Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Progettazione e validazione di circuiti aritmetico-logici resistenti a crittanalisi di tipo side-channel

Relatore: Prof. Gerardo Pelosi
Correlatore: Prof. Alessandro Barenghi

Tesi di Laurea di:
Davide Macocchi Matr. 750154

Anno Accademico 2014–2015

Un ringraziamento a tutti coloro che mi hanno incoraggiato e sostenuto durante tutti questi anni.
Grazie di tutto.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

La *crittologia* é la scienza che studia le problematiche della trasmissione di informazioni sensibili su canali di comunicazione non sicuri, ovvero quei canali in cui queste informazioni potrebbero essere intercettate e lette da chi non ne é autorizzato. É un termine derivante dal greco kriptòs (nascosto) e logos (discorso) il quale indica la filosofia alla base di queste tecniche che consiste nel cifrare i messaggi in modo che risultino decifrabili solo dai destinatari dei messaggi stessi.

La crittologia si suddivide in due parti fondamentali che sono la *crittografia* e la *crittoanalisi*. Il compito della crittografia é quello di progettare i sistemi che permettono la comunicazione di messaggi su canali non sicuri mentre la crittoanalisi studia questi sistemi per poterli rompere, ovvero trovare delle possibili falle per decifrare i messaggi cifrati.

L'origine storica di queste tecniche non é molto chiara ma esistono tracce di algoritmi crittografici già nell'antica Roma, il cifrario di Cesare é l'esempio di algoritmo crittografico più antico di cui si abbia memoria. È un cifrario a scorrimento in cui ogni lettera del testo in chiaro viene sostituita dalla lettera più avanti nell'alfabeto di un numero prefissato di posizioni ed il suo scopo era quello di proteggere le comunicazioni militari trasmesse alle truppe.

Pur essendo un sistema crittografico molto semplice é risultato molto efficace in quanto chi intercettava i messaggi non era in grado di decifrarlo perchè non esistevano metodi di crittoanalisi in grado di romperlo. Con la scoperta della tecnica dell'analisi delle frequenze é stato possibile rompere facilmente questo crittosistema conoscendo il numero di occorrenze delle

lettere piu frequenti dei vocaboli di una determinata lingua.

Col passare dei secoli, il continuo perfezionamento delle tecniche di crittoanalisi ha portato alla continua evoluzione delle tecniche di crittografia che divennero sempre più sofisticate. Viceversa, l'aumento della complessita delle tecniche crittografiche ha spinto all'evoluzione delle tecnichce di crittoanalisi che sono diventate sempre più raffinate.

Nel 1880 venne dato un ulteriore sviluppo alle tecniche crittografiche grazie al principio di Kerckhoffs, il quale afferma che *"La sicurezza di un crittosistema non deve dipendere dal tener celato il crittoalgoritmo. La sicurezza deve dipendere solo dal tener celata la chiave"*. In questo modo un sistema crittografico ben progettato riesce a resistere agli attacchi crittografici anche se il nemico é a conoscenza dell'algoritmo utilizzato, l'unica informazione che deve essere protetta é quindi la chiave segreta.

Con l'introduzione di questo principio sono state sviluppate tecniche di crittoanalisi statistica che cercano di studiare le relazioni tra i testi in chiaro e i testi cifrati. L'avvento delle moderne tecnologie inoltre ha permesso raffinare queste tecniche crittoanalitiche e di superare facilmente i limiti imposti dalle tecniche crittografiche perchè é aumentata la potenza di calcolo a disposizione per rompere i crittosistemi.

L'ultimo passo compiuto dalle tecniche crittografiche é stato quello di cercare di ridurre al minimo la correlazione tra i testi in chiaro e i rispettivi testi cifrati in modo da vanificare la crittoanalisi statistica, inoltre si é cercato di allungare i tempi di calcolo necessari per decifrare un messaggio aumentando la complessità dell'algoritmo e la lunghezza della chiave segreta.

Le tecniche crittoanalitiche si sono evolute di nuovo per cercare nuovi tipi di attacchi per rompere questi crittosistemi. Sono state introdotte le tecniche denominate *Side Channel Attack* che sfruttano l'ormai indissolubile legame tra sistema crittografico e circuito elettronico. Un sistema crittografico che sfrutta un algoritmo crittografico viene infatti realizzato su un circuito elettronico che prende così il nome di circuito crittografico.

Gli attacchi SCA cercano di ricavare la chiave segreta attaccando le grandezze fisiche relative al circuito crittografico come ad esempio il consumo di potenza, i tempi di elaborazione o le radiazioni elettromagnetiche emesse. Tutti questi tipi di attacchi sono mirati ad attaccare l'implementazione dell'algoritmo crittografico piuttosto che l'algoritmo crittografico la cui

struttura é di dominio pubblico per il principio di Kerckhoffs.

I crittografi hanno allora elaborato delle possibili contromisure per prevenire o cercare di limitare questi attacchi.

In questo elaborato verranno sviluppati dei circuiti crittografici che implementano funzioni aritmetiche e sbox rappresentanti gli algoritmi di cifratura nelle versioni con e senza protezione dagli attacchi SCA di tipo *Correlation Power Analysis*, verrà studiato l'impatto ottenuto dall'introduzione della protezione agli attacchi CPA chiamata *Masking ISW* e verranno effettuati gli attacchi CPA veri e propri per valutarne la validità.

Nel Capitolo 2 verranno introdotte le basi teoriche degli attacchi crittografici chiamati *Side Channel Attack* con particolare riferimento alla tecnica chiamata *Correlation Power Analysis* e verrà esaminata la contromisura a questi attacchi denominata *Masking ISW*.

Nel Capitolo 3 verranno mostrate le architetture dei circuiti utilizzati per gli attacchi, in particolare verrano mostrati il circuito del sommatore binario, del moltiplicatore binario e della sbox dell'AES. I circuiti verranno sviluppati prima nella versione normale e poi nella versione protetta utilizzando le tecniche introdotte nel Capitolo 2.

Nel Capitolo 4 verranno spiegate le metodologie utilizzate per sviluppare i circuiti crittografici protetti col masking ISW e per eseguire gli attacchi, verranno inoltre riportate figure di merito riguardanti l' area dei circuiti risultanti e il loro cammino critico.

Infine nel Capitolo 5 verranno analizzati i risultati ottenuti dall'esecuzione degli attacchi CPA sui circuiti normali e protetti al variare del grado di protezione introdotto.

Capitolo 2

Attacchi Side Channel e Contromisure

In questo capitolo verranno mostrate le basi teoriche degli attacchi crittografici e delle relative contromisure con particolare riferimento alle tecniche utilizzate nei successivi capitoli di questo elaborato.

Nella prima sezione verranno introdotti i principi base degli attacchi crittografici chiamati *Side Channel Attack* tra i quali verrà analizzato il *Correlation Power Analysis*.

Nella seconda sezione verrà introdotta la contromisura a questi attacchi chiamata *Masking ISW* con i relativi dettagli per permettere l'implementazione di circuiti crittografici protetti.

2.1 Side Channel Attack

Un algoritmo crittografico é un procedimento con cui é possibile cifrare un messaggio in modo da renderlo comprensibile solo da chi é in possesso della chiave di decifratura. Questi algoritmi sono di solito implementati da dei circuiti elettronici che prendono il nome di circuiti crittografici.

Esistono principalmente due classi di algoritmi crittografici a seconda delle chiavi utilizzate nei processi di cifratura e decifratura. Gli algoritmi a chiave simmetrica utilizzano la stessa chiave segreta sia per cifrare sia per decifrare mentre gli algoritmi a chiave asimmetrica utilizzano una coppia di

chiavi formata dalla chiave pubblica in fase di cifratura e dalla chiave privata in fase di decifratura.

La sicurezza fornita da ciascuno di questi algoritmi si misura dalla difficoltà di eseguire attacchi crittografici, ovvero riuscire a ricavare informazioni sull'algoritmo utilizzato o sulla chiave segreta che permettano di decifrare facilmente il testo cifrato.

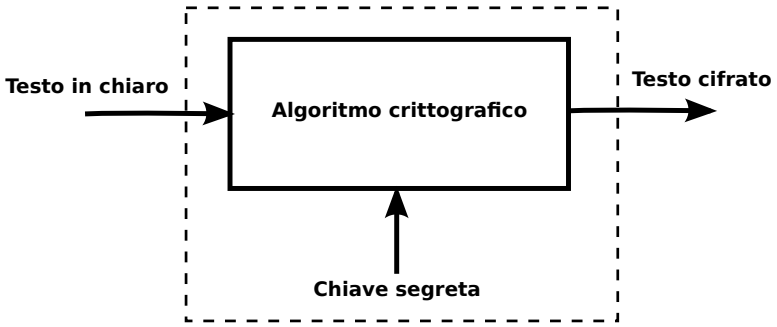


Figura 2.1: Circuito crittografico in fase di cifratura

La rappresentazione di un generico circuito crittografico in fase di cifratura é mostrata nella Figura 2.1.

Gli attacchi crittografici principalmente sviluppati in passato si concentravano sull'analisi dei testi in chiaro e cifrati per ricavare delle informazioni sull'algoritmo di cifratura o sulla chiave segreta. Esistono attacchi che si basano sulla conoscenza del solo testo cifrato *ciphertext-only attacks*, attacchi che sfruttano entrambi i testi in chiaro e cifrati *known plaintext attacks* e attacchi che cercano di ricavare informazioni sull'algoritmo di cifratura impostando opportunamente il testo in chiaro da usare come input, *chosen plaintext attacks*.

La difficoltà nell'eseguire questo tipo di attacchi si può incrementare rendendo l'algoritmo di cifratura più complesso in modo da rendere il testo cifrato meno dipendente dal relativo testo in chiaro e viceversa.

Questo tipo di sicurezza è strettamente legata all'algoritmo utilizzato, gli attacchi appena mostrati diventano meno efficaci una volta reso matematicamente robusto l'algoritmo di cifratura.

Indipendentemente dalla robustezza matematica dell'algoritmo, è possibile estrarre informazione riguardo alla chiave utilizzata misurando parametri ambientali (e.g., il consumo energetico) della computazione [7]. Questo

approccio è stato proposto per la prima volta in [6] da Kocher et al., con l' introduzione della tecnica nota come differential power analysis.

Un dispositivo elettronico produce durante l'elaborazione degli output fisici che corrispondono a delle grandezze misurabili come ad esempio il tempo impiegato ad eseguire la computazione, le radiazioni elettromagnetiche di diverso genere prodotte o il consumo di potenza del dispositivo stesso.

E' possibile, in aggiunta all'utilizzo di misure di grandezze ambientali che non alterano il funzionamento del dispositivo, andare a ricavare ulteriori informazioni inducendo errori controllati nella sua computazione. Questi errori possono essere ottenuti, ad esempio riducendo la tensione di alimentazione del circuito in maniera controllata, o alterandone il segnale di clock.

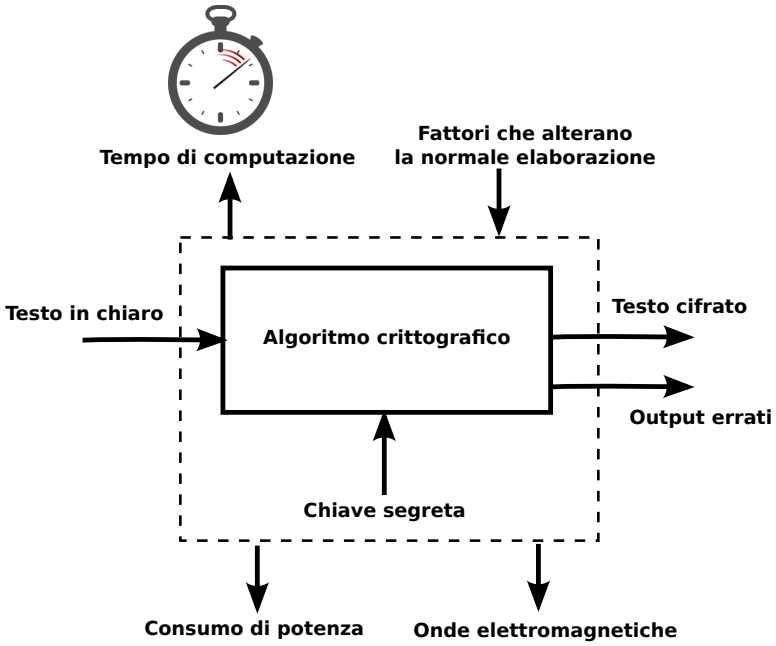


Figura 2.2: Circuito crittografico con side channel information

Il circuito crittografico della Figura 2.1 si può quindi arricchire con le nuove fonti di informazioni come mostrato nella Figura 2.2.

Ciascuna di queste informazioni aggiuntive si chiama *Side Channel Information* che é definita come l'informazione che si può ricavare dal circuito crittografico durante l'elaborazione che non sia né il testo in chiaro né quello cifrato. Sfruttando l'informazione ottenuta da uno dei precedenti side chan-

nel information si può quindi sviluppare un attacco crittografico, il quale prenderà il nome di *Side Channel Attack*.

Il tipo di side channel attack studiato in questo elaborato é quello legato all' analisi dell' energia necessaria per la computazione, comunemente noto come analisi in potenza.

2.1.1 Power Analysis Attacks

Gli attacchi denominati *Power Analysis Attacks* sfruttano il side shannel legato alla potenza elettrica dissipata da un circuito crittografico durante l'elaborazione per ottenere informazioni sulla chiave segreta o sull'algoritmo crittografico utilizzato.

L'idea alla base di questo attacco é quella di raccogliere un insieme di misure di consumo, dette *tracce*, e di tentare di indovinare una piccola porzione della chiave segreta per ciascuna di esse. Più alta sarà la relazione tra la traccia in esame e l' ipotesi di chiave segreta fatta e più alta sarà la probabilità che la chiave segreta sia quella realmente utilizzata durante l'esecuzione dell'algoritmo.

La forza di questo approccio sta nel poter ipotizzare separatamente piccole porzioni di chiave, riducendo significativamente la complessità computazionale dell'attacco, rispetto ad un tentativo di indovinare l' intera chiave. L' approccio è reso possibile dal fatto che la chiave segreta viene combinata con il testo in chiaro da parti del circuito crittografico indipendenti tra loro (e.g., un operazione di `xor` bit a bit), consentendo dunque di indovinarli separatamente [11].

Simple Power Analysis

L'attacco di tipo *Simple Power Analysis* (SPA) cerca di sfruttare l'eventuale dipendenza tra le operazioni eseguite dal circuito crittografico durante l'elaborazione e la chiave segreta.

L'idea di fondo é che l'analisi dei tracciati dei consumi di potenza potrebbe evidenziare profili di consumo diversi a seconda del tipo di operazione eseguita.

L'attacco consiste nell'individuare gli istanti di tempo in cui é nota l'esecuzione di operazioni che agiscono sulla chiave e osservare se esiste una relazione tra i profili dei tracciati e i possibili valori della chiave.

L'esempio piu classico di questa situazione é quello della traccia di potenza legata al calcolo di x^y mod n con la tecnica dello *Square and Multiply* presente nell' algoritmo di firma del crittosistema RSA.

In questo algoritmo la chiave segreta é l'esponente y utilizzato per determinare l'operazione da eseguire come mostrato qui di seguito.

```
1    def executeSquareAndMultiply(x,y,n):
2      #conversione di y in binario
3      binary_y = []
4      while y != 0:
5          binary_y.append(y%2)
6          y = y/2
7      binary_y.reverse()
8      #esecuzione square and multiply
9      result = 1
10     for i in binary_y:
11         if i = = 0:
12             #Square
13             result = (result*result) % n
14         else:
15           #Square and Multiply
16           result = (result*result*x) % n
17     return result
```

Si può notare come la condizione dell'if (riga 11) dipenda esclusivamente dal valore dei bit della chiave y infatti nel caso in cui il bit di y in esame sia 0 verrà eseguita solo l'operazione di Square (riga 13), altrimenti verrà eseguita l'operazione di Square e Multiply (riga 16).

La Figura 2.3 mostra il consumo di un circuito che effettua i calcoli dello Square and Multiply per diversi valori di byte di chiave y. É possibile notare come le due operazioni di Square e di Multiply abbiano dei profili di consumo di potenza ben riconoscibili che permettono immediatamente di ricavare il valore dell'esponente y utilizzato e quindi la chiave segreta.

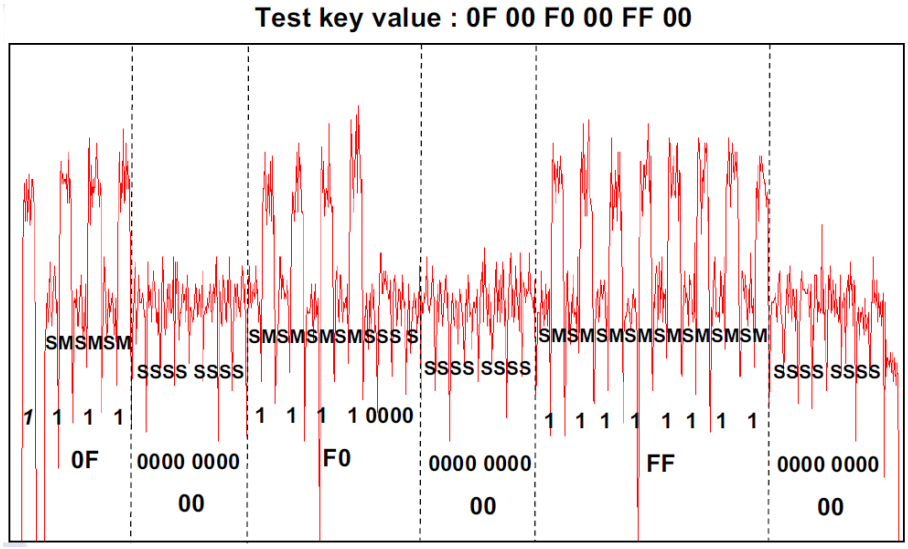


Figura 2.3: SPA applicata allo Square and Multiply

Differential Power Analysis

Gli attacchi di tipo *Differential Power Analysis* (DPA) cercano di sfruttare l'eventuale dipendenza tra i valori assunti dal consumo di potenza e la chiave segreta.

L'idea alla base dell'attacco DPA é quella di calcolare il possibile andamento del consumo di potenza per tutte le combinazioni della chiave e successivamente confrontare ciascuna di queste previsioni con il consumo effettivamente rilevato assegnandoli un grado di somiglianza. La chiave segreta restituita dall'attacco sarà quella la cui previsione di consumo di potenza si avvicina di più al consumo misurato.

