

Nuovi approcci per garantire la sicurezza nei sistemi hardware.

Analisi delle vulnerabilità: tecniche di rilevazione e mitigazione

New approaches to guarantee security in hardware systems.

Analysis of Vulnerabilities: Detection and Mitigation Techniques

Alessandro Palumbo [◆], Kerem Arıkan [□], Giuseppe Bianchi [◆], Marco Ottavi [◆]

[◆] *Università degli Studi di Roma Tor Vergata*

[□] *TOBB University of Economics and Technology*

Sommario

Il tema della sicurezza dei sistemi di calcolo elettronici, in seguito per brevità chiamati hardware, comprende problemi di sicurezza e fiducia ad ampio raggio, che abbracciano l'intero ciclo di vita di un dispositivo e tutti i suoi livelli di astrazione (chip, PCB, sistemi e sistema di sistemi). Con l'aumento delle vulnerabilità della sicurezza e dei problemi di fiducia, il ruolo dell'hardware come punto riferimento di sicurezza di un sistema informatico è messo in discussione.

A causa della tendenza di affidare le differenti fasi di progettazione e fabbricazione del circuito a diverse strutture e di fare sempre più affidamento su core di proprietà intellettuale (IP) di terze parti, i sistemi su chip (SoC) stanno diventando sempre più vulnerabili ad attività dannose e alterazioni denominate Hardware Trojan. Le modifiche circuitali da loro introdotte possono far trapelare informazioni sensibili (o private) e consentire la fattibilità di attacchi, che hanno lo scopo, ad esempio, di interrompere il funzionamento del sistema e la riduzione della sua affidabilità.

D'altra parte, osservando alcune caratteristiche del circuito, spesso vengono esposte vulnerabilità inaspettate che lanciano nuove sfide a progettisti e ingegneri della sicurezza. Ad esempio, le differenze temporali introdotte dalle cache o dall'esecuzione speculativa di un programma possono essere sfruttate per far trapelare informazioni o rilevare potenziali attività. Un circuito con componenti la cui origine è affidabile (ovvero fidati, in inglese "trusted") potrebbe essere comunque attaccato; così come un circuito a prova di attacco non ha necessariamente componenti che non siano stati manomessi.

I problemi di fiducia (inglese trust) dell'hardware derivano dal coinvolgimento di entità potenzialmente non affidabili nel ciclo di vita di un hardware, inclusi IP non attendibili o fornitori di strumenti CAD (Computer-Aided Design) e strutture di progettazione, fabbricazione, test o distribuzione. I problemi di sicurezza hardware derivano dalla sua stessa

*Nuovi Approcci per Garantire la Sicurezza nei Sistemi Hardware.
Analisi delle Vulnerabilità: Tecniche di Rilevazione e Mitigazione*

A. Palumbo, K. Arıkan, G. Bianchi, M. Ottavi

vulnerabilità agli attacchi (ad esempio, attacchi tipo side-channel o Trojan) a diversi livelli (come chip o PCB), nonché dalla mancanza di un solido supporto hardware per la sicurezza del software e del sistema. Proteggere i microprocessori dalle minacce esistenti non è banale ed è reso ancora più difficile dalla continua comparsa di nuovi attacchi (ad esempio Spectre [1], Meltdown[2]).

Le sfide sono due: rilevare (e possibilmente non innescare) modifiche dannose ai circuiti e proteggere l'hardware dagli attacchi. In questo articolo presentiamo due architetture e un nuovo approccio, basato su strutture dati probabilistiche, per garantire la sicurezza del circuito. Il primo è in grado di rilevare modifiche indesiderate dell'hardware; il secondo protegge il processore dagli attacchi architetturali di tipo side-channel (MSCA). L'idea è di aggiungere ai microprocessori moduli per il controllo della sicurezza, con lo scopo di osservare le istruzioni eseguite, identificare e segnalare possibili attività sospette.

Per valutarne l'efficacia, gli approcci proposti sono stati integrati sul core RISC-V disponibile liberamente per lo sviluppo. Riguardo il trust dell'hardware abbiamo dimostrato che il nostro design è in grado di rilevare se ci sono state manomissioni tra memorie e core; sulla sicurezza dell'hardware abbiamo dimostrato la sua efficacia nel rilevare gli attacchi Spectre [1], Orchestration [31] e Battery Drain [32]. Inoltre esso è configurabile in fase di progettazione (e riconfigurato dopo l'installazione da parte dell'utente) in modo da mantenere sempre aggiornato l'elenco degli attacchi che il checker è in grado di identificare.

