et al.. Sage Mathematics Software (Version 6.7), The Sage Development Team, 2015,`http://www.sagemath.org`.