Un attacco DPA é composto da cinque fasi ben distinte:

1. Nella prima fase viene identificata la funzione $f(d, k)$ che calcola un valore intermedio dell'algoritmo crittografico dipendente da un input noto d e da k che rappresenta una porzione della chiave segreta. Viene utilizzata una porzione di chiave e non la chiave intera per non rendere troppo grosso lo spazio di ipotesi della chiave, da qui in poi denominato $\mathcal{K}$. In genere si utilizzano $6-8$ bit e una volta ricavata quella porzione di chiave segreta si riesegue l'attacco passando alla porzione successiva.

2. Nella seconda fase si raccolgono N tracce di consumo di potenza $\mathbf{t_i}$ utilizzando N differenti input noti d_i e una chiave fissa $\hat{k}$. Queste misure costituiranno la matrice $\mathbf{T}$ di dimensioni $N \times M$, dove ogni traccia di potenza rappresenta un vettore riga, mentre i valori di input noti costituiranno il vettore colonna $\mathbf{d}$

$$\mathbf{T} = \begin{pmatrix} \mathbf{t_1} \\ \mathbf{t_2} \\ \vdots \\ \mathbf{t_i} \\ \vdots \\ \mathbf{t_N} \end{pmatrix} = \begin{pmatrix} t_{1,1} & t_{1,2} & \dots & t_{1,j} & \dots & t_{1,M} \\ t_{2,1} & t_{2,2} & \dots & t_{2,j} & \dots & t_{2,M} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ t_{i,1} & t_{i,2} & \dots & t_{i,j} & \dots & t_{i,M} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ t_{N,1} & t_{N,2} & \dots & t_{N,j} & \dots & t_{N,M} \end{pmatrix} \qquad \mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_i \\ \vdots \\ d_N \end{pmatrix}$$

3. Nella terza fase vengono calcolati i possibili valori intermedi $v_{i,l}$ per ogni input noto d_i e ogni possibile valore di $k_j \in \{k_1, k_2, \dots, k_l, \dots k_{\mathcal{K}}\}$ tale che $v_{i,l} = f(d_i, k_l)$. Questo calcolo produce una matrice $\mathbf{V}$ di dimensioni $N \times |\mathcal{K}|$ che contiene ogni valore intermedio per ogni possibile valore di chiave k_l, come mostrato qui di seguito:

$$\mathbf{V} = \begin{pmatrix} v_{1,1} = f(d_1, k_1) & \dots & v_{1,l} = f(d_1, k_l) & \dots & v_{1,|\mathcal{K}|} = f(d_1, k_{|\mathcal{K}|}) \\ v_{2,1} = f(d_2, k_1) & \dots & v_{2,l} = f(d_2, k_l) & \dots & v_{2,|\mathcal{K}|} = f(d_2, k_{|\mathcal{K}|}) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ v_{i,1} = f(d_i, k_1) & \dots & v_{i,l} = f(d_i, k_l) & \dots & v_{i,|\mathcal{K}|} = f(d_i, k_{|\mathcal{K}|}) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ v_{N,1} = f(d_N, k_1) & \dots & v_{N,l} = f(d_N, k_l) & \dots & v_{N,|\mathcal{K}|} = f(d_N, k_{|\mathcal{K}|}) \end{pmatrix}$$

Ogni colonna l contiene il valore intermedio calcolato assumendo uno specifico valore di chiave k_l tra tutti quelli possibili e il valore di input noto d_i. Lo scopo dell'attacco DPA é quello di identificare quale colonna contiene il valore intermedio calcolato dal circuito crittografico che permetterà quindi di ricavare la porzione di chiave k_l corretta tra tutte quelle possibili.

4. Nella quarta fase viene utilizzato un modello di consumo di potenza che mappa la matrice $\mathbf{V}$ dei possibili valori intermedi con la matrice $\mathbf{P}$ dei possibili valori di consumo di potenza, il modello dei consumi f_s

viene scelto a seconda dell'implementazione dell'algoritmo di cifratura:

$$\mathbf{P} = \begin{pmatrix} p_{1,1} = f_s(v_{1,1}) & \dots & p_{1,l} = f_s(v_{1,l}) & \dots & p_{1,|\mathcal{K}|} = f_s(v_{1,|\mathcal{K}|}) \\ p_{2,1} = f_s(v_{2,1}) & \dots & p_{2,l} = f_s(v_{2,l}) & \dots & p_{2,|\mathcal{K}|} = f_s(v_{2,|\mathcal{K}|}) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{i,1} = f_s(v_{i,1}) & \dots & p_{i,l} = f_s(v_{i,l}) & \dots & p_{i,|\mathcal{K}|} = f_s(v_{i,|\mathcal{K}|}) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{N,1} = f_s(v_{N,1}) & \dots & p_{N,l} = f_s(v_{N,l}) & \dots & p_{N,|\mathcal{K}|} = f_s(v_{N,|\mathcal{K}|}) \end{pmatrix}$$

5. Nella quinta e ultima fase ogni colonna della matrice **P** dei consumi stimati viene confrontata con ciascuna delle colonne della matrice **T** in modo da calcolare una misura di correlazione tra i consumi stimati, per ogni ipotesi di chiave, e i consumi effettivi. Lo scopo di questo confronto é determinare quale delle ipotesi di chiave dà luogo alle ipotesi di consumo meglio correlate.

   Per ciascuno dei valori di chiave verrà associato alla colonna dei consumi stimati un valore di confidenza statistica per indicare quanto il confronto con il valore misurato risulti veritiero. Tra tutti questi valori, quello con il valore di confidenza più alto sarà quello che conterrà la porzione di chiave cercata.

Il nucleo fondamentale di tutti gli attacchi DPA é il test statistico utilizzato per individuare la dipendenza tra le predizioni dei consumi e le misure dei consumi vere e proprie e in base a questo tipo di test possono assumere nomi diversi come avviene per il DPA mostrato nella prossima sezione.

Correlation Power Analysis

L'indice di correlazione di Pearson $\rho_{X,Y}$ tra due variabili casuali X e Y é definito come il rapporto tra la covarianza delle due variabili $\text{cov}(X, Y)$ e il prodotto delle loro deviazioni standard, rispettivamente σ_X e σ_Y

$$\rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

Questo indice serve a stabilire il grado di correlazione lineare tra due variabili casuali in una scala tra $[-1, 1]$. Valori prossimi a -1 o 1 indicano

un alto grado di correlazione lineare rispettivamente diretta e inversa, mentre valori vicini allo 0 indicano un basso livello di correlazione lineare o la mancanza di correlazione.

L'indice di correlazione di Pearson può essere impiegato come test statistico per misurare la correttezza delle previsioni dei consumi di potenza in un attacco DPA, questa tecnica è nota in letteratura come *Correlation Power Analysis* (CPA) [1].

É necessario considerare i valori dei consumi misurati ad un certo istante di tempo $\mathbf{t}_j = [t_{1,j}, t_{2,j}, \ldots, t_{i,j}, \ldots, t_{N,j}]$ e le predizioni dipendenti da un valore di chiave $\mathbf{p}_l = [p_{1,l}, p_{2,l}, \ldots, p_{i,l}, \ldots, p_{N,l}]$ come campioni estratti da due variabili aleatorie. Avendo a disposizione i campioni di queste due variabili e non la loro distribuzione teorica, si calcola l'indice di correlazione di Pearson dai campioni $r_{\mathbf{t}_j,\mathbf{p}_l}$ come uno stimatore unbiased del valore dell'indice di correlazione di Pearson $\rho_{\mathbf{t}_j,\mathbf{p}_l}$.

L'indice di correlazione di Pearson dai campioni $r_{\mathbf{t}_j,\mathbf{p}_l}$ contenuti in $\mathbf{p}_l$ e $\mathbf{t}_j$ si calcola con la formula

$$r_{\mathbf{t}_j,\mathbf{p}_l} = \frac{\sum_i (t_{i,j} - \overline{t}_j)(p_{i,l} - \overline{p}_l)}{\sqrt{\sum_i (t_{i,j} - \overline{t}_j)^2 \sum_i (p_{i,l} - \overline{p}_l)^2}} \tag{2.1}$$

dove $\overline{t}_j$ e $\overline{p}_l$ sono le medie campionarie rispettivamente su $\mathbf{t}_j$ e $\mathbf{p}_l$.

In definitiva, durante l'attacco CPA, l'attaccante calcola il valore di $r_{\mathbf{t}_j,\mathbf{p}_l}$ lungo tutti gli istanti di tempo delle tracce misurate per ogni ipotesi di chiave k_l e scegli come chiave corretta quella che possiede il piu alto valore di correlazione tra tutte quelle calcolate.

2.2 Contromisure

Nella precedente sezione sono state introdotte le basi degli attacchi che sfruttano le informazioni rivelate dai side channel. In questa sezione verranno esposte le contromisure adottabili per rendere più difficile l'esecuzione di questi attacchi.

L'idea di base delle contromisure ai side shanne attacks (SCA) é quella di rendere meno dipendente l'elaborazione dalle informazioni rilasciate ai side

channel.

Esistono due tipologie di contromisure ovvero l'*hiding* e il *masking*. L'hiding cerca di rimuovere la dipendenza nascondendo l'informazione che arriva al side channel interessato mentre il masking cerca invece di mascherarla aggiungendo informazioni randomizzate che si mescolano a quelle utili per gli attacchi.

L'hiding applicato ai power analysis attacks si realizza rendendo il consumo di potenza costante o completamente randomizzato aggiungendo quindi operazioni che alterano radicalmente le informazioni fornite al Side Channel.

Nel masking si effettua una strategia diversa ovvero si alterano i risultati intermedi delle operazioni utilizzando dei valori randomizzati. In questo modo l'elaborazione avviene senza problemi ma rende più difficoltosi gli attacchi DPA che sfruttano appunto questi risultati intermedi per effettuare delle previsioni sui consumi.

Qui di seguito verrà illustrata la tecnica utilizzata per impedire gli attacchi power analysis attacks che verrà utilizzata nella realizzazione dei circuiti.

2.2.1 Masking ISW

La tecnica del *Masking ISW* prende il nome dai suoi tre inventori Ishai, Sahai e Wagner ed effettua il masking a basso livello introducendo dei valori casuali estratti a runtime nelle porte logiche del circuito crittografico [5] [4].

Questa tecnica si compone di tre fasi:

1. Nella prima fase viene effettuato lo *share splitting*, ovvero ogni singolo bit in input viene suddiviso in un numero $d+1$ prefissato di bit o *share* utilizzando d valori casuali

2. Nella seconda fase viene eseguita l'elaborazione prevista dall'algoritmo crittografico sostituendo ad ogni porta logica la sua versione protetta, la quale utilizzerà le share al posto dei singoli bit. Le operazioni eseguite dalle porte logiche protette sono progettate in maniera tale da conservare la proprietà tale per cui, ricombinando le $d+1$ share é possibile ottenere il risultato corretto.

3. L'ultima fase avviene al termine dell'elaborazione quando l'output é disponibile in uscita nella sua versione composta da share. Questa fase

prevede di ricostruire la versione non mascherata dell'output a partire dalla versione in $d + 1$ share

Nelle prossime sezioni verranno approfondite nel dettaglio queste tre fasi del masking ISW.

2.2.2 Share Splitting

La fase di share splitting é la fase preliminare del masking ISW. Il suo scopo é quello di suddividere ciascuno dei bit che compongono l'input in $d+1$ frammenti o share.

Per eseguire questa operazione vengono estratti d valori booleani casuali $r_0, \ldots, r_d$. Il singolo bit i viene suddiviso in $d+1$ share denominate $s_0, \ldots, s_d$ attraverso le seguenti regole

$$\begin{cases} s_0 = (((i \oplus r_0) \oplus r_1) \oplus \ldots) \oplus r_{d-1} \\ s_i = r_{i-1} \quad i \in \{1, \ldots, d\} \end{cases} \tag{2.2}$$

Dall'equazione 2.2 si può notare che la share s_0 é composta dall'input i a cui viene applicata la funzione di `xor` bit a bit con tutti i d valori casuali mentre le rimanenti share corrispondono proprio a questi valori casuali $r_0, \ldots, r_d$.

2.2.3 Porte logiche protette

In questa sezione verranno presentate tutte le porte logiche protette presenti nella seconda fase del masking ISW, le quali eseguono le stesse operazioni binarie delle porte logiche normali con la differenza di operare sulle share al posto dei singoli bit.

Porta `not` Protetta

La porta logica `not` protetta si ottiene prendendo le share $s_0, \ldots, s_d$ e negando la share s_0 ottenendo quindi in output $(\neg s_0)\, s_1, \ldots, s_d$.

Questa operazione é corretta perchè, come verrà mostrato più avanti, la ricostruzione dell'output avviene eseguendo lo `xor` bit a bit di tutte le share e quindi negandone una si ottiene proprio la negazione di tutto il risultato ovvero l'operazione di `not`.

Porta `xor` Protetta

La porta logica `xor` protetta ha il compito di calcolare lo `xor` bit a bit di due input u e v convertiti in $d+1$ shares, $u_0, \ldots, u_d$ e $v_0, \ldots, v_d$. Questa operazione va realizzata senza che nessun valore intermedio sia direttamente dipendente da uno dei due input: a questo scopo sono necessari d valori casuali $r_0, \ldots, r_{d-1}$.

Per prima cosa é necessario ricavare le share del valore intermedio $\tilde{u}$ applicando i bit random r alle share di u nel seguente modo

$$\begin{cases} \tilde{u}_0 = \oplus_{j=0}^{d-1} r_j \\ \tilde{u}_{i+1} = u_{i+1} \oplus r_i \quad i \in \{0, \ldots, d-1\} \end{cases} \tag{2.3}$$

Similmente si calcolano le share del valore intermedio $\tilde{v}$ con le seguenti formule

$$\begin{cases} \tilde{v}_0 = \oplus_{j=0}^{d-1} r_j \\ \tilde{v}_{i+1} = v_{i+1} \oplus r_i \quad i \in \{0, \ldots, d-1\} \end{cases} \tag{2.4}$$

Infine, i due risultati temporanei precedenti vengono ricombinati nel seguente modo:

$$o_i = \tilde{u}_i \oplus \tilde{v}_i \quad i \in \{0, \ldots, d\} \tag{2.5}$$

La formula dell'equazione 2.5 esegue l'operazione di `xor` in quanto i contributi casuali introdotti nei valori intermedi $\tilde{u}$ e $\tilde{v}$ si annullano come verrà mostrato in fase di ricostruzione dell'output.

Porta `and` protetta

La porta logica `and` protetta ha il compito di eseguire la `and` bit a bit di due input u e v convertiti in share.

Per eseguire questa operazione é necessario utilizzare $\frac{d(d+1)}{2}$ valori casuali $z_{i,j}$ per $i, j \in \{0, \ldots, d\}$ con $i < j$. Successivamente vengono calcolati i valori $z_{j,i}$ con $i < j$ con la seguente formula

$$z_{j,i} = (u_i \wedge v_j) \oplus z_{j,i} \oplus (u_j \wedge v_i)$$

In questa formula é fondamentale rispettare l' ordine con cui vengono eseguite le operazioni che la compongono, altrimenti verrebbe esposto un valore dipendente dal valore degli ingressi u e v rendendo la porta vulnerabile ad attacchi di tipo SCA.

Il risultato finale, ovvero l'`and` bit a bit dell'input u e v convertito in share, é dato dalla seguente formula

$$o_i = (u_i \wedge v_i) \underset{i \neq j}{\oplus} z_{i,j} \tag{2.6}$$

Porta `or` protetta

La porta logica `or` protetta é ottenuta attraverso una combinazione di `and` e `not`, applicando la legge di De Morgan come segue:

$$A \vee B = \overline{(\overline{A} \wedge \overline{B})} \tag{2.7}$$

2.2.4 Ricostruzione output

L'ultima fase del masking ISW é rappresentata dall'operazione di ricostruzione dell'output in chiaro o partendo dalle share $s_0, \ldots, s_d$ che lo compongono.

La ricostruzione si effettua con la seguente formula

$$o = \oplus_{j=0}^{d} s_j \tag{2.8}$$

La correttezza della formula 2.8 é una conseguenza della metodologia con cui sono state ricavatele le $d + 1$ shares secondo la formula 2.2.

La prima share s_0 corrisponde alla `xor` dell'input in chiaro i con tutti i valori casuali rappresentati dalle altre share. Sommando tutte le share $s_0, \ldots, s_d$ si esegue quindi due volte lo `xor` del contributo di ciascuno dei d input casuali che si annullano a vicenda lasciando in output il solo valore del risultato in chiaro.

Conclusione

In questo capitolo sono sono state mostrate le basi teoriche degli attacchi SCA con riferimento all'attacco CPA il cui scopo é quello di ricavare la chiave segreta utilizzata da un circuito crittografico analizzando il consumo di potenza del circuito stesso durante l'elaborazione.

Successivamente é stata introdotta una possibile contromisura, il masking ISW, che permette di mascherare i valori intermedi sfruttati dall'attacco CPA per ricavare informazioni sulla chiave segreta.

Queste basi teoriche sono necessarie per implementare i circuiti le cui architetture sono descritte nel Capitolo 3 ed effettuare gli attacchi effettuati alle loro versioni normali e protette presenti nel Capitolo 5.

Capitolo 3

Unità Funzionali per Circuiti Crittografici

In questo capitolo verranno introdotte le architetture e i principi di funzionamento dei circuiti su cui verranno effettuati gli attacchi CPA già introdotti nel Capitolo 2. Il primo circuito mostrato é il sommatore binario, si proseguirà quindi con il moltiplicatore binario per poi concludere con la sbox dell'algoritmo di cifratura AES. Per ogni circuito verrà illustrato il procedimento per ricavare il circuito della versione base e verranno poi descritti i passaggi per realizzare la versione protetta utilizzando il masking ISW introdotto nel Capitolo 2.

3.1 Sommatore

Il sommatore binario é il circuito il cui compito é quello di eseguire l'addizione di due numeri in codifica binaria delle stesse dimensioni massime A e B. Esistono diverse tipologie architetture di sommatori binari, la versione presentata é la versione *ripple carry adder*.

In questa architettura il risultato finale della somma S e il riporto C viene calcolato sommando ciascuno dei bit nelle posizioni A_i e B_i con il riporto C_{i-1} propagato dalla somma dei bit A_{i-1} e B_{i-1}. Il riporto si propaga in sequenza dal bit meno significativo al bit più significativo.

3.1.1 Implementazione

Prima di presentare il circuito del sommatore vero e proprio é necessario introdurre i circuiti base che lo compongono, cioé l'*half adder* e il *full adder*. Combinando opportunamente questi singoli circuiti é possibile ottenere un sommatore di due parole a N bit con N scelto in modo arbitrario.