Abstract

The topic of hardware security encompasses wide-ranging security and trust issues, which span the entire lifecycle of electronic hardware, and all its abstraction levels (chips, PCBs, systems, and system of systems). With increasing security vulnerabilities and trust issues, the role of hardware as a trust anchor of a computing system is being challenged.

Due to the emerging trend of outsourcing the design and fabrication services to external facilities and increasing reliance on third-party Intellectual Property (IP) cores, Systems on chip (SoCs) are becoming increasingly vulnerable to malicious activities and alterations referred to as Hardware Trojans. The modification introduced by them can leak sensitive (or private) information as well as enable launching other possible attacks, for example, denial of service and reduction in reliability.

On the other hand, observing some features of the circuit, often are exposed unexpected vulnerabilities that pose new challenges to designers and security engineers. For example, the timing differences introduced by caches or speculative execution can be exploited to leak information or detect activity patterns.

A circuit with trustable components could be attacked; an attack-proof circuit does not necessarily have components that have not been tampered with. Hardware trust issues arise from involvement of untrusted entities in the life cycle of a hardware, including untrusted IP or computer-aided design (CAD) tool vendors, and untrusted design, fabrication, test, or distribution facilities. Hardware security issues arise from its own vulnerability to attacks (e.g.,

side-channel or Trojan attacks) at different levels (such as, chip or PCB), as well as from lack of robust hardware support for software and system security. Protecting microprocessors from existing attacks is an extremely and it is made even harder by the continuous rise of new attacks (e.g. Spectre [1], Meltdown [2]).

The challenges are two: detecting (and possibly bypassing) circuit malicious modifications and protecting the hardware from attacks. In this paper we present two architectures and a new approach, based on probabilistic data structures in order guarantee the safety of the circuit. The first one is able to detect undesired modification of the hardware; the second one protects microprocessor against Microarchitectural Side-Channel Attacks (MSCA). The idea is to add to microprocessor-based systems a security checking modules in charge of observing the fetched instructions and of identifying and signaling possible suspicious activity.

We integrated the proposed approaches in a RISC-V core. About the trustable of the hardware we proved that our design is able to detect design-time by the designer (and reconfigured after deployment by the user) in order to always keep updated the list of the attacks the checker is able to identify. if there have been any modifications between memories and core; about the hardware security we showed its effectiveness in detecting the Spectre [1], Orchestration [31], and Battery Drain [32] attacks. In addition it is configurable at design-time by the designer (and reconfigured after deployment by the user) in order to always keep updated the list of the attacks the checker is able to identify.

1. Introduzione

La sicurezza informatica è diventata una parte essenziale del mondo elettronico moderno. I requisiti di un sistema, affinché sia sicuro, sono stringenti e di vitale importanza in ogni settore, come ad esempio nei settori di *Internet of Things (IoT)/Edge Computing* [3], *Industria 4.0* [4] o *Automotive* [5].

La sicurezza delle informazioni, in generale, fornisce riservatezza, integrità e disponibilità dei dati mediante la protezione dai processi (accesso, uso, modifica, distruzione) non “autorizzati”. La sicurezza della rete si concentra sugli attacchi alle informazioni condivise dai dispositivi che sono connessi alla medesima rete e sui meccanismi per garantire l’usabilità e l’integrità dei dati, designati da potenziali attacchi. La sicurezza del software si concentra sugli attacchi “dannosi” all’esecuzione del programma, spesso sfruttando bug di implementazione, (come ad esempio la gestione degli errori incoerente e gli *overflow*) e sulle tecniche adibite a garantire un funzionamento affidabile dell’algoritmo eseguito dal programma stesso, anche in presenza di potenziali rischi per la sicurezza.

Per garantire l’esecuzione sicura di un qualsiasi software, è necessario espletarla su un circuito dal processo produttivo affidabile. L’*Hardware Security* è la scienza che si occupa della sicurezza dell’hardware, andando a proteggere i vari componenti elettronici che detengono dati sensibili e privati. Costituisce la base della sicurezza del sistema, fornendo un punto di ancoraggio alla fiducia per altre componenti di altro sistema che interagiscono strettamente con esso. Comprende la definizione dell’architettura del circuito, l’implementazione e la successiva

convalida. Il focus della sicurezza dell'hardware si concentra sull'annientare attacchi predisposti per boicottare o compromettere risorse e sul progettare approcci per proteggere queste ultime. Le risorse in esame sono i componenti dello stesso hardware, ad esempio circuiti integrati (di tutti i tipi), componenti passivi (come resistori, condensatori, induttori) e PCB; così come i dati memorizzati all'interno di questi componenti (ad esempio chiavi crittografiche, dati utente sensibili, firmware, dati di configurazione).

Le proprietà di sicurezza tradizionali (riservatezza, integrità e confidenzialità dei dati), sono generalmente garantite dagli algoritmi crittografici (ad esempio AES [6] e RSA [7]) e dalle funzioni di *hashing* (ad esempio SHA 3 [8]). Matematicamente tali algoritmi risultano robusti, ma le loro possibili implementazioni possono soffrire di falle di sicurezza. Negli ultimi anni, diversi acceleratori hardware crittografici hanno dimostrato di essere predisposti a numerosi attacchi, tra cui *Differential Fault Analysis* (DFA) [9] e *Side-Channel Analysis* (SCA) [10]. Di conseguenza, i sistemi implementati possono essere vulnerabili, sebbene siano dotati di moduli dedicati alla sicurezza.