Half Adder

L'half adder é il sommatore a bit singolo più semplice. Il suo compito é sommare due singoli bit A e B e fornire in output due bit che rappresentano la loro somma S e l'eventuale riporto C.

Tabella 3.1: Tabella di verità half adder

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Come si può notare dalla Tabella 3.1 la somma S corrisponde all'operazione di `xor` dei bit di input mentre il riporto C corrisponde alla `and`.

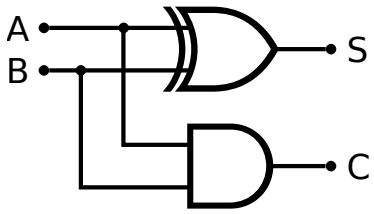


Figura 3.1: Rete combinatoria half adder

La rete combinatoria ottenuta é visibile in Figura 3.1.

L'half adder può essere visto come un sommatore di due bit senza riporto in ingresso e può essere utilizzato per realizzare il full adder mostrato nella seguente sezione.

Full Adder

Il full adder é un circuito digitale il cui scopo é quello di sommare assieme due addendo di un bit, tenendo in considerazione la possibile presenza di un

riporto in entrata. A differenza dell'half adder oltre ai due bit di ingresso A e B é presente un terzo bit C_{in} che rappresenta il riporto in ingresso.

Il full adder é quindi un sommatore a due bit con riporto in ingresso i cui output sono presenti il bit di somma S e il bit di riporto C_{out}.

Tabella 3.2: Tabella di verità full adder

C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Il comportamento del full adder rispecchia quanto esposto nella tabella di verità 3.2.

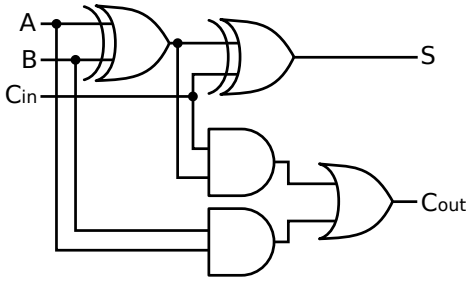


Figura 3.2: Rete combinatoria full adder con porte logiche elementari

La rete combinatoria composta da porte logiche elementari é mostrata nella Figura 3.2.

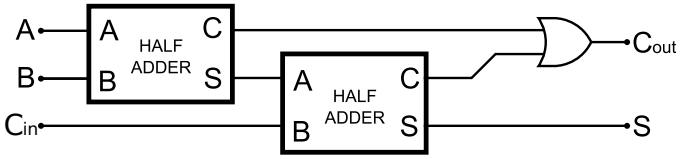


Figura 3.3: Rete combinatoria full adder con half adder

Il circuito del full adder può essere riscritto utilizzando il circuito dell'half adder precedentemente introdotto come mostrato nella Figura 3.3.

Sommatore

Il sommatore di due parole a N bit é realizzato collegando in cascata N full adder, in questo modo verranno sommati assieme tutti gli N bit delle due parole in ingresso, mentre i riporti verranno propagati dal bit meno significativo fino al bit più significativo.

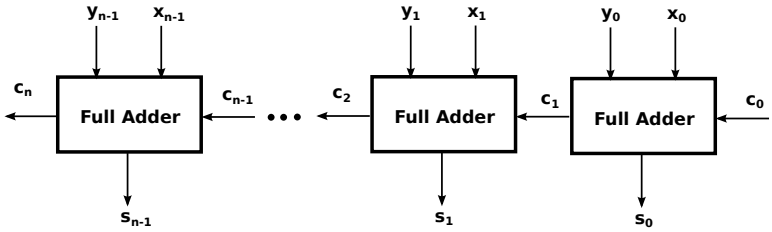


Figura 3.4: Rete combinatoria sommatore

Il circuito del sommatore é presentato in Figura 3.4. Per permettere il corretto funzionamento del circuito é necessario impostare il primo riporto C_0 a 0 oppure in alternativa é possibile sostituire il full adder che somma i bit meno significativi con un half adder. I vantaggi attribuibili a questa architettura provengono dalla semplicità e dall'uniformità dei blocchi che la costituiscono, in particolare la modifica del numero di bit delle parole in ingresso avviene aggiungendo o rimuovendo dei blocchi dalla sequenza di full adder. Viceversa gli svantaggi sono legati alle prestazioni, infatti prima di avere in uscita gli output corretti bisogna attendere che tutti i riporti siano propagati dal bit meno significativo al bit più significativo.

3.1.2 Sommatore protetto da attacchi side-channel

La progettazione di un sommatore che effettui calcoli interamente in forma share-split avviene effettuando delle modifiche alle componenti di base lasciando inalterata la struttura dei collegamenti tra i vari blocchi come indicato nel Capitolo 2.

Le porte logiche interessate da questa modifica sono la `xor` e la `and` dell'half adder e la `or` del full adder che verranno sostituite con le rispettive

versioni protette. Per rendere completa la modifica bisogna prevedere l'inserimento di un ulteriore bus di bit in input a tutto il sommatore che permette di propagare i segnali random necessari per l'elaborazione con le porte logiche protette.

L'ultima operazione da eseguire riguarda i collegamenti tra i vari blocchi che rimangono esattamente gli stessi a patto che ogni singolo bit venga sostituito con un bus di bit il cui numero corrisponde al numero di share utilizzate per il mascheramento.

3.2 Moltiplicatore

Il moltiplicatore binario é il circuito utilizzato per effettuare il prodotto di due numeri. Sebbene la moltiplicazione di due numeri possa essere rimpiazzata da una serie di somme, la necessità di avere il risultato di questa operazione in tempi brevi ha spinto lo sviluppo di questo tipo di circuito.

3.2.1 Implementazione

L'implementazione adottata ricalca il metodo di calcolo base della moltiplicazione, ovvero ogni cifra del primo fattore viene moltiplicata per ogni cifra del secondo creando delle somme parziali dopodiché queste somme parziali vengono sommate assieme e forniscono in uscita il prodotto degli input come mostrato nella Figura 3.5.

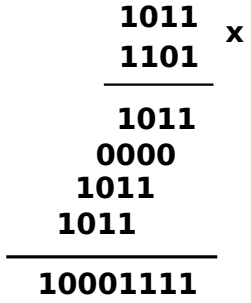


Figura 3.5: Esempio di moltiplicazione binaria

Il calcolo vero e proprio del prodotto delle parole binarie in ingresso é stato diviso in due step. Nel primo step vengono effettuate tutte le moltiplicazioni dei bit delle due parole per renderle disponibili al circuito del secondo step dove vengono effettivamente sommate.

Nel circuito del primo step le due parole binarie entrano in un circuito in cui ogni bit della prima parola viene moltiplicato per ogni bit della seconda parola. L'operazione di moltiplicazione a livello di bit si traduce nella `and` dei bit stessi. Il circuito quindi prende in ingresso due parole di M e N bit e fornisce in output una matrice di $M \times N$ bit.

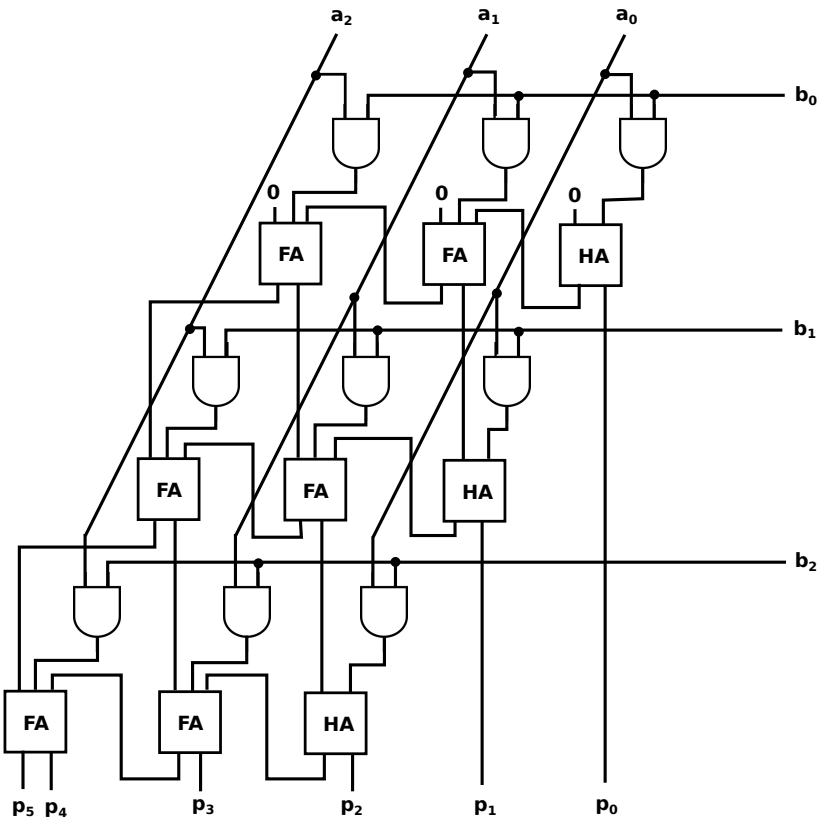


Figura 3.6: Rete combinatoria di un moltiplicatore con ingressi a 3 bit

La struttura della versione a tre bit del moltiplicatore é mostrata in Figura 3.6. Tutti i blocchi presenti nella rete combinatoria sono l'half adder e il full adder già introdotti nella costruzione del sommatore della sezione 3.1.

Il vantaggio dell'utilizzo di questo circuito é la regolarità dei blocchi utilizzati. Anche in questo tipo di circuito la modifica del numero di bit delle parole in ingresso si traduce nell'aggiunta o nella rimozione di righe o colonne di blocchi.

I problemi derivati dall'uso di questo moltiplicatore sono legati a problemi

di area e di prestazioni. Aumentare il numero di bit in ingresso si traduce nell'aggiungere righe di porte `and` nel circuito del primo step e nell'aggiungere righe di sommatori nel circuito del secondo step che si ripercuote sull'area occupata del circuito finale.

Le prestazioni sono influenzate dal numero di bit delle parole in ingresso perché il prodotto in uscita sarà disponibile solo quando tutte le somme parziali saranno effettivamente calcolate e riportate alle righe successive, quindi più aumentano i bit in ingresso maggiore sarà il cammino critico del circuito.

3.2.2 Moltiplicatore protetto da attacchi side-channel

La progettazione del moltiplicatore protetto dagli attacchi CPA avviene in maniera simile a quanto accaduto per il sommatore.

Le porte logiche interessate sono le porte logiche della matrice di `and` e tutte le porte logiche dei sommatori che verranno sostituite con le rispettive versioni protette. Bisogna prevedere anche qui l'inserimento di un ulteriore bus di bit in input a tutto il moltiplicatore dedicato ai segnali random per permettere l'elaborazione delle porte logiche protette.

I collegamenti tra i vari blocchi rimangono anche qui gli stessi sotto la condizione che ogni singolo bit venga sostituito con un bus di bit il cui numero corrisponde al numero di share utilizzate per il mascheramento.

3.3 Sbox

L'*Advanced Encryption Standard* (AES) é la specifica di un algoritmo di cifratura a blocchi utilizzato come standard dalla National Institute of Standards and Technology (NIST) degli Stati Uniti d'America [8]. É un algoritmo a chiave simmetrica che opera su input di 128 bit utilizzando un chiave segreta che può essere scelta a 128, 192 o 256 bit.

L'algoritmo alla base dell'AES si chiama Rijndael [2] e il suo funzionamento prevede di organizzare i 128 bit di input in una matrice di 4x4 bytes, chiamata *matrice di stato*, la quale verrà modificata da quattro primitive per un numero prefissato di cicli detti *round*.

Il numero di esecuzioni dei round dipende dalla lunghezza della chiave segreta in ingresso all'algoritmo, essi verranno infatti eseguiti 9, 11 o 13 volte rispettivamente per chiavi da 128, 192 e 256 bit.

All'inizio dell'esecuzione dell'algoritmo viene eseguita una funzione chiamata *Key Expansion* che non modificherà la matrice di stato ma ha il compito di generare le chiavi da 128 bit utilizzate da ogni round, dette *chiavi di round*.

Lo schema di funzionamento dell'AES é mostrato in Figura 3.7.

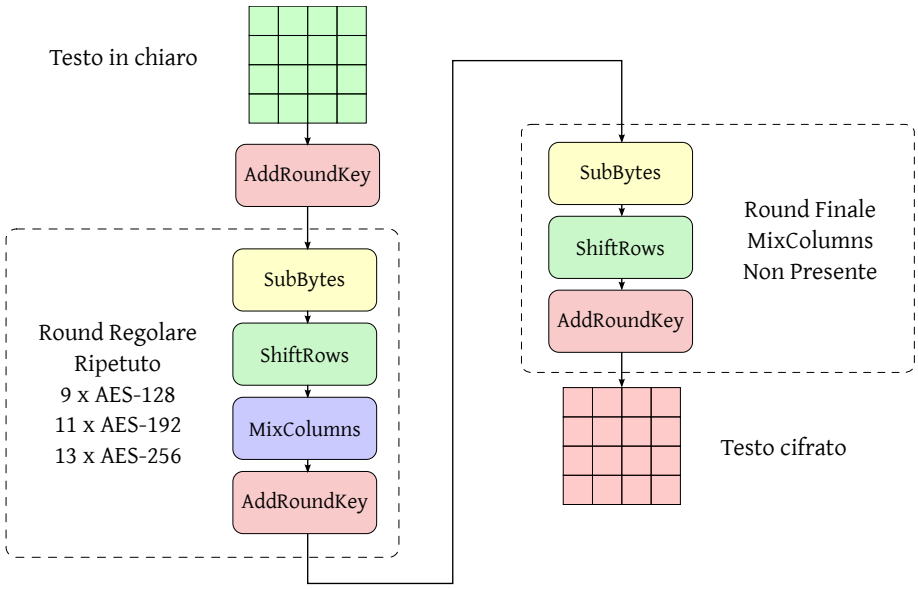


Figura 3.7: Schema di funzionamento dell'AES

Le primitive applicate applicate alla matrice di stato da ogni round sono le seguenti:

1. Lo step *SubBytes* applica a ciascuno dei 16 byte della matrice la trasformazione non lineare realizzata dalla sbox

2. Lo step *ShiftRows* ruota a sinistra i byte di ogni riga della matrice di un numero prefissato di posizioni. La prima riga rimane invariata, la seconda ruota di una posizione, la terza di due posizioni e la quarta di tre posizioni

3. Lo step *MixColumns* opera su ogni colonna della matrice e gli applica una trasformazione lineare invertibile

4. Lo step *AddRoundKey* esegue lo `xor` tra la chiave di round generata e la matrice di stato

La sbox della primitiva subbytes serve a introdurre non linearità nella cifratura e si basa sull'inversione del polinomio che si ottiene considerando i bit che compongono il byte in ingresso come i coefficienti di un polinomio su $GF(2^8)$.

Esistono diversi tipi di implementazioni della sbox, ad esempio é possibile mappare esaustivamente tutti i 256 possibili valori in ingresso con i relativi valori in output utilizzando una rete di multiplexer e demultiplexer i cui selettori verranno collegati ai bit in ingresso. I questo modo i selettori abiliteranno o disabiliteranno dei collegamenti tra i diversi elementi della rete permettendo di ottenere in output il valore corretto corrispondente all'output della sbox. Questa soluzione presenta vantaggi in termini di velocità in quanto i valori sono già tutti mappati nei collegamenti della rete e i tempi d'accesso a questi valori sono molto rapidi ma presenta svantaggi legati all'area occupata dal circuito vero e proprio.

L'implementazione proposta in questa sezione esegue il calcolo dell' inverso moltiplicativo del byte in ingresso suddividendo il calcolo sui due nibble che lo compongono così da operare su $GF(2^4)$ e con lo stesso procedimento arrivare ad operare su $GF(2^2)$. Questa soluzione risulta essere significativamente compatta, per quanto più lenta di quella a mappatura esaustiva realizzata tramite multiplexer e demultipliexer.

3.3.1 Implementazione

Durante la fase di cifratura l'operazione di subbytes prevede di calcolare l'inverso moltiplicativo degli 8 bit in ingresso in $GF(2^8)$ e successivamente di applicare la trasformazione affine TA. In decifratura dovrà essere applicata prima la trasformazione affine inversa TA^{-1} e poi verrà effettuata l'inversione in $GF(2^8)$, in modo da eseguire l'operazione di subbytes inversa.

Una possibile struttura per realizzare la sbox in hardware è quella presente in Figura 3.8, dove si nota la possibilità di riutilizzare il circuito che realizza l'inverso moltiplicativo su GF(2^8) per realizzare sia la sbox diretta, che quella inversa, al costo di inserire due multiplexer.

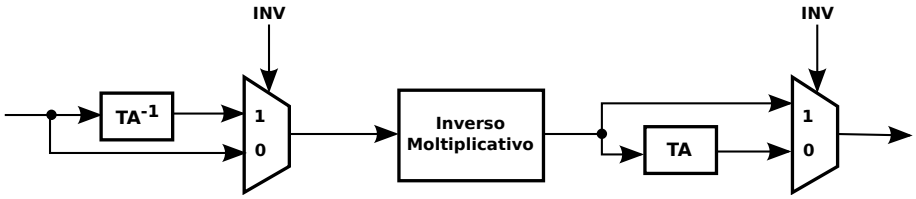


Figura 3.8: Schema blocco subbytes dell'AES

Trasformazione Affine

La trasformazione affine applicata in cifratura prevede di prendere gli otto bit in ingresso $i_7, \ldots, i_0$ e calcolare i relativi bit in uscita $o_7, \ldots, o_0$ applicando la seguente formula:

$$o_i = i_i \oplus i_{(i+4)mod8} \oplus i_{(i+5)mod8} \oplus i_{(i+6)mod8} \oplus i_{(i+7)mod8} \oplus c_i \tag{3.1}$$

per $0 \leq i < 8$ e $c = (01100011)_2$.

Questa trasformazione affine può essere rappresentata in forma matriciale in questo modo:

$$AT(i) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} i_7 \\ i_6 \\ i_5 \\ i_4 \\ i_3 \\ i_2 \\ i_1 \\ i_0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \tag{3.2}$$

in cui tutti gli elementi della matrice sono elementi di GF(2), perciò il prodotto e la somma tra due elementi corrispondono alla `and` e alla `xor` bit a bit.