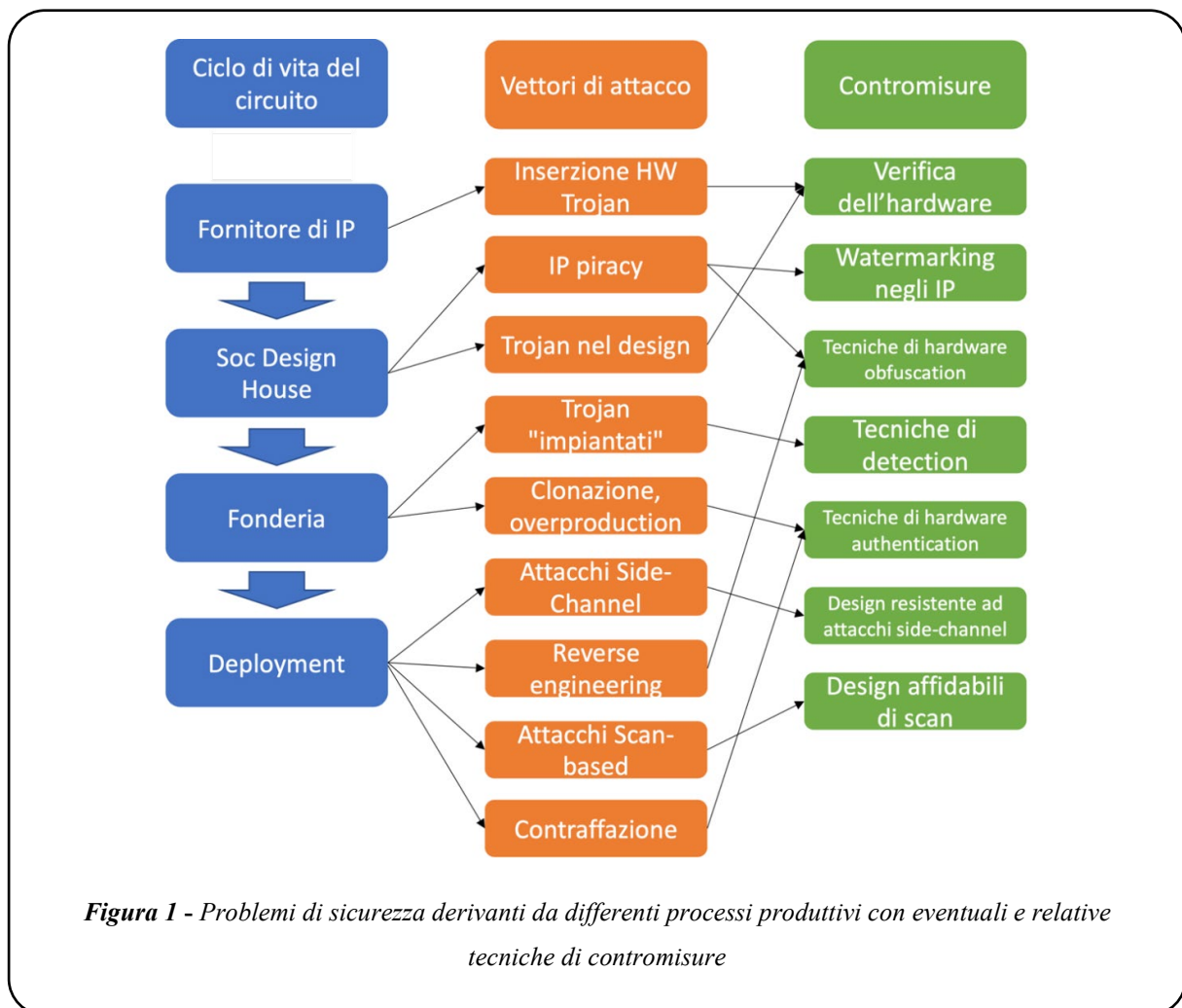


Figura 1 - Problemi di sicurezza derivanti da differenti processi produttivi con eventuali e relative tecniche di contromisure

2. Modelli per la sicurezza

Per descrivere un potenziale problema del circuito soggetto ad un attacco, e di conseguenza pensare ad una possibile soluzione, è fondamentale descrivere in modo univoco il modello di sicurezza corrispondente. A questo proposito vanno specificati, il *modello di trust*, che indica quali parti o componenti del circuito sono affidabili e sicuri e il *modello della minaccia*, che descrive lo scopo e il meccanismo di un attacco. Questo modello descrive l'obiettivo dell'attacco e la modalità tramite la quale viene innescato. Ad esempio, far trapelare dati da un SoC (o boicottare il suo comportamento funzionale), mediante l'inserimento malizioso di un trojan, o sfruttando una vulnerabilità del circuito non prevista dai progettisti.

2.1. Vulnerabilità

Le vulnerabilità si riferiscono alle debolezze dell'architettura hardware, dell'implementazione o dei processi di progettazione e/o di test. In una di queste fasi un operatore malintenzionato può attivare un attacco, o modificare la funzione per la quale il circuito è stato progettato. Questi punti deboli possono essere funzionali o non funzionali e variano in base alla natura di un sistema e ai suoi scenari di utilizzo. Un attacco tipico consiste nell'identificazione di una o più vulnerabilità, seguita dal loro sfruttamento per un attacco riuscito. L'identificazione è la prima delle fasi dell'iter di qualsiasi tipo di attacco.

Più in dettaglio, possiamo differenziare le tipologie di vulnerabilità in:

- **criticità funzionali:** la maggior parte delle vulnerabilità sono causate da pratiche di progettazione e di test inadeguate. Includono un'implementazione hardware crittografica debole e una protezione insufficiente delle risorse in un SoC. Gli aggressori possono rilevare queste vulnerabilità analizzando la funzionalità di un sistema per diverse condizioni di input per cercare eventuali comportamenti anomali;
- **criticità da side-channels:** rappresentano problemi a livello di implementazione. In questo contesto possono trapelare informazioni critiche memorizzate all'interno di un componente hardware attraverso diverse forme di canali laterali [11]. Gli aggressori possono rilevare queste vulnerabilità andando ad analizzare caratteristiche del circuito (anche durante il funzionamento dello stesso), apparentemente non correlate all'elaborazione della funzione che sta eseguendo l'hardware;
- **infrastrutture di test / debug:** in sede di test un operatore malintenzionato potrebbe utilizzare le infrastrutture adibite al debug del circuito in modo inappropriato. In particolare potrebbe estrarre informazioni sensibili dal momento che ha un accesso completo al sistema, concessogli dal momento che si è nel contesto di test e debug;

- **controllo degli accessi o flusso di informazioni:** un sistema potrebbe non distinguere tra utenti autorizzati e non autorizzati. Questa vulnerabilità può consentire a un utente malintenzionato di accedere a risorse e funzionalità private che possono essere sfruttate o utilizzate in modo improprio. Scenario peggiore sarebbe se si andasse a monitorare il flusso di informazioni durante il funzionamento del sistema, per poi decifrare informazioni critiche per la sicurezza (come il flusso di controllo di un programma e l'indirizzo di memoria di una regione protetta dell'hardware).