La formula della trasformazione affine può quindi essere riscritta nel

seguente modo:

$$AT(i) = \begin{pmatrix} i_7 \oplus i_6 \oplus i_5 \oplus i_4 \oplus i_3 \\ i_6 \oplus i_5 \oplus i_4 \oplus i_3 \oplus i_2 \oplus 1 \\ i_5 \oplus i_4 \oplus i_3 \oplus i_2 \oplus i_1 \oplus 1 \\ i_4 \oplus i_3 \oplus i_2 \oplus i_1 \oplus i_0 \\ i_7 \oplus i_3 \oplus i_2 \oplus i_1 \oplus i_0 \\ i_7 \oplus i_6 \oplus i_2 \oplus i_1 \oplus i_0 \\ i_7 \oplus i_6 \oplus i_5 \oplus i_1 \oplus i_0 \oplus 1 \\ i_7 \oplus i_6 \oplus i_5 \oplus i_4 \oplus i_0 \oplus 1 \end{pmatrix} \tag{3.3}$$

da cui é possibile ricavare il corrispondente circuito.

Trasformazione Affine Inversa

L'operazione di trasformazione affine inversa esegue l'operazione inversa della trasformazione affine usata durante l'operazione di cifratura.

La rappresentazione matriciale di questa funzione é mostrata dalla 3.4

$$AT^{-1}(i) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} i_7 \\ i_6 \\ i_5 \\ i_4 \\ i_3 \\ i_2 \\ i_1 \\ i_0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \tag{3.4}$$

Gli elementi di questa trasformazione affine inversa sono tutti elementi di GF(2) a cui ad ogni somma corrisponde una `xor` e ad ogni moltiplicazione corrisponde una `and`.

La formula 3.4 può essere riscritta in questo modo in cui sono evidenziate le operazioni bit a bit necessarie per la realizzazione del corrispondente

circuito

$$AT^{-1}(i) = \begin{pmatrix} i_6 \oplus i_4 \oplus i_1 \\ i_5 \oplus i_3 \oplus i_0 \\ i_7 \oplus i_4 \oplus i_2 \\ i_6 \oplus i_3 \oplus i_1 \\ i_5 \oplus i_2 \oplus i_0 \\ i_7 \oplus i_4 \oplus i_1 \oplus 1 \\ i_6 \oplus i_3 \oplus i_0 \\ i_7 \oplus i_5 \oplus i_2 \oplus 1 \end{pmatrix} \tag{3.5}$$

Inverso Moltiplicativo

La funzione di questo blocco é quella di eseguire l'operazione di calcolo dell'inverso moltiplicativo dei bit in ingresso considerandoli un polinomio con elementi in $GF(2^8)$. La strategia utilizzata é quella di effettuare la scomposizione di questo polinomio in un polinomio con elementi in un GF di ordine minore, eseguire l'inversione su quel GF e infine riconvertire il polinomio ottenuto in un polinomio con elementi in $GF(2^8)$.

Prima di mostrare lo schema del circuito che effettua l'inverso moltiplicativo sui campi finiti in $GF(2^8)$ é necessario comprenderne il funzionamento.

Ognuno dei singoli bit del byte in ingresso può essere visto come un coefficiente di un polinomio in $GF(2^8)$, ad esempio i bit in ingresso $(10001011)_2$ rappresentano il polinomio $i^7 + i^3 + i + 1$ in $GF(2^8)$. Le operazioni eseguibili su questo GF sono la somma e la moltiplicazione. La somma viene eseguita applicando lo `xor` bit a bit tra i termini del polinomio mentre la moltiplicazione viene eseguita tenendo in considerazione tutti i prodotti incrociati tra i termini del polinomio. Per evitare questi prodotti incrociati si può scomporre il polinomio fino ad arrivare ad avere elementi in GF(2), dove l'operazione di somma corrisponde ancora alla `xor` mentre la moltiplicazione corrisponde direttamente alla `and` bit a bit.

Dato un polinomio irriducibile $x^2 + Ax + B$ con elementi in $GF(2^4)$ é possibile riscrivere un qualsiasi polinomio di $GF(2^8)$ nella forma $bx + c$ dove b e c sono elementi di $GF(2^4)$. É possibile notare che si può procedere al contrario ricavando il valore di un elemento di $GF(2^8)$ a partire dagli elementi del polinomio corrispondente in $GF(2^4)$.

Allo stesso modo é possibile procedere con la scomposizione di un polinomio con elementi in $GF(2^4)$ in un polinomio con elementi in $GF(2^2)$ e poi alla scomposizione di quest'ultimo in un polinomio con elementi in $GF(2)$.

Il vantaggio di effettuare queste scomposizioni risiede nel fatto che il calcolo dell'inverso moltiplicativo agirà su polinomi con coefficienti sempre più piccoli e nel caso delle operazioni di somma e moltiplicazione in GF(2) sarà immediato ricavare il circuito corrispondente in quanto corrispondono rispettivamente alle operazioni di `xor` e `and` bit a bit.

I polinomi irriducibili scelti per effettuare queste scomposizioni sono indicati nella Tabella 3.3

Tabella 3.3: Tabella polinomi irriducibili

GF	Polinomio
$GF(2^4)$	$x^2 + x + \lambda$ con $\lambda = (1100)_2$
$GF(2^2)$	$x^2 + x + \varphi$ con $\varphi = (10)_2$
$GF(2)$	$x^2 + x + 1$

Una volta scomposto il polinomio corrispondente al byte in ingresso é necessario effettuare il calcolo dell'inverso moltiplicativo vero e proprio. Per eseguire questa operazione si può utilizzare la seguente formula:

$$(bx + c)^{-1} = b(b^2B + bcA + c^2)^{-1}x + (c + bA)(b^2B + bcA + c^2)^{-1} \quad (3.6)$$

la cui identità é verificata con i seguenti passaggi:

$$(bx + c)^{-1} = b(b^2B + bcA + c^2)^{-1}x + (c + bA)(b^2B + bcA + c^2)^{-1}$$

$$(bx + c)^{-1} = (bx + c + BA)(b^2B + bcA + c^2)^{-1}$$

$$(b^2B + bcA + c^2) = (bx + c + bA)(bx + c)$$

$$b^2B + bcA + c^2 = b^2x^2 + bcx + bcx + c^2 + b^2Ax + bcA$$

la somma in $GF(2^4)$ corrisponde allo `xor` degli elementi quindi $bcx + bcx$ si annullano ottenendo

$$b^2B + bcA + c^2 = b^2x^2 + c^2 + b^2Ax + bcA$$

il polinomio irriducibile $x^2 + Ax + B$ utilizzato come modulo implica che $x^2 + Ax + B = 0$ e quindi $x^2 = Ax + B$, perciò

$$b^2B + bcA + c^2 = b^2(Ax + B) + c^2 + b^2Ax + bcA$$

$$b^2B + bcA + c^2 = b^2Ax + b^2B + c^2 + b^2Ax + bcA$$

da cui si prova l'identità annullando i due termini b^2Ax e riordinando i termini.

Il polinomio irriducibile scelto per la scomposizione in $GF(2^4)$ é $x^2+x+\lambda$ quindi, ponendo $A = 1$ e $B = \lambda = (1100)_2$, l'equazione 3.6 può essere riscritta nella 3.7.

$$(bx + c)^{-1} = b(b^2\lambda + bc + c^2)^{-1}x + (c + b)(b^2\lambda + bc + c^2)^{-1} \tag{3.7}$$

Nell'equazione 3.7 sono presenti operazioni di somma, moltiplicazione, elevamento al quadrato e inverso moltiplicativo con elementi in $GF(2^4)$.

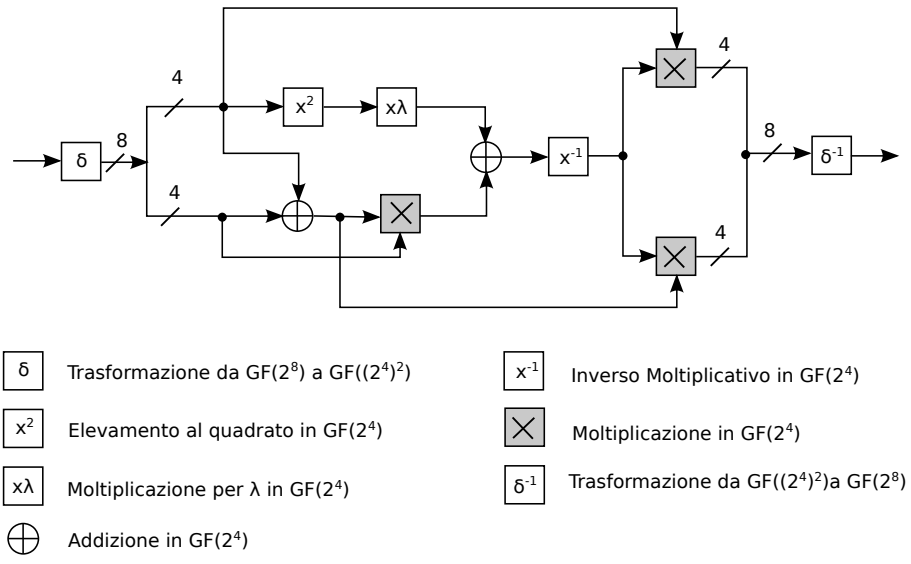


Figura 3.9: Schema inverso moltiplicativo AES

Ognuna delle operazioni dell'equazione 3.7 può essere quindi convertita

in un singolo blocco ed essere utilizzato per sviluppare lo schema dell'inverso moltiplicativo. Lo schema e la legenda di ogni singolo blocco del calcolo dell'inverso moltiplicativo é presente nella Figura 3.9 e il loro funzionamento e struttura verrà mostrato nelle prossime sezioni.

Trasformazione da $GF(2^8)$ a $GF((2^4)^2)$ e viceversa

L'idea alla base che permette di trasformare un elemento da $GF(2^8)$ a $GF((2^4)^2)$ é quella di cambiare la rappresentazione del polinomio che lo rappresenta utilizzando un isomorfismo. Un elemento $a \in GF(2^8)$ può essere rappresentato con un polinomio $bx+c$ con $b, c \in GF(2^4)$. Questa conversione funziona perché viene applicato un isomorfismo che preserva il risultato delle operazioni di somma e moltiplicazione sia nella rappresentazione originale sia nella nuova rappresentazione.

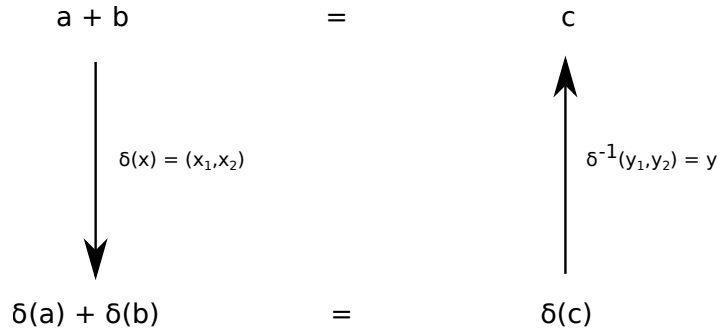


Figura 3.10: Isomorfismo diretto e inverso

La figura 3.10 mostra come la somma di due polinomi a e b restituisca sempre il polinomio c sia eseguendo il calcolo diretto sia passando attraverso l'isomorfismo, la somma utilizzando la nuova rappresentazione e infine attraverso l'isomorfismo inverso.

L'isomorfismo diretto, il cui procedimento per ricavare le equazioni é mostrato in [12], é effettuato dalla funzione δ espressa dalla formula 3.8.

$$\delta(i) = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} i_7 \\ i_6 \\ i_5 \\ i_4 \\ i_3 \\ i_2 \\ i_1 \\ i_0 \end{pmatrix} \tag{3.8}$$

L'equazione 3.8 si traduce in un circuito utilizzando le porte `xor` come mostrato nell'equazione 3.9

$$\delta(i) = \begin{pmatrix} i_7 \oplus i_5 \\ i_7 \oplus i_6 \oplus i_4 \oplus i_3 \oplus i_2 \oplus i_1 \\ i_7 \oplus i_5 \oplus i_3 \oplus i_2 \\ i_7 \oplus i_5 \oplus i_3 \oplus i_2 \oplus i_1 \\ i_7 \oplus i_6 \oplus i_2 \oplus i_1 \\ i_7 \oplus i_4 \oplus i_3 \oplus i_2 \oplus i_1 \\ i_6 \oplus i_4 \oplus i_1 \\ i_6 \oplus i_1 \oplus i_0 \end{pmatrix} \tag{3.9}$$

Al termine del calcolo di inversione é necessario ricostruire l'elemento in $GF(2^8)$. Per eseguire questa operazioni si usa la funzione δ^{-1}, ovvero l'inversa della δ, espressa dalla funzione 3.10.

$$\delta^{-1}(i) = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} i_7 \\ i_6 \\ i_5 \\ i_4 \\ i_3 \\ i_2 \\ i_1 \\ i_0 \end{pmatrix} \tag{3.10}$$

Questa funzione inversa viene convertita in circuito utilizzando le porte `xor` come mostrato nella 3.11.

$$\delta^{-1}(i) = \begin{pmatrix} i_7 \oplus i_6 \oplus i_5 \oplus i_1 \\ i_6 \oplus i_2 \\ i_6 \oplus i_5 \oplus i_1 \\ i_6 \oplus i_5 \oplus i_4 \oplus i_2 \oplus i_1 \\ i_5 \oplus i_4 \oplus i_3 \oplus i_2 \oplus i_1 \\ i_7 \oplus i_4 \oplus i_3 \oplus i_2 \oplus i_1 \\ i_5 \oplus i_4 \\ i_6 \oplus i_5 \oplus i_4 \oplus i_2 \oplus i_0 \end{pmatrix} \tag{3.11}$$

Addizione in $GF(2^4)$

La somma di due elementi di $GF(2^4)$ si effettua eseguendo lo `xor` bit a bit dei due elementi in quanto somma di due polinomi con coefficienti binari.

Elevamento al quadrato in $GF(2^4)$

Sia $k = q^2$, dove k e q sono due elementi di $GF(2^4)$ con coefficienti rappresentati da due numeri binari da 4 cifre $(k_3k_2k_1k_0)_2$ e $(q_3q_2q_1q_0)_2$.

Si procede alla scomposizione del polinomio del quadrato con i seguenti passaggi

$$k = \left(\underbrace{q_3q_2}_{q_H}\underbrace{q_1q_0}_{q_L} \right)^2 = (q_Hx + q_L)^2$$

$$k = {q_H}^2x^2 + q_Hq_Lx + q_Lq_Hx + {q_L}^2$$

$$k = {q_H}^2x^2 + {q_L}^2$$

Il termine x^2 può essere ridotto utilizzando il polinomio $x^2 + x + \varphi$ con $\varphi = (10)_2$. Si pone $x^2 = x + \varphi$ e si sostituisce ottenendo

$$k = {q_H}^2 (x + \varphi) + {q_L}^2$$

$$k = \underbrace{{q_H}^2}_{k_H}x + \underbrace{{q_H}^2\varphi + {q_L}^2}_{k_L} \in GF(2^2)$$

Questa equazione é composta solo da elementi di $GF(2^2)$. Procedendo nello stesso modo é possibile scomporre k_H e k_L in elementi di $GF(2)$ e arivare alla formula operativa per calcolare l'elevamento al quadrato vero e proprio.

$$k_H = {q_H}^2 = (q_3 q_2)^2 = (q_3 x + q_2)^2$$

$$k_H = {q_3}^2 x^2 + q_3 q_2 x + q_3 q_2 x + {q_2}^2 = q_3 x^2 + q_2$$

Il termine x^2 può essere sostituito usando il polinomio irriducibile $x^2 + x + 1$. Ponendo $x^2 = x + 1$ si ottengono le seguenti equazioni.

$$k_H = q_3 (x + 1) + q_2$$

$$k_H = \underbrace{q_3}_{k_3} x + \underbrace{(q_3 + q_2)}_{k_2} \tag{3.12}$$

Seguendo dei passaggi analoghi si scompone k_L in elementi di $GF(2^2)$ ricordando che $\varphi = (10)_2$ come indicato nella Tabella 3.3.

$$k_L = {q_H}^2 \varphi + {q_L}^2 = (q_3 q_2)^2 (10)_2 + (q_1 q_0)^2$$

$$k_L = (q_3 x + q_2)^2 (1x + 0) + (q_1 x + q_0)^2$$

$$k_L = \left({q_3}^2 x^2 + q_2 q_3 x + q_2 q_3 x + {q_2}^2\right) x + \left({q_1}^2 x^2 + q_0 q_1 + q_0 q_1 + {q_0}^2\right)$$

$$k_L = \left(q_3 x^2 + q_2\right) x + \left(q_1 x^2 + q_0\right)$$

$$k_L = q_3 x^3 + q_2 x + q_1 x^2 + q_0$$

In questo caso bisogna eseguire due sostituzioni. La prima riguarda x^2 che si risolve ponendo $x^2 = x + 1$. La seconda riguarda x^3 in cui bisogna porre $x^3 = x^2 x$ da cui segue $x^3 = (x + 1) x = x^2 + x = x + 1 + x = 1$.

$$k_L = q_3 (1) + q_2 x + q_1 (x + 1) + q_0$$

$$k_L = \underbrace{(q_2 + q_1)}_{k_1} x + \underbrace{q_3 + q_1 + q_0}_{k_0} \in GF(2) \tag{3.13}$$

Unendo i risultati esposti nella 3.12 e nella 3.13 si ottengono le formule finali per calcolare l'elevamento al quadrato in $GF(2^4)$ utilizzando elementi

di $GF(2)$.

$$\begin{cases} k_3 = q_3 \\ k_2 = q_3 + q_2 \\ k_1 = q_2 + q_1 \\ k_0 = q_3 + q_1 + q_0 \end{cases}$$

Lo schema circuitale si ricava sostituendo ad ogni somma in $GF(2)$ una porta `xor`.

$$\begin{cases} k_3 = q_3 \\ k_2 = q_3 \oplus q_2 \\ k_1 = q_2 \oplus q_1 \\ k_0 = q_3 \oplus q_1 \oplus q_0 \end{cases}$$

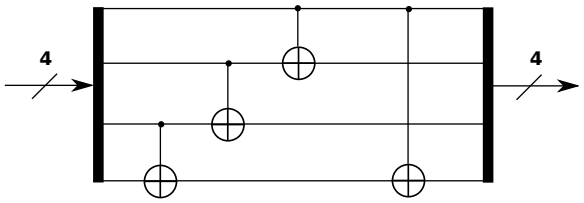


Figura 3.11: Schema elevamento al quadrato in $GF(2^4)$

Lo schema circuitale risultante é mostrato nella Figura 3.11

Moltiplicazione per λ in $GF(2^4)$

Il circuito per calcolo della moltiplicazione di un elemento di $GF(2^4)$ per $\lambda = (1100)_2$ si può ricavare con passaggi simili a quelli mostrati nella precedente sezione.

Sia $k = q\lambda$, con $k = (k_3k_2k_1k_0)_2$ e $q = (q_3q_2q_1q_0)_2$ elementi di $GF(2^4)$.

$$k = \left(\underbrace{q_3q_2}_{q_H}\underbrace{q_1q_0}_{q_L} \right) \left(\underbrace{11}_{\lambda_H}\underbrace{00}_{\lambda_L} \right)$$

$$k = (q_Hx + q_L)(\lambda_Hx + \lambda_L)$$

Nelle precedente equazione λ_L si é potuto semplificare perché $\lambda_L = (00)_2$

$$k = q_H\lambda_Hx^2 + q_L\lambda_Hx$$

Il calcolo procede sostituendo $x^2 = x + \varphi$ usando il polinomio indicato nella Tabella 3.3

$$k = q_H \lambda_H (x + \varphi) + q_L \lambda_H x$$

$$k = \underbrace{(q_H \lambda_H + q_L \lambda_H)}_{k_H} x + \underbrace{q_H \lambda_H \varphi}_{k_L} \in GF(2^2)$$

Si procede ora al calcolo individuale di k_H e k_L

$$k_H = q_H \lambda_H + q_L \lambda_H$$

$$k_H = (q_3 q_2)((11)_2) + (q_1 q_0)((11)_2)$$

$$k_H = (q_3 x + q_2)(x + 1) + (q_1 x + q_0)(x + 1)$$

$$k_H = (q_3 x + q_2)(x + 1) + (q_1 x + q_0)(x + 1)$$

$$k_H = q_3 x^2 + (q_3 + q_2) x + q_2 + q_1 x^2 + (q_1 + q_0) x + q_0$$

Sostituendo $x^2 = x + 1$ si ottiene la formula finale di k_H in $GF(2)$.

$$k_H = q_3 (x + 1) + (q_3 + q_2) x + q_2 + q_1 (x + 1) + (q_1 + q_0) x + q_0$$

$$k_H = (q_3 + q_3 + q_2 + q_1 + q_1 + q_0) x + (q_3 + q_2 + q_1 + q_0)$$

$$k_H = \underbrace{(q_2 + q_0)}_{k_3} x + \underbrace{(q_3 + q_2 + q_1 + q_0)}_{k_2} \in GF(2) \tag{3.14}$$

Si segue lo stesso procedimento per scomporre k_L in $GF(2)$.

$$k_L = q_H \lambda_H \varphi$$

$$k_L = (q_3 q_2)((11)_2)((10)_2)$$

$$k_L = (q_3 x + q_2)(x + 1)(x)$$

$$k_L = q_3 x^3 + q_2 x^2 + q_3 x^2 + q_2 x$$

Il termine x^2 si può sostituire usando $x^2 = x + 1$, mentre per x^3 si usa la sostituzione $x^3 = 1$ come già visto nella precedente sezione.

$$k_L = q_3\,(1) + q_2\,(x+1) + q_3\,(x+1) + q_2 x$$

$$k_L = (q_3 + q_2 + q_2)\,x + q_3 + q_3 + q_2$$

$$k_L = \underbrace{q_3}_{k_1} x + \underbrace{q_2}_{k_0} \in GF(2) \tag{3.15}$$

Unendo i risultati esposti nella 3.14 e nella 3.15 si ottengono le formule finali per calcolare la moltiplicazione per λ in $GF(2^4)$ utilizzando elementi di $GF(2)$.

$$\begin{cases} k_3 = q_2 + q_0 \\ k_2 = q_3 + q_2 + q_1 + q_0 \\ k_1 = q_3 \\ k_0 = q_2 \end{cases}$$

Ricordandosi che le somme in $GF(2)$ equivalgono all'`xor` logico degli elementi si ottiene lo schema circuitale finale di questo blocco.

$$\begin{cases} k_3 = q_2 \oplus q_0 \\ k_2 = q_3 \oplus q_2 \oplus q_1 \oplus q_0 \\ k_1 = q_3 \\ k_0 = q_2 \end{cases}$$

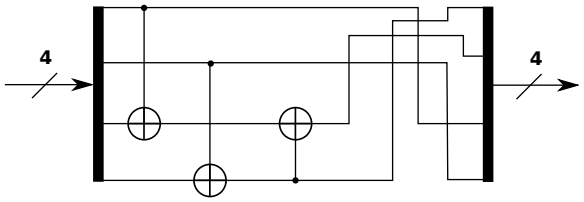


Figura 3.12: Schema moltiplicazione per λ in $GF(2^4)$

Lo schema del circuito finale di questo blocco é mostrato nella Figura 3.12

Moltiplicazione in $GF(2^4)$

Sia $k = qw$, dove $k = (k_3k_2k_1k_0)_2$, $q = (q_3q_2q_1q_0)_2$ e $w = (w_3w_2w_1w_0)_2$ sono elementi di $GF(2^4)$.

$$k = \left(\underbrace{q_3 q_2 q_1 q_0}_{q_H \quad q_L} \right) \left(\underbrace{w_3 w_2 w_1 w_0}_{w_H \quad w_L} \right) = (q_H x + q_L)(w_H x + w_L)$$

$$k = (q_H w_H) x^2 + (q_H w_L + q_L w_H) x + q_L w_L$$

Sostituendo il termine x^2 utilizzando $x^2 = x + \varphi$ prosegue con le seguenti formule.

$$k = (q_H w_H)(x + \varphi) + (q_H w_L + q_L w_H) x + q_L w_L$$

$$k = \left(\underbrace{q_H w_H + q_H w_L + q_L w_H}_{k_H} \right) x + \underbrace{q_H w_H \varphi + q_L w_L}_{k_L} \in GF(2^2) \qquad (3.16)$$

L'equazione 3.16 contiene somme e moltiplicazioni con elementi in $GF(2^2)$. L'addizione in questo campo finito equivale allo `xor` logico dei bit dei due elementi, mentre la moltiplicazione richiede la scomposizione in $GF(2)$.

Un'alternativa alla scomposizione in $GF(2)$ che potrebbe generare equazioni troppo lunghe e complesse é ricavare lo schema circuitale della moltiplicazione in $GF(2^4)$ dalla formula 3.16.

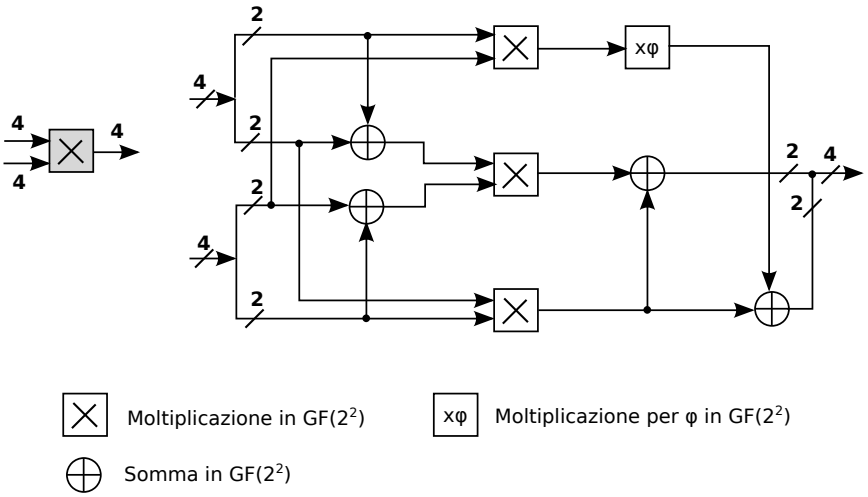


Figura 3.13: Schema moltiplicazione in $GF(2^4)$

La Figura 3.13 mostra come la moltiplicazione in $GF(2^4)$ richieda di ricavare in $GF(2^2)$ solo i blocchi che eseguono la moltiplicazione di due

elementi e la moltiplicazione per φ. Gli schemi circuitali di questi due blocchi sono ricavati nelle prossime sezioni.

Addizione in $GF(2^2)$

La somma di due elementi di $GF(2^2)$ si effettua eseguendo lo `xor` tra i bit dei due elementi in quanto somma di due polinomi con coefficienti binari.

Moltiplicazione in $GF(2^2)$

Sia $k = qw$, dove $k = (k_1k_0)_2$, $q = (q_1q_0)_2$ e $w = (w_1w_0)_2$ sono elementi di $GF(2^2)$.

$$k = (q_1q_0)(w_1w_0) = (q_1x + q_0)(w_1x + w_0)$$

$$k = q_1w_1x^2 + q_0w_1x + q_1w_0x + q_0w_0$$

Il calcolo prosegue sostituendo x^2 grazie all'equazione $x^2 = x + 1$.

$$k = q_1w_1(x + 1) + q_0w_1x + q_1w_0x + q_0w_0$$

$$k = \underbrace{(q_1w_1 + q_0w_1 + q_1w_0)}_{k_1}x + \underbrace{q_1w_1 + q_0w_0}_{k_0} \in GF(2) \tag{3.17}$$

Dall'equazione 3.17 si ricavano le formule finale della moltiplicazione di due elementi di $GF(2)$.

$$\begin{cases} k_1 = q_1w_1 + q_0w_1 + q_1w_0 \\ k_0 = q_1w_1 + q_0w_0 \end{cases}$$

Queste formule possono essere convertite in un circuito sostituendo ogni moltiplicazioni con una porta logica `and` e ogni somma con una porta logica `xor`.

$$\begin{cases} k_1 = q_1w_1 \oplus q_0w_1 \oplus q_1w_0 \\ k_0 = q_1w_1 \oplus q_0w_0 \end{cases}$$

Il circuito utilizzato per implementare le precedenti equazioni é illustrato nella Figura 3.14.

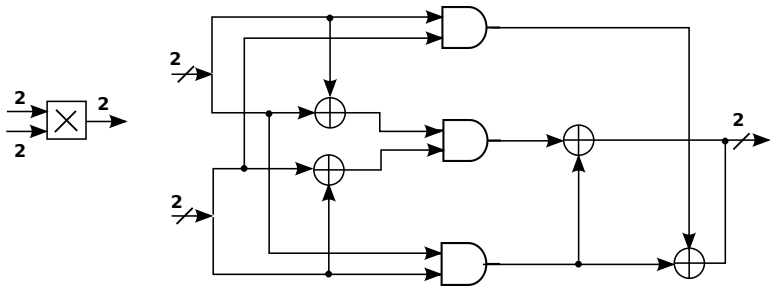


Figura 3.14: Schema moltiplicazione in $GF(2^2)$

Moltiplicazione per φ in $GF(2^2)$

Sia $k = q\varphi$, dove $k = (k_1 k_0)_2$, $q = (q_1 q_0)_2$ e $\varphi = (10)_2$ sono elementi di $GF(2^2)$.

$$k = (q_1 q_0)\left((10)_2\right) = (q_1 x + q_0)\, x = q_1 x^2 + q_0 x$$

Effettuando la sostituzione del termine x^2 utilizzando $x^2 = x + 1$ si ottengono le formule finali di questo blocco.

$$k = q_1 (x + 1) + q_0 x$$

$$k = \underbrace{(q_1 + q_0)}_{k_1} x + \underbrace{q_1}_{k_0} \in GF(2) \tag{3.18}$$

L'equazione 3.18 permette di ottenere le equazioni finali della moltiplicazione per φ in $GF(2^2)$.

$$\begin{cases} k_1 = q_1 + q_0 \\ k_0 = q_1 \end{cases}$$

Come già accaduto nei precedenti paragrafi si sostituiscono le somme con le porte `xor` per ricavare lo schema circuitale.

$$\begin{cases} k_1 = q_1 \oplus q_0 \\ k_0 = q_1 \end{cases}$$

Lo schema circuitale finale del circuito é rappresentato dalla Figura 3.15.

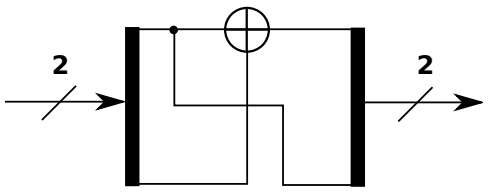


Figura 3.15: Schema moltiplicazione per una costante in $GF(2^2)$

Inverso Moltiplicativo in $GF(2^4)$

Sia q un elemento di $GF(2^4)$. L'inverso moltiplicativo di q é quell'elemento $q^{-1} = \left(q_3^{-1}, q_2^{-1}, q_1^{-1}, q_0^{-1}\right)$ tale che $\left(q^{-1}\right) q = 1 \in GF(2^4)$.

Le formule per ricavare il circuito dell'inverso moltiplicativo di un elemento in $GF(2^4)$ si ricavano seguendo la stessa metodologia di scomposizione in GF di ordine inferiore utilizzata per ricavare l'inversione $GF(2^8)$. Le formule 3.19 mostrano le equazioni finali per ricavare l'inverso moltiplicativo, come mostrato in [14].

$$
\begin{aligned}
q_3^{-1} &= q_3 + q_3q_2q_1 + q_3q_0 + q_2 \\
q_2^{-1} &= q_3q_2q_1 + q_3q_2q_0 + q_3q_0 + q_2 + q_2q_1 \\
q_1^{-1} &= q_3 + q_3q_2q_1 + q_3q_1q_0 + q_2 + q_2q_0 + q_1 \\
q_0^{-1} &= q_3q_2q_1 + q_3q_2q_0 + q_3q_1 + q_3q_1q_0 + q_3q_0 + q_2 + q_2q_1 + q_2q_1q_0 + q_1 + q_0
\end{aligned}
\tag{3.19}
$$

Il circuito si ricava sostituendo una porta `and` per ogni moltiplicazione e una porta `xor` per ogni somma come gia visto nell'analisi dei precedenti blocchi.

3.3.2 Applicazione della protezione contro side channel alla sbox

La versione protetta tramite masking della sbox introdotta nella precedente sezione é stata implementata adeguando tutti i blocchi come già mostrato nelle sezioni del sommatore e del moltiplicatore.

Ogni porta logica di ogni singolo blocco esposto finora verrà adeguata con la versione protetta dal masking ISW e verrà aggiunto un bus ausiliario per permettere la propagazione dei segnali casuali necessari per l'elaborazione

protetta delle funzioni logiche delle porte. Inoltre ogni singola connessione rappresentante un bit verrà sostituita da un bus contenente le share.

Conclusione

In questo capitolo sono state mostrati nel dettaglio le architetture e i principi di funzionamento dei circuiti, con e senza masking ISW, su cui verranno applicati gli attacchi CPA.

Le prestazioni di questi circuiti verranno analizzate nel Capitolo 4 mentre i risultati ottenuti dagli attacchi CPA verranno analizzati nel Capitolo 5.

Capitolo 4

Risultati Sperimentali

In questo capitolo verrà descritta la metodologia usata per ottenere la validazione funzionale dei circuiti realizzati ed esaminare la loro resistenza ad attacchi side channel. Nella prima sezione verrà mostrato il flusso di lavoro delle simulazioni, ovvero come é stato svolto tutto il lavoro a partire dall'implementazione dei circuiti in VHDL fino ad arrivare agli attacchi CPA. Nella seconda sezione verranno analizzate le prestazioni offerte dai circuiti implementati approfondendo in particolar modo l'impatto dell'introduzione del masking ISW sull'area occupata e sul cammino critico.

4.1 Flusso di lavoro delle simulazioni

In questa sezione verranno analizzate le fasi necessarie per arrivare ad applicare gli attacchi CPA ai circuiti introdotti nel capitolo precedente. Il flusso di lavoro di tutte le attività é mostrato nella Figura 4.1.

4.1.1 Sviluppo circuiti VHDL

Il VHDL è un linguaggio di descrizione ad alto livello utilizzato per la specifica e la progettazione di circuiti elettronici digitali, standardizzato in [13]. Permette di modellare i circuiti elettronici come entità che presentano un numero prefissato di input e di output.

In questo elaborato sono state tradotte in VHDL tutte le entità presenti nel sommatore, nel moltiplicatore e nella sbox dell'AES. Sono state implementate le entità a partire dalle piu semplici porte logiche alle entità

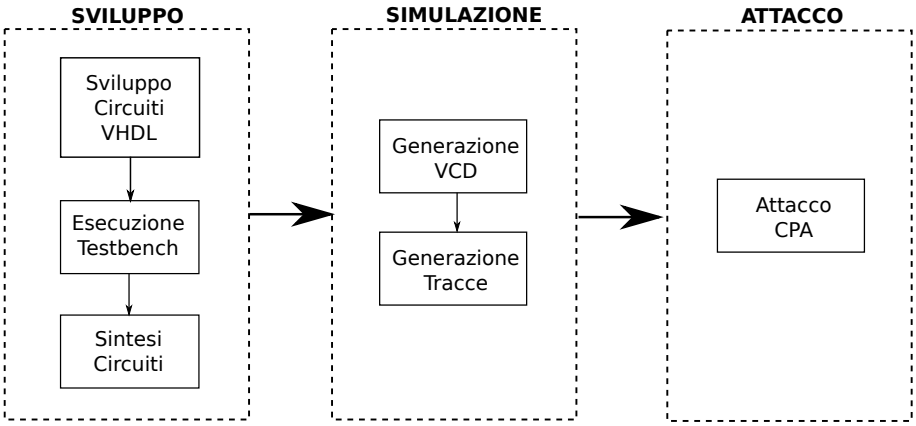


Figura 4.1: Flusso di lavoro delle simulazioni

che rappresentano i circuiti finali desiderati con le architetture mostrate nel Capitolo 3.

Le porte logiche elementari normali e protette sono state sviluppate in VHDL in modo *behavioural*, ovvero modellando il comportamento desiderato di ogni entità descrivendo la relazione tra input e output. Tutte le altre entità sono state sviluppate in modo *structural*, ovvero organizzando e connettendo altre entità in modo da realizzare la funzione desiderata.

Una volta implementate tutte le entità dei tre circuiti normali, sono state ricavate le relative versioni protette. Sono state per prima cosa implementate le entità responsabili di convertire un singolo bit nella versione a share e viceversa, dopodiché tutte le entità protette sono state ricavate da quelle in chiaro sostituendo ad ogni bit il relativo bus di share e ad ogni porta logica elementare l'entità della relativa porta logica protetta.

Per lo sviluppo del codice VHDL e successivamente per tutte le attività di simulazione funzionale a livello logico é stato usato il software ModelSim PE Student Edition 10.4 eseguito su una macchina con processore Intel Core i7-3632QM e 8GB di memoria RAM.

4.1.2 Esecuzione testbench

In questa fase sono state testate tutte le singole entità normali e protette utilizzando dei testbench scritti in VHDL. Il test di ogni entità prevede di applicare degli input noti e di confrontare l'output del circuito con l'output

atteso per quei particolari input. Se ogni asserzione di uguaglianza tra l'output effettivo del circuito e l'output atteso é rispettata allora il circuito si può considerare corretto.

I valori di input e output attesi per ogni test sono stati generati tramite script scritti in *Python* e salvati in file testuali letti durante le simulazioni VHDL. Gli input sono stati assegnati ai bit delle entità corrispondenti mentre gli output attesi sono stati confrontati con gli output effettivi del circuito.

Il testbench del sommatore a 32 bit é stato realizzato testando 1000 esecuzioni dell'algoritmo del circuito. Il valore del primo addendo é stato scelto in maniera casuale ad ogni iterazione del test mentre il secondo addendo é rimasto fisso per tutte le 1000 iterazioni. La stessa strategia di test é stata applicata al moltiplicatore dove il primo fattore varia in ciascuna delle 1000 iterazioni mentre il secondo fattore rimane fisso per tutto il testbench. Il test effettuato per la sbox é leggermente diverso in quanto l'input é composto da 8 bit e può essere testato in maniera esaustiva per tutte le 256 possibili combinazioni.

I testbench eseguiti per le entità intermedie dei circuiti sono serviti per certificarne il corretto funzionamento mentre i testbench eseguiti per le entità finali del sommatore, del moltiplicatore e della sbox sono serviti non solo per certificarne il corretto funzionamento ma anche come base per eseguire gli attacchi. I testbench per tutte le entità, dalle più semplici fino a quelle dei circuiti finali, sono stati utilizzati per testare sia i circuiti normali sia i circuiti protetti con l'accortezza di aggiungere un bus dedicato alle share random necessarie per le porte logiche protette.

Le share casuali necessarie per l'elaborazione protetta sono state impostate come input aggiuntivi nei file con i valori letti durante le simulazioni VHDL. Questa scelta é accettabile nel contesto di questo elaborato in quanto sono eseguite solamente delle simulazioni dei circuiti e non delle rilevazioni di grandezze fisiche reali. Nell'implementazione reale di questo tipo di circuiti i valori delle share casuali saranno generate tramite dei circuiti dedicati chiamati *generatori di numeri pseudo-casuali* che richiederanno una loro implementazione dedicata andando ad occupare una porzione dello spazio nel circuito.

4.1.3 Sintesi circuiti

Una volta generati i file VHDL é necessario procedere alla fase successiva che é quella della sintesi dei circuiti.

Viene utilizzata una libreria di componenti che permette di realizzare la rappresentazione strutturale del circuito, detta *netlist*, la quale contiene solo questi componenti e le informazioni di come sono posizionati e collegati sul circuito finale.

La libreria contiene solo entità di basso livello pronte per essere sintetizzate con tutte le informazioni riguardanti le celle elementari che le compongono. Il tipo di informazioni riguardano le grandezze fisiche delle singole celle elementari quali resistenze, induttanze, capacità e i rispettivi delay che permettono quindi di effettuare simulazioni più realistiche rispetto ai testbench puramente funzionali. In particolare si possono eseguire simulazioni sul consumo di potenza, sull'area occupata dal circuito finale e sui tempi di elaborazione. Sono inoltre presenti le informazioni di come le rispettive porte logiche vanno realizzate in termini di area su silicio che permettono quindi di poter posizionare e collegare i singoli componenti di libreria per effettuare la sintesi vera e propria.

La libreria utilizzata é la libreria FreePDK45 [9], resa disponibile in modo opensource per scopi di ricerca dalla North Carolina State University [10]. Questa libreria permette di eseguire simulazioni funzionali con celle a 45nm e di sintetizzare le netlist. Tutte le netlist da 1 a 8 share sono state generate in circa tre ore utilizzando il software Design Compiler di Synopsys versione 2014.F su una macchina con Intel Core i7-920 e con 12 GB di RAM DDR3-1600.

4.1.4 Generazione VCD

I file di tipo Value Change Dump (VCD) sono dei file basati su codifica ASCII che vengono utilizzati per memorizzare l'andamento di tutti i segnali digitali di un circuito durante una simulazione. Presentano un formato standard composto da tre sezioni.

Nella prima sezione vengono elencate le informazioni generali quali data della simulazione, software utilizzato e timescale. Quest'ultimo rappresenta la risoluzione della simulazione, ovvero il più piccolo intervallo di tempo

per cui é stata registrata una variazione nei segnali. La seconda sezione é la sezione dichiarativa in cui sono presenti le definizioni di tutti i segnali utilizzati nella terza sezione. La terza e ultima sezione presenta i valori veri e propri dei segnali che variano nel corso della simulazione con una risoluzione pari al timescale.

Questi file sono necessari per la fase di generazione delle tracce e racchiudono tutto il comportamento logico di un particolare circuito nel corso di una simulazione, ovvero contengono i cambiamenti di stato di tutti i segnali della netlist. A seconda della risoluzione utilizzata si possono ottenere file di pochi MB così come file da diversi GB. In questo elaborato il VCD più piccolo é quello del sommatore non protetto e misura circa 1MB, ottenuto in pochi minuti, mentre il più grande é quello del moltiplicatore a 4 share con circa 3,5GB ottenuto dopo 20 ore circa.

I file VCD ricavati da questo step sono quelli relativi ai testbench rieseguiti sulle netlist dei circuiti del sommatore, moltiplicatore e sbox normali e protetti. Il software utilizzato é lo stesso utilizzato nella fase di sviluppo e simulazione del codice VHDL impostando un timescale di 1ps.

4.1.5 Generazione tracce

Le tracce di potenza sono la base per eseguire gli attacchi già introdotti nei capitoli precedenti. In questo step é stato utilizzato il software PrimeTime-PX, versione 2014.F sempre della suite Synopsys, che permette di ricavare l'andamento del consumo dinamico di potenza e di salvarlo in file testuali in preparazione agli attacchi CPA veri e propri. PrimeTime é un software che permette di eseguire analisi legate alle prestazioni temporali dei circuiti, PrimeTime-PX é un add-on di PrimeTime che permette di eseguire simulazioni di consumo di potenza dato un circuito sintetizzato e la corrispondente libreria di celle.

La generazione delle tracce di potenza viene realizzata tramite degli script di configurazione in cui é stato necessario specificare la libreria di sintesi utilizzata, le entità coinvolte, il file VCD in ingresso e la risoluzione temporale.

La libreria di sintesi utilizzata é la stessa utilizzata per generare la netlist, ovvero la FreePDK45. L'entità indicata é la netlist principale del circuito in esame il cui testbench ha generato il file VCD. La risoluzione temporale

é necessaria per determinare quanti istanti di tempo vengono elaborati dai dati del file VCD per generare i consumi di potenza dinamica nei file delle tracce di potenza. Uno dei vincoli di PrimeTime é quello di non poter impostare la risoluzione uguale al timescale utilizzato nei file VCD ma di doverla impostare maggiore di almeno un ordine di grandezza, in queste simulazioni é stata impostata una risoluzione di 10ps largamente sufficiente per avere precisione sui successivi attacchi.

I file di output generati contengono una lunga lista di istanti di tempo e i relativi valori di potenza consumati. Analizzando i file VCD con un editor, come ad esempio GTKWave, é possibile notare come i consumi di potenza avvengano solo negli istanti in cui i valori dei segnali commutano dallo stato basso allo stato alto o viceversa. In presenza di segnali costanti i transistor all'interno delle librerie non commutano e il loro consumo dinamico è nullo. Il consumo statico del circuito non è stato preso in considerazione in queste simulazioni

4.1.6 Attacco CPA

La raccolta delle tracce dei consumi di potenza sono il primo step per eseguire gli attacchi CPA introdotti nel Capitolo 2. In questa fase verranno utilizzate per eseguire gli attacchi veri e propri.

Gli attacchi crittografici alle tre unità aritmetico-logiche sono possibili una volta ricavati i circuiti crittografici equivalenti per cui é necessario specificare i quattro elementi fondamentali: algoritmo crittografico, input, chiave segreta e output.

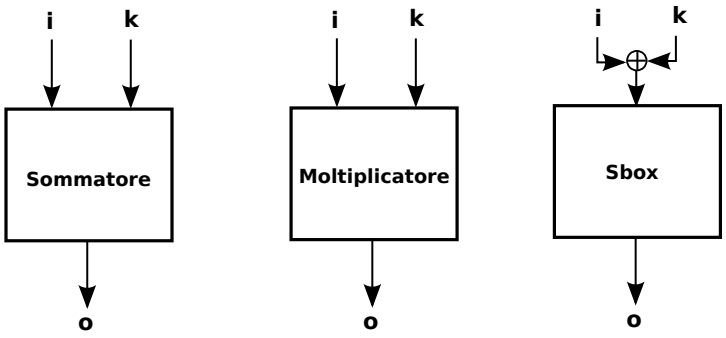


Figura 4.2: Circuiti crittografici

Nel circuito crittografico del sommatore l'algoritmo crittografico corrisponde alla somma di due numeri, l'input é uno dei due addendi, la chiave segreta corrisponde all'altro addendo mentre l'output é il risultato della somma. Per il moltiplicatore l'algoritmo crittografico corrisponde alla moltiplicazione di due numeri, l'input é uno dei due fattori, la chiave segreta corrisponde all'altro fattore mentre l'output é il risultato della moltiplicazione. Nel circuito crittografico della sbox algoritmo e output corrispondono esattamente a quelli del circuito della sbox. Dato che la sbox presenta un solo ingresso, l'input del circuito crittografico e la chiave segreta sono messi in `xor` prima di entrare nel circuito della sbox. I tre circuiti crittografici così ottenuti sono rappresentati in Figura 4.2.

Per eseguire gli attacchi CPA é stata utilizzato il progetto opensource OpenSCA, disponibile qui [3]. OpenSCA é una toolbox di Matlab che fornisce gli script e le primitive base per eseguire gli attacchi DPA e permette di ottenere dei report sugli attacchi sia testuali sia visuali tramite la creazione di grafici per mettere in evidenza le chiavi segrete trovate.

Dopo una prima fase di pulizia dei file delle tracce, sono stati adattati gli script e le funzioni per in modo da eseguire l'attacco CPA, il quale é riuscito a restituire i valori della chiave segreta in circa un minuto. I risultati degli attacchi verranno mostrati nelle prossime sezioni.

4.2 Analisi delle prestazioni

In questa sezione verranno mostrate le informazioni relative ai circuiti utilizzati per le simulazioni.

I circuiti presentati nel Capitolo 3 presentano la peculiarità di essere puramente combinatori, ovvero sono dei circuiti il cui funzionamento dipende esclusivamente dalla funzione che mappa l'ingresso con la rispettiva uscita. Non presentano infatti segnali esterni di sincronizzazione o elementi di memoria come nei circuiti sequenziali.

Le grandezze di interesse per questi tipi di circuiti sono l'area e il cammino critico. L'area é una grandezza fisica legata allo spazio che occupano tutte le celle di un circuito una volta sintetizzata la netlist. Il cammino critico é una grandezza fisica legata al tempo di elaborazione del circuito ed é rappre-

sentata dal percorso ingresso-uscita che presenta il ritardo di propagazione maggiore.

I software di sintesi permettono di eseguire ottimizzazioni durante il processo di generazione della netlist. A seconda della configurazione impostata si può per esempio cercare di ottimizzare l'area occupata dal circuito oppure cercare di ridurre il cammino critico in modo da avere frequenze di lavoro più alte. In questo elaborato é stata impostata l'ottimizzazione in modo da avere l'area piu contenuta possibile.

Al fine di ottenere questi valori sono state ricavate le netlist dei circuiti per le versioni non protette (che per comodità chiameremo ad 1 share) e per le versioni protette da 2 a 8 share.

4.2.1 Analisi area circuiti

Qui di seguito sono mostrati le tabelle contenenti le informazioni sulle aree ottenute in seguito alla sintesi delle netlist. I valori rilevati sono il numero di componenti di libreria utilizzati, il numero di connessioni di questi ultimi, il numero di celle occupate e l'area totale occupata dal circuito espressa in numero gate equivalenti (GE).

Esprimere l'area occupata in GE permette di esprimere l'area in una grandezza indipendente dalla libreria di sintesi utilizzata, infatti il numero di gate di libreria utilizzato rimarrà sempre lo stesso e una volta impostata la libreria di sintesi si potrà calcolare l'area effettivamente occupata in nm^2 o nei suoi sottomultipli.

Tabella 4.1: Analisi area sommatore a 32 bit

Share	**Componenti**	**Connessioni**	**Celle**	**Area[GE]**
1	129	130	35	284.39
2	226	1140	1011	2623.85
3	323	2370	2175	6259.52
4	420	3555	3293	8962.69
5	517	5022	4692	12746.18
6	614	5424	5025	14838.79
7	711	6624	6155	17912.71
8	808	9261	8721	25379.74

Da queste tabelle é possibile notare come l'area occupata cresca linearmente col numero di share utilizzate, infatti aumentare il numero di share

Tabella 4.2: Analisi area moltiplicatore a 16 bit

Share	Componenti	Connessioni	Celle	Area[GE]
1	96	945	704	2957.99
2	160	10087	10022	27186.07
3	224	20261	20162	58358.86
4	288	27887	27753	78094.80
5	352	39229	39059	112687.37
6	416	42676	42469	130840.83
7	180	53171	52926	160256.56
8	544	70525	70241	214573.81

Tabella 4.3: Analisi area sbox

Share	Componenti	Connessioni	Celle	Area[GE]
1	48	199	191	598.35
2	64	584	567	1820.88
3	80	944	917	3033.08
4	96	1286	1248	4230.73
5	112	1876	1826	6059.60
6	128	2227	2164	7143.21
7	144	2575	2498	8251.23
8	160	3649	3557	11573.40

equivale ad aumentare il numero di porte logiche e fili di connessione che vanno ad impattare proprio sul numero di celle occupate. L'incremento maggiore é presente nel passaggio da 1 a 2 share in quanto bisogna tenere in considerazione l'introduzione delle porte logiche protette con relativo bit di share aggiuntivo e l'introduzione del bus di share casuali non presente per 1 share. Dalla seconda share in poi l'area occupata aumenta in maniera più lieve in quanto il nuovo bus di share casuali é già presente ed é necessario solo aumentare e gestire il nuovo numero di share.

4.2.2 Analisi cammino critico circuiti

Il cammino critico di un circuito indica indica il percorso tra input e output che presenta il maggior ritardo di propagazione del segnale.

Tutti i circuiti implementati sono puramente combinatori quindi, conoscendo il cammino critico, é possibile ricavare i limiti teorici di elaborazione a cui si può spingere un determinato circuito.

La Tabella 4.4 indica il cammino critico e la frequenza massima per ogni tipologia di circuito. Il cammino critico t_{crit} é espresso in ns mentre la frequenza f_{max}, calcolata come $f_{max} = 1/t_{crit}$, é espressa in MHz.

Tabella 4.4: Tabella cammini critici e frequenza massime dei circuiti

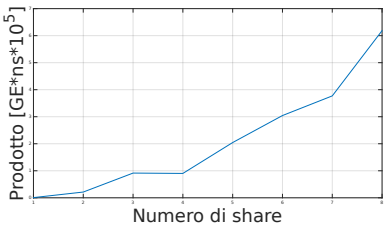
	Sommatore		**Moltiplicatore**		**Sbox**	
Share	$\mathbf{t_{crit}}[ns]$	$\mathbf{f_{max}}[MHz]$	$\mathbf{t_{crit}}[ns]$	$\mathbf{f_{max}}[MHz]$	$\mathbf{t_{crit}}[ns]$	$\mathbf{f_{max}}[MHz]$
1	2.58	387.59	4.57	218.81	1.85	540.54
2	8.11	123.30	11.82	84.60	2.87	348.43
3	14.61	68.44	20.10	49.75	3.08	324.67
4	10.06	99.40	24.45	40.89	3.14	318.47
5	16.04	62.34	22.53	44.38	4.11	243.30
6	20.51	48.75	24.76	40.38	3.75	266.66
7	21.05	47.50	29.13	34.32	3.73	268.09
8	24.37	41.03	29.38	34.03	4.16	240.38

Dai valori ottenuti in tabella si può notare come il cammino critico generalmente cresca al crescere del numero di share, maggiore il numero di share maggiore sarà il numero di porte logiche che il segnale di ingresso dovrà attraversare prima di arrivare all'uscita del circuito. Solo in alcuni casi il cammino critico sembra diminuire ma dipende dal tipo di ottimizzazioni applicate dal tool di generazione della netlist per risparmiare area. L'ottimizzazione impostata per la sintesi dei circuiti é quella che cerca di ridurre al minimo l'area occupata dal circuito.

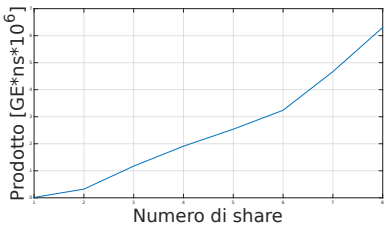
Confrontando le tabelle delle aree occupate e dei cammini critici si può notare come le prestazioni vadano a ridursi all'aumentare del numero di share.

Un'immagine accurata del degrado delle prestazioni é fornita dalla Figura 4.3 in cui é mostrato il prodotto tra area occupata e cammino critico, all'aumentare del numero di share la curva rappresentante il prodotto aumenta sempre di più a conferma del crescere delle due quantità e quindi del degrado delle prestazioni. Nel caso opposto, ovvero in cui le prestazioni migliorano, si sarebbe dovuta visualizzare una curva decrescente.

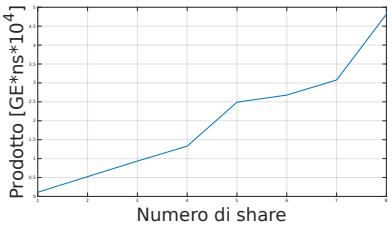
Dalla Figura 4.3(a) si può notare il degrado delle prestazioni del sommatore all'aumentare del numero di share utilizzate. L'area occupata dal sommatore diventa infatti 9 volte più grande già solo introducendo il masking ISW con 2 share e arriva ad essere piu grande di 89 volte con il masking a 8 share. Il cammino critico e la frequenza massima teorica del circuito ri-



(a) Sommatore



(b) Moltiplicatore



(c) Sbox

Figura 4.3: Prodotti area occupata per cammino critico

sentono anch'essi dell'introduzione del masking ISW. Nel sommatore si passa da una riduzione della frequenza massima del 68% introducendo il masking a 2 share ad una riduzione del $89,41\%$ con 8 share.

Il moltiplicatore diventa 9 volte più esteso introducendo il masking a 2 share e arriva a 72 volte con il masking a 8 share. Per quanto riguarda il cammino critico si passa da una riduzione del 61% con 2 share ad una riduzione del 84% con 8 share. La riduzione delle prestazioni é confermata dal prodotto area occupata per cammino critico mostrato in Figura 4.3(b).

La sbox ha un andamento più contenuto rispetto ai precedenti circuiti ma le prestazioni sono comunque decrescenti. L'incremento di spazio é maggiore di 3 volte con l'introduzione del masking a 2 share fino a 19 volte con 8 share. Anche il degrado della frequenza massima é più ridotto rispetto al sommatore e al moltiplicatore, infatti si passa da una riduzione del 35% con 2 share a una riduzione del 55% con 8 share. Anche qui il degrado delle prestazioni é confermato dal prodotto area occupata per cammino critico mostrato in Figura 4.3(c).

Conclusione

In questo capitolo é stata mostrata la metodologia utilizzata per sviluppare i circuiti con e senza masking ISW e sono state analizzate le prestazioni offerte da questi circuiti.

Si può notare in particolar modo come l'introduzione del masking ISW porti ad un degrado notevole delle prestazioni già solo con l'introduzione con due share, l'area occupata aumenta di almeno tre volte e la frequenza massima del circuito cala almeno del 35%.

In definitiva applicare il masking ISW ha un costo legato al degrado delle prestazioni, il circuito occupa più spazio ed é più lento del corrispettivo circuito non protetto, inoltre questo costo aumenta con l'aumentare del numero di share utilizzate.

L'effetto dell'introduzione del masking ISW in termini di protezione offerta verrà analizzato nel Capitolo 5.

Capitolo 5

Analisi Attacchi CPA

In questo capitolo verrà analizzato l'effetto degli attacchi CPA sul sommatore, sul moltiplicatore e sulla sbox nelle versioni con e senza masking ISW.

Tutti gli attacchi sono stati prima eseguiti sulle versioni non protette dei circuiti e poi sulle versioni protette. Per ognuno di questi circuiti verrà prima spiegato come é stato strutturato il testbench e poi quali sono i risultati ottenuti dall'attacco CPA a seconda della protezione utilizzata.

Sebbene siano state generate netlist dei circuiti protetti fino a 8 share é stato però possibile testare solo i circuiti fino a 4 share in quanto i file VCD generati richiedevano troppo tempo di simulazione (più di 20 ore) e iniziavano a assumere dimensioni nell'ordine dei GB. Questo comportamento é dovuto all'elevato numero di porte logiche e fili di collegamenti introdotti dal masking che comportavano un maggior numero di valori da memorizzare nel VCD durante le simulazioni dei testbench.

5.1 Analisi attacchi sommatore

Per applicare correttamente l'attacco CPA al circuito del sommatore é necessario ricavare il circuito crittografico che utilizza come algoritmo di cifratura la somma di due numeri.

L'input di questo circuito crittografico é rappresentato dal primo addendo del sommatore, la chiave segreta é rappresentata dal secondo addendo mentre l'output del circuito crittografico corrisponde all'output del sommatore. Questo circuito é mostrato in Figura 5.1.

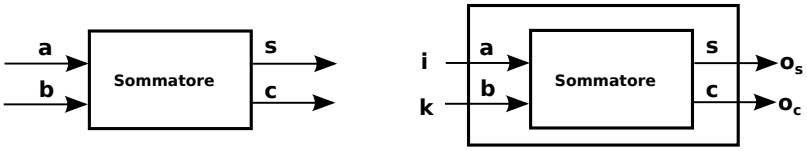


Figura 5.1: Circuito crittografico sommatore

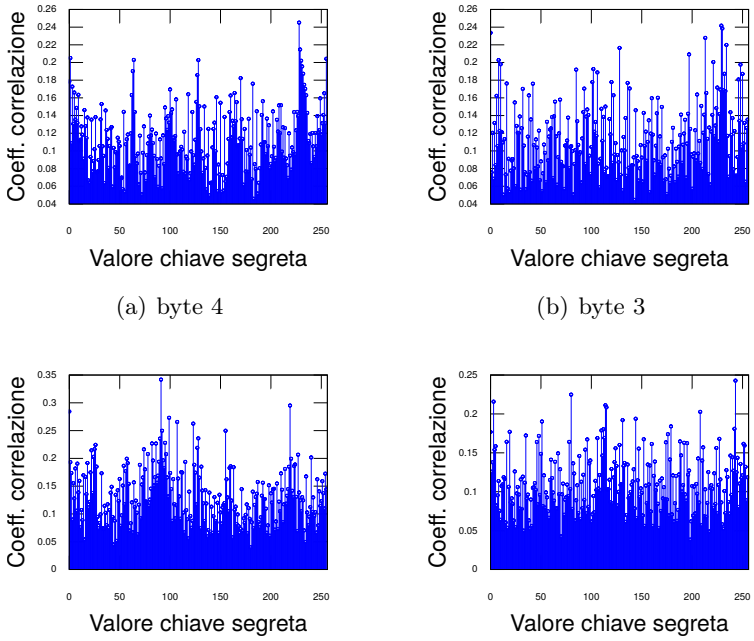
Per applicare l'attacco CPA al circuito così ottenuto é necessario utilizzare un testbench in cui vengono forniti diversi valori di input i lasciando fissa la chiave segreta k. Questo testbench é proprio il testbench utilizzato per certificare il corretto funzionamento del circuito del sommatore, infatti é stato fatto variare per 1000 iterazioni il primo addendo scegliendo i valori in maniera casuale mentre il secondo addendo é stato lasciato fisso per tutte le iterazioni.

Il valore del secondo addendo, che corrisponde alla chiave segreta del circuito crittografico, é stato scelto in maniera casuale e in questo attacco corrisponde al numero espresso in esadecimale F25AE5E4.

Un'ultima decisione fondamentale per poter applicare l'attacco CPA é la scelta della funzione intermedia che verrà poi usata per calcolare il vettore che verrà testato sulle tracce di potenza per ricavare la chiave con correlazione più alta. La funzione intermedia scelta per questo attacco al sommatore é la somma dei byte del testo in chiaro con i byte la chiave segreta.

Per ricavare la chiave segreta si é proceduto per gradi dal byte meno significativo al byte più significativo. In questo modo si riduce lo spazio possibile di chiavi per singolo attacco a solo 256 possibili combinazioni. I risultati degli attacchi al sommatore ad 1 share sono mostrati in Figura 5.2.

Attaccando il byte meno significativo della chiave segreta, che corrisponde al valore esadecimale E4, sono stati ottenuti i valori di correlazione mostrati in Figura 5.2(a). Nell'attacco CPA standard si prende come possibile chiave solo quella con correlazione più alta. Sebbene siano presenti valori di correlazione bassi, infatti sono inferiori a 0,25, si può notare che il valore della chiave che spicca rispetto a tutti gli altri corrisponde al valore decimale 228, che corrisponde proprio al valore in esadecimale E4. Si può quindi affermare che per questo byte l'attacco ha avuto successo.



(a) byte 4 (b) byte 3

(c) byte 2 (d) byte 1

Figura 5.2: Attacco CPA al sommatore

Lo scopo dell'attacco CPA é quello di ridurre il numero possibile di chiavi segrete tra tutte quelle possibili ad una sola. Analizzando i picchi della Figura 5.2(a) si possono contare, oltre al picco più alto della chiave 228, anche altri 4 picchi che si discostano dagli altri per valore di correlazione. In totale quindi ci sono molto probabilmente solo 5 possibili valori di chiave da testare rispetto dall'attacco brute force che li prova tutti e 256. Il motivo per cui esistono dei picchi prossimi al valore vero della chiave segreta può essere ricondotto al numero ridotto di test effettuati, infatti sono 1000 su uno spazio di 2^{32} possibili test, oppure al fatto che ci sono valori di chiave tali per cui l' errore sulla stima di consumo non è completo, di conseguenza c'è comunque una correlazione, anche se minore, che fornisce risultati comparabili con quelli della chiave vera.

L'attacco al terzo byte della chiave segreta, che vale in esadecimale E5, ha ottenuto i valori di correlazione mostrati nella Figura 5.2(b). Anche in questo caso i valori di correlazione sono bassi ma quello che spicca di più rispetto a tutti gli altri é quello con valore di correlazione 0,241 della chiave

decimale 229, che corrisponde proprio all'esadecimale E5. Anche per questo byte si può affermare che l'attacco abbia avuto successo nel ricavare la chiave segreta.

Per considerare l'attacco valido é però necessaria una precisazione. Il tipo di circuito in esame é un sommatore in cui vengono sommati i bit delle due parole di input che generano i bit di riporto intermedi. Attaccando un byte alla volta si rende indipendente un byte da quello precedente perciò si ottiene l'effetto di perdere il valore del bit di riporto. É perciò necessario considerare i risultati ottenuti a meno del bit meno significativo per i byte successivi al primo e quindi si può tollerare un errore solo sul bit meno significativo per poter affermare il successo dell'attacco.

É necessario quindi considerare come byte di chiave sia il valore E5, 11100101 in binario, sia il valore E4, 11100100 in binario.

Si procede al byte successivo con la stessa metodologia applicata per i byte precedenti. Per questo attacco la chiave segreta da ricercare corrisponde al valore esadecimale 5A e i risultati sono mostrati in Figura 5.2(c). In questo attacco il valore di chiave che possiede correlazione più alta é la chiave con il valore decimale pari a 90, che corrisponde al valore esadecimale 5B. É necessario prendere il byte 5B a meno del bit meno significativo per i motivi già spiegati nell'attacco al byte precedente. I due valori da considerare sono quindi 5B e 5A che corrispondono rispettivamente a 01011011 e 01011010. L'attacco riesce quindi a trovare il valore della chiave segreta tra i due possibili valori da considerare e perciò si può affermare che abbia avuto successo.

Si procede infine all'attacco al primo byte a cui corrisponde la chiave segreta in esadecimale F2. I valori di correlazione ottenuti sono mostrati nella Figura 5.2(d). Il picco più alto é presente con correlazione 0,242 nel valore di chiave decimale 243, che corrisponde al valore esadecimale F3. Anche qui é necessario prendere il valore a meno del bit meno significativo e quindi bisogna considerare F3 e F2 che corrispondono rispettivamente ai valori binari 11110011 e 11110010. Anche in questo caso si può considerare l'attacco un successo perché il valore vero ricade tra i due valori possibili.

Tutta la metodologia appena mostrata mostra come l'attacco CPA sia

valido e abbia avuto successo nel ricavare la chiave segreta o nel restringere il campo di chiavi possibili rispetto ad un attacco brute force. In un attacco reale infatti non si conosce la chiave segreta vera e si procede a testare quella ricavata dall'attacco CPA.

Nel caso si verificasse una situazione come quella del sommatore in cui ad ogni byte possono corrispondere più valori bisogna elencare tutte le possibili chiavi risultanti e testarle tutte.

In questa situazione si sarebbe ottenuta la seguente lista di chiavi in cui é evidenziata quella corretta: F25AE4E4, **F25AE5E4**, F25BE4E4, F25BE5E4, F35AE4E4, F35AE5E4, F35BE4E4, F35BE5E4.

Questo attacco CPA applicato al sommatore restringe il numero di possibili chiavi da provare da 2^{32} ad uno spazio di sole 8 chiavi e ne conferma quindi la validità e la potenza del suo utilizzo.

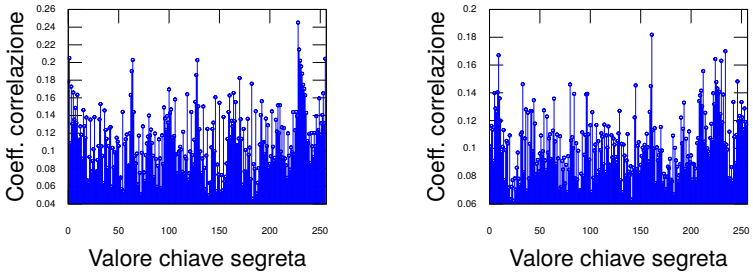
Analisi attacchi sommatore protetto

Verranno ora analizzati i risultati dello stesso attacco applicato al sommatore protetto utilizzando le versioni con masking ISW da due a quattro share. Verrà applicata la stessa metodologia, ovvero verrà attaccato prima il byte meno significativo per poi passare ai successivi tenendo nota del problema del riporto già spiegato in precedenza.

Ripetendo l'attacco partendo dal quarto byte si può notare come l'attacco fallisca per i circuiti protetti con due, tre e quattro share. Nella Figura 5.3 sono mostrati i valori di correlazione ottenuti per cercare di ricavare la chiave segreta E4, che in decimale vale 228.

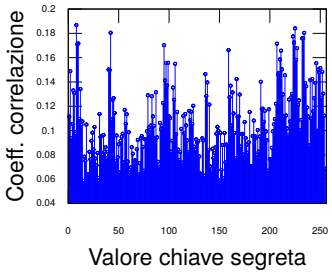
Nel caso del circuito a due share, mostrato in Figura 5.3(b), il valore di correlazione trovato vale 0,181 ed appartiene alla chiave in decimale 161, che in decimale vale A1. In questa situazione si può notare come si sbagli molto più di un bit e come i valori di correlazione siano molto più bassi rispetto ai risultati ottenuti con l'attacco al circuito non protetto.

Nell'attacco al circuito con tre share in Figura 5.3(c) la situazione non cambia di molto, anche qui il valore massimo di correlazione più alto vale 0,186 ed appartiene alla chiave decimale 8, il cui valore é identico in esadecimale. I valori di correlazione sono ancora più bassi di quelli ottenuti con l'attacco al circuito non protetto e vengono sbagliati molti bit.

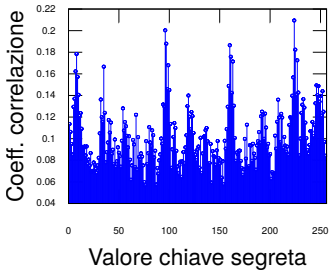


(a) 1 share

(b) 2 share



(c) 3 share



(d) 4 share

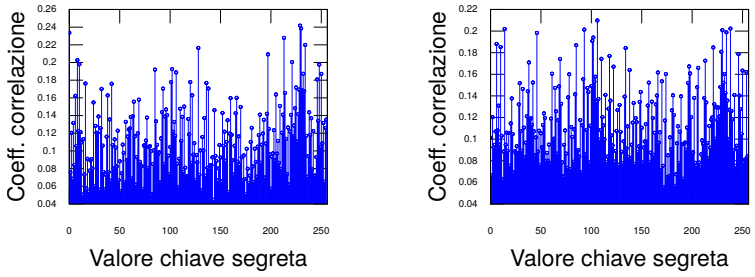
Figura 5.3: Attacco CPA al sommatore protetto - quarto byte

Nel risultato ottenuto nella figura in Figura 5.3(d) con l'attacco al circuito con quattro share si può notare come la correlazione maggiore appartenga alla chiave decimale 224, E0 in esadecimale, con un valore di 0,209. Viene sbagliato solo un bit ma non é quello meno significativo, inoltre anche qui i valori di correlazione ottenuti sono mediamente più bassi.

Per ciascuno dei tre circuiti protetti mostrati si può affermare che la protezione del masking ISW a due, tre e quattro share ha funzionato impedendo di risalire al quarto byte della chiave segreta.

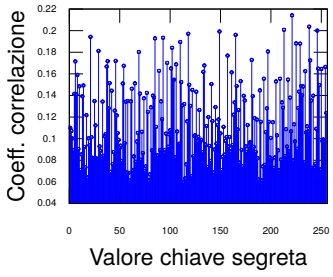
Il prossimo byte della chiave segreta da attaccare é il terzo, il cui valore vero in esadecimale é E5 che vale 229 in decimale. Nella Figura 5.4 sono mostrati i risultati dell'attacco.

L'attacco al terzo bit del circuito con due share mostrato in Figura 5.4(b) restituisce come risultato la chiave segreta 106, 6A in esadecimale, con correlazione massima 0,209. L'attacco è fallimentare in quanto vengono sbagliati tanti bit rispetto alla chiave vera che vale E5.

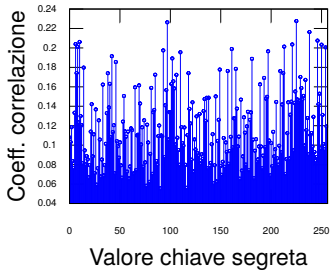


(a) 1 share

(b) 2 share



(c) 3 share



(d) 4 share

Figura 5.4: Attacco CPA al sommatore protetto - terzo byte

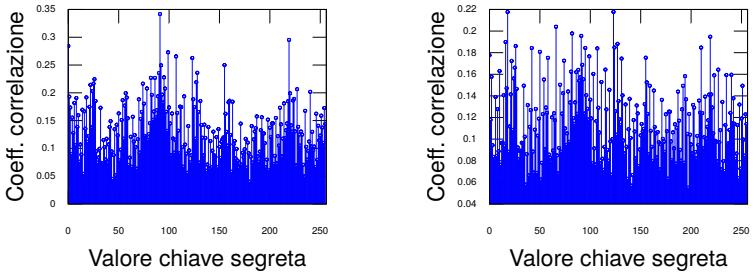
L'attacco mostrato in Figura 5.4(c) al circuito protetto con il masking a tre share restituisce come risultato la chiave segreta 221, DD in esadecimale, con correlazione massima 0,214. L'attacco è anch'esso fallimentare in quanto vengono sbagliati tanti bit rispetto alla chiave vera che vale E5.

Nell'attacco al circuito con quattro share mostrato in Figura 5.4(d) si ottiene la chiave segreta decimale 225, che vale in esadecimale E1. Il suo valore di correlazione 0,227 é piu basso rispetto a quello dell'attacco al circuito non protetto e sbaglia di un solo bit che purtroppo non é quello di riporto. L'attacco non può essere considerato un successo.

Anche per l'attacco al terzo byte la protezione offerta dal masking ISW é valida e impedisce l'attacco CPA.

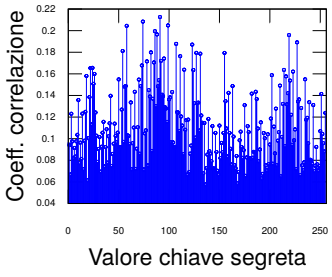
Si procede all'attacco CPA applicato al byte di chiave segreta successivo, ovvero il secondo, il cui valore vero é 5A in esadecimale o 90 in decimale. I risultati dell'attacco sono mostrati in Figura 5.5.

Nella Figura 5.5(b) é mostrato l'attacco CPA applicato al secondo byte

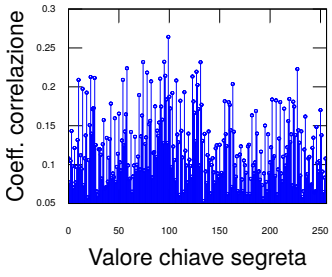


(a) 1 share

(b) 2 share



(c) 3 share



(d) 4 share

Figura 5.5: Attacco CPA al sommatore protetto - secondo byte

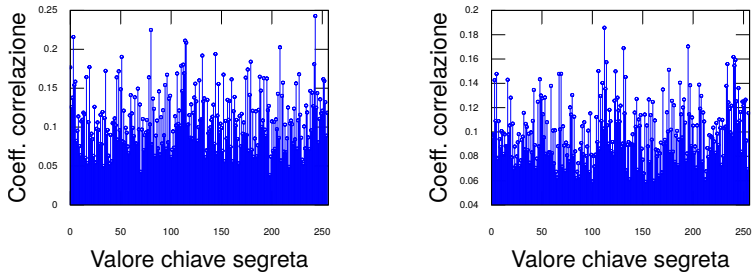
della chiave segreta applicato al circuito protetto col masking a due share. L'attacco restituisce la chiave con valore decimale 123, che corrisponde all'esadecimale 7B, con valore di correlazione pari a 0,217. L'attacco non ha avuto successo perchè sbaglia molti bit e in media restituisce valori di correlazione minori dell'attacco al circuito non protetto.

Nella Figura 5.5(c) viene mostrato l'attacco al circuito protetto con tre share. Il valore di correlazione maggiore viene assegnato alla chiave 235, che corrisponde all'esadecimale 5B. In questo caso l'attacco ha successo perché sbaglia di un bit, proprio nella posizione di riporto. Questa situazione in cui il circuito é protetto e l'attacco ha successo é da considerare un caso isolato, infatti sarà l'unica occorrenza verificata su tutti gli attacchi ai circuiti protetti.

L'attacco al circuito protetto con quattro share é mostrato in Figura 5.5(d). L'attacco restituisce la chiave in decimale 99 che corrisponde all'esadecimale 63. Il valore della chiave segreta ottenuta é diverso da quella vera per molti bit e quindi l'attacco non ha avuto successo.

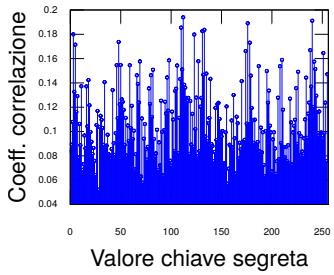
Analizzando gli attacchi al secondo byte si può affermare che anche qui la protezione offerta dal masking ISW riesce a proteggere dagli attacchi CPA.

Gli ultimi attacchi da applicare al sommatore sono quelli riguardanti il byte più significativo della chiave segreta che vale in esadecimale F2. I risultati dell'attacco sono mostrati in Figura 5.6.

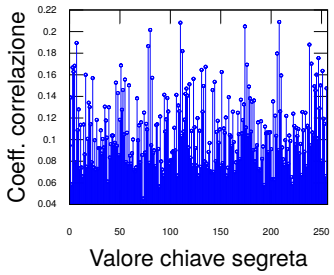


(a) 1 share

(b) 2 share



(c) 3 share



(d) 4 share

Figura 5.6: Attacco CPA al sommatore protetto - primo byte

L'attacco al circuito protetto con due share, mostrato in Figura 5.6(b), risulta fallimentare. Associa la correlazione più alta alla chiave decimale 112 che corrisponde in esadecimale al valore 70. Anche qui vengono sbagliati tanti bit quindi l'attacco non ha successo.

L'attacco al circuito protetto con tre share, mostrato in Figura 5.6(c), assegna la correlazione più alta alla chiave decimale 112 che corrisponde in esadecimale al valore 70. Anche qui vengono sbagliati tanti bit quindi l'attacco é da considerare fallimentare.

L'ultimo attacco da effettuare, mostrato in Figura 5.6(d), assegna la correlazione più alta alla chiave decimale 208, che corrisponde al valore esa-

decimale D0. Vengono anche qui sbagliati molti bit e l'attacco non ha quindi successo nemmeno qui.

Gli attacchi al primo byte non hanno avuto successo nemmeno nelle versioni protette del sommatore con due, tre e quattro share.

Le analisi dei risultati mostrati nei paragrafi precedenti mostrano come gli attacchi CPA falliscano in presenza di circuiti protetti col masking ISW. Non solo le chiavi ottenute sono completamente sbagliate ma si può notare come i valori di correlazione ottenuti vengano mediamente abbassati quando si passa dalla versione non protetta a quella protetta con masking.

5.2 Analisi attacchi moltiplicatore

In questa sezione verranno affrontate le problematiche riscontrate nell'attacco CPA applicato al moltiplicatore.

Riprendendo la metodologia applicata per il sommatore, il primo step necessario é quello di ricavare il circuito crittografico equivalente. Questo circuito considera uno dei due fattori come input i del circuito crittografico mentre l'altro fattore verrà considerato come chiave segreta k. Questo circuito crittografico é mostrato in Figura 5.7.

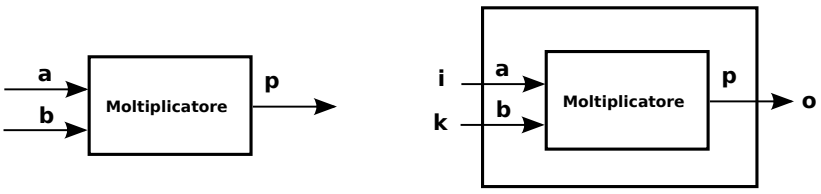


Figura 5.7: Circuito crittografico moltiplicatore

Il testbench utilizzato per gli attacchi prevede 1000 iterazione del calcolo della moltiplicazione dei due ingressi. Viene fatto variare l'input i in maniera casuale mentre viene lasciato fisso il valore di k, a cui é stato assegnato in maniera casuale in valore esadecimale F0E6.

Il problema riscontrato nell'attacco al moltiplicatore é il problema fondamentale degli attacchi CPA, ovvero quello di scegliere una funzione intermedia adatta per eseguire l'attacco CPA. Sono state provate diverse funzioni

per cercare di associare il consumo di potenza in un determinato istante ad un valore dipendente dalla chiave ma l'esito é stato sempre negativo.

Qualsiasi attacco effettuato utilizzando queste funzioni intermedie applicate al circuito non protetto non é stato in grado di ricavare nessun byte di chiave segreta. É stato anche provato l'attacco per ricavare interamente la chiave segreta di 16 bit ma anche in questo caso i risultati sono stati inconcludenti.

Questo comportamento può essere giustificato dalla struttura del circuito del moltiplicatore in cui non esiste un istante temporale per cui in cui il consumo in potenza è ben modellato dai metodi di predizione appena citati. Il circuito del moltiplicatore, mostrato in Figura 3.6, presenta una matrice di `and` sovrapposta ai full adder che eseguono le somme parziali delle righe della moltiplicazione, verranno quindi eseguite contemporaneamente due operazioni, l'`and` e le somme dei bit degli input, che si sovrappongo a vicenda nel consumare potenza elettrica e non permettono di ottenere delle tracce di potenza chiare e adatte per l'attacco CPA.

Questa situazione é la prova che gli attacchi CPA, sebbene siano potenzialmente in grado di ricavare la chiave segreta indipendentemente dal tipo di circuito esaminato, possono fallire nel caso in cui il modello di predizione del consumo non sia sufficientemente vicino all'effettivo comportamento del circuito.

5.3 Analisi attacchi sbox

In questa sezione verranno mostrati i risultati ottenuti con l'applicazione dell'attacco CPA al circuito della sbox con e senza masking ISW.

Per prima cosa é necessario ricavare il circuito crittografico da attaccare. Il circuito della sbox non presenta un input dedicato per la chiave segreta k, infatti presenta un solo byte di ingresso e presenta un solo byte in uscita.

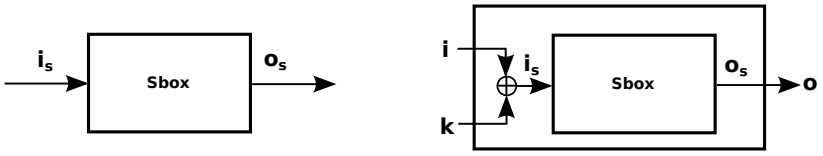


Figura 5.8: Circuito crittografico sbox

Risulta quindi necessario trasformare questo circuito in un circuito adatto per l'attacco CPA. La chiave segreta k verrà quindi messa in `xor` con l'input i del circuito, così come mostrato nella Figura 5.8.

Il testbench utilizzato per il test funzionale della sbox verrà riutilizzato per eseguire l'attacco CPA. Vengono quindi testati esaustivamente tutti i possibili 256 input. La chiave segreta é stata scelta in modo casuale ed equivale al valore esadecimale 37.

La funzione intermedia scelta per questo attacco é lo XOR dell'input del circuito con la chiave segreta.

Per questo circuito non verrà effettuato l'attacco per ricavare una porzione della chiave ma verrà ricavata immediatamente tutta la chiave segreta in quanto composta da un solo byte.

La Figura 5.9 mostra i risultati dell'attacco CPA effettuati per la sbox non protetta.

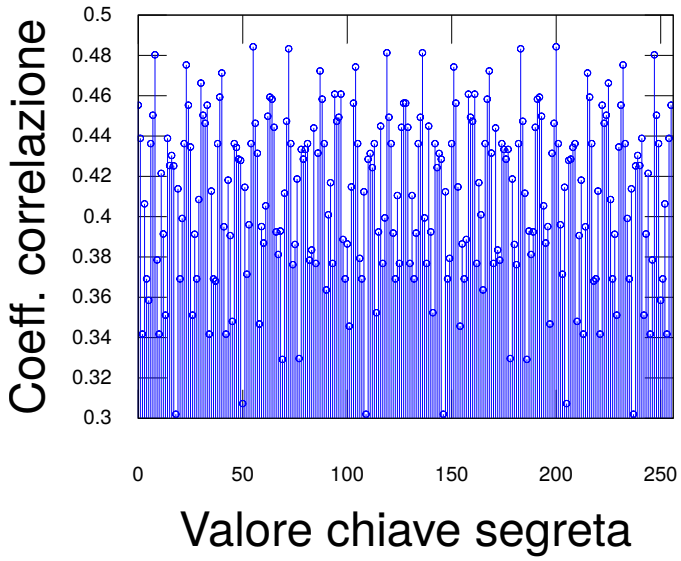


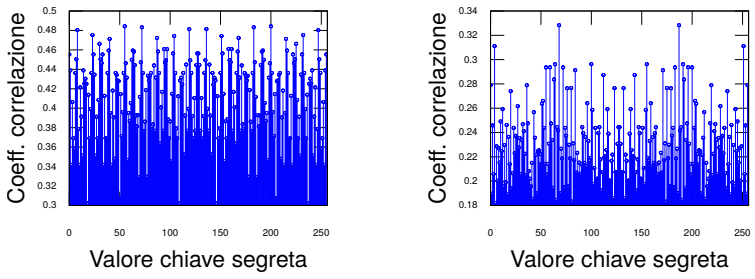
Figura 5.9: Attacco CPA alla sbox

L'attacco CPA mostrato in Figura 5.9 associa la correlazione maggiore alla chiave decimale 55, che in esadecimale equivale a 37, con un valore di 0,484. L'attacco ha avuto successo perché questa chiave ottenuta é proprio

quella corretta sebbene ci siano anche altri picchi di correlazione che risultano sensibilmente più bassi analizzando i valori ottenuti.

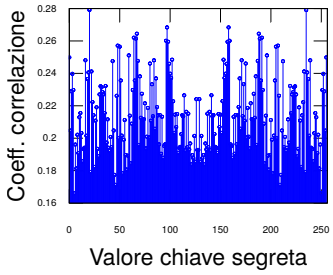
Analisi attacchi sbox protetta

In questa sezione verranno mostrati i risultati ottenuti applicando il masking ISW al circuito della sbox utilizzando due, tre e quattro share come mostrato in Figura 5.10.

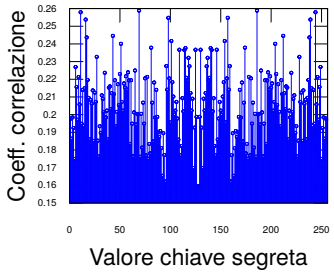


(a) 1 share

(b) 2 share



(c) 3 share



(d) 4 share

Figura 5.10: Attacco CPA alla sbox protetta

La Figura 5.10(b) mostra i risultati dell'attacco al circuito a due share. La correlazione maggiore possiede un valore di 0,328 ed appartiene alla chiave decimale 68, che equivale in esadecimale a 44. I risultati mostrano come la chiave ottenuta sia non corretta e quindi l'attacco si può considerare fallito.

La Figura 5.10(c) rappresenta i risultati dell'attacco CPA al circuito protetto con tre share. Anche in questo caso l'attacco fallisce, infatti la correlazione maggiore di 0,279 é stata assegnata alla chiave decimale 14, 20 in esadecimale, che si discosta di tanti bit dalla chiave vera.

L'ultimo attacco analizzato é quello della sbox protetta con quattro share. I risultati sono mostrati in Figura 5.10(d) e assegnano la correlazione più alta alla chiave decimale 45, che corrisponde in esadecimale a 69. L'attacco al circuito protetto ha quindi fallito.

Conclusione

In questo capitolo sono stati analizzati tutti gli attacchi CPA applicati ai circuiti introdotti nei precedenti capitoli ed é stato possibile valutare la protezione offerta dal masking ISW.

Sono state incontrate tutte le problematiche tipiche dell'applicazione degli attacchi CPA. Si sono infatti verificati casi di successo degli attacchi, come avvenuto per il sommatore e la sbox non protetti, ma anche casi di insuccesso nell'applicare gli attacchi come avvenuto per il moltiplicatore o per tutti i circuiti protetti col masking ISW.

É stato possibile valutare positivamente la protezione offerta dal masking ISW nel contrastare gli attacchi CPA in quanto nessun attacco é riuscito a ricavare la chiave segreta da un circuito protetto. Questa protezione possiede il degrado delle prestazioni mostrato nel Capitolo 4 ma é risultata valida per qualsiasi numero di share.

Capitolo 6

Conclusioni

L'obiettivo di questo elaborato era valutare l'impatto dell'introduzione del masking ISW come protezione contro gli attacchi crittografici di tipo CPA.

Nel Capitolo 2 sono stati mostrati i principi degli attacchi crittografici SCA con particolare riferimento alla tecnica CPA utilizzata per attaccare i circuiti crittografici implementati. É stata inoltre approfondita la tecnica del masking ISW mostrando nel dettaglio come avviene il mascheramento a livello di singola porta logica.

Nel Capitolo 3 sono stati mostrati i circuiti crittografici implementati entrando nel dettaglio delle singole architetture. Sono stati mostrati il sommatore binario a propagazione di riporto, il moltiplicatore binario puramente combinatorio e la sbox dell'AES. La struttura dei singoli circuiti é necessaria per comprendere il successo o il fallimento degli attacchi CPA come avvenuto per il moltiplicatore dove la contemporaneità di due operazioni vanno a ridurre la correlazione tra il consumo di potenza e la chiave.

Nel Capitolo 4 sono state mostrate le prestazioni ottenute dai circuiti con e senza protezione dagli attacchi CPA. É stato possibile notare come l'introduzione del masking ISW provochi un degrado delle prestazioni in quanto aumenta l'area occupata e diminuisce la frequenza massima teorica di funzionamento del circuito.

Nel Capitolo 5 sono stati analizzati i risultati degli attacchi applicati a tutti i tipi di circuiti con e senza protezione. Per quanto riguarda i circuiti senza protezione si sono verificati casi di successo degli attacchi per il som-

matore e la sbox e di insuccesso per il moltiplicatore. Per quanto riguarda gli attacchi ai circuiti protetti non si sono mai verificati casi di successo a conferma della validità della protezione.

In definitiva la minaccia degli attacchi SCA esiste ed é stato mostrato come possa essere sviluppato un attacco di tipo CPA a partire dall'analisi delle tracce di consumo di potenza generate dai software di simulazione.

Una volta verificato come questo tipo di attacco possa funzionare é stato mostrato come la protezione offerta dal masking ISW permetta di modificare l'esito di tale attacchi vanificandoli pur lasciando inalterato il risultato effettivo della computazione, infatti la chiave segreta non é mai stata ricavata nei circuiti con due o più share.

Per quanto riguarda l'aspetto delle prestazioni, il masking ISW presenta però delle controindicazioni sia per quanto riguarda la frequenza massima di lavoro del circuito sia per l'area occupata dal circuito, entrambe infatti subiscono un progressivo degrado al crescere del numero di share. Tuttavia questo peggioramento delle prestazioni é compensato dal successo nel contrastare gli attacchi CPA e pertanto l'introduzione del masking ISW é pienamente giustificata.

Bibliografia

[1] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[2] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Springer, 2002.

[3] Elisabeth Oswald. OpenSCA, An open source toolbox for Matlab. `http://www.cs.bris.ac.uk/home/eoswald/opensca.html`, 2013.

[4] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits II: keeping secrets in tamperable circuits. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 308–327. Springer, 2006.

[5] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[6] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27, 2011.

[7] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.

[8] National Institute of Standards and Technology. FIPS 197: ADVANCED ENCRYPTION STANDARD (AES). `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`, 2001.

[9] North Carolina State University. FreePDK process design kit: an Open-Access-based PDK for the 45nm technology node and the Predictive Technology Model. `http://www.eda.ncsu.edu/wiki/FreePDK`, 2011.

[10] North Carolina State University. NCSU Electronic Design Automation (EDA). `http://www.eda.ncsu.edu/wiki/NCSU_EDA_Wiki`, 2012.

[11] Masoud Rostami, Farinaz Koushanfar, Jeyavijayan Rajendran, and Ramesh Karri. Hardware security: threat models and metrics. In Jörg Henkel, editor, *The IEEE/ACM International Conference on Computer-Aided Design, ICCAD'13, San Jose, CA, USA, November 18-21, 2013*, pages 819–823. IEEE, 2013.

[12] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A compact rijndael hardware architecture with s-box optimization. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2001.

[13] IEEE Design Automation Standards Subcommittee and IEEE Standards Coordinating Committee 20. Automatic Test Program Generation Committee. IEEE standard VHDL : language reference manual. `http://ieeexplore.ieee.org/servlet/opac?punumber=4772738`, 2009.

[14] Xinmiao Zhang and Keshab K. Parhi. High-speed VLSI architectures for the AES algorithm. *IEEE Trans. VLSI Syst.*, 12(9):957–967, 2004.