2.2. Contromisure

Le possibili contromisure possono essere impiegate nelle fasi di progettazione o di test.

Il flusso di manifattura di un SoC è costituito da quattro fasi concettuali: progettazione, pianificazione, sviluppo e produzione. Nelle prime due fasi viene definita l'architettura del circuito e quindi delineate le funzioni che l'hardware dovrà implementare. La fase di sviluppo consiste nella verifica degli obiettivi delineati in sede di progettazione. Se soddisfatti si procede con la fabbricazione dei chip.

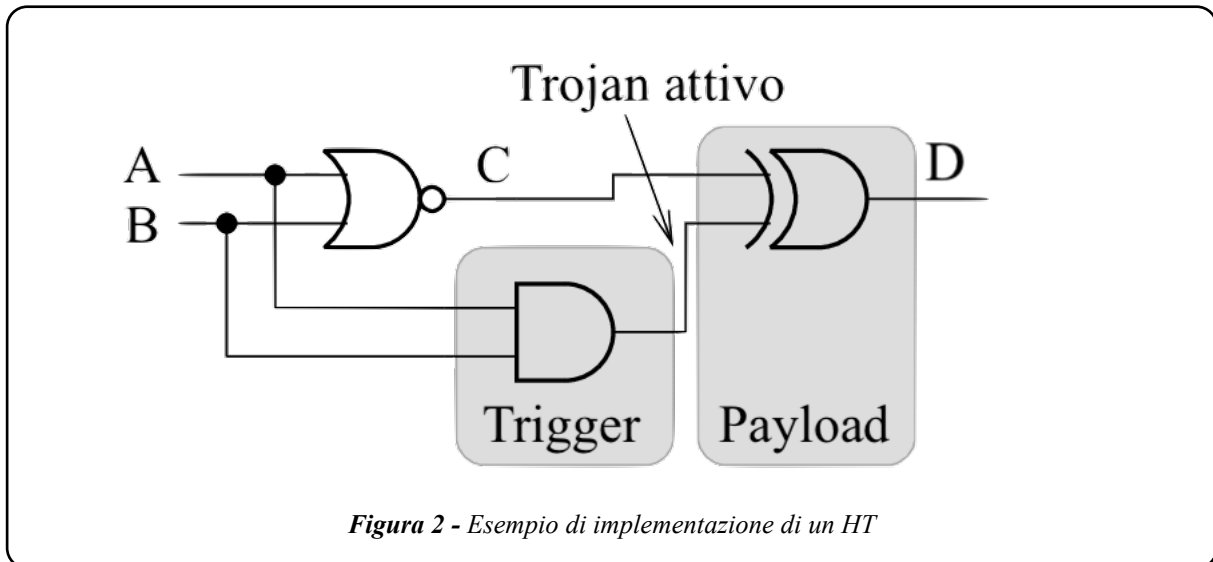
La valutazione della sicurezza viene eseguita durante la fase di pianificazione, che identifica le risorse in un SoC, i possibili attacchi ad esse e i requisiti per l'esecuzione sicura del software. Questo passaggio va a creare una serie di requisiti di sicurezza. Successivamente, viene definita l'architettura ed eseguita la convalida della sicurezza per assicurarsi che l'implementazione soddisfi adeguatamente i requisiti desiderati. Un'analogia convalida della sicurezza viene eseguita dopo la fabbricazione dei chip, per garantire che gli stessi, una volta prodotti, non abbiano vulnerabilità. Queste tecniche includono la revisione del codice, la verifica formale durante la convalida, il *fuzzing* e i test di penetrazione [12].

Le pratiche di progettazione per la sicurezza (*Design for Security*, DfS) sono emerse come potenti contromisure e offrono spesso soluzioni di implementazione a basso *overhead* ed efficaci che forniscono una difesa attiva o passiva contro vari attacchi. Le tecniche DfS, come l'offuscamento [14][15], l'uso di primitive di sicurezza affidabili, la resistenza del canale laterale (ad esempio tecniche di mascheramento e occultamento) e schemi di protezione avanzata contro l'inserimento di trojan, possono proteggere in modo affidabile da molti dei principali vettori di attacco. Allo stesso modo, l'architettura di sicurezza SoC, resistente ad attacchi software, rappresenta la sicurezza del circuito stesso.

3. Hardware Trojans

Un HT viene definito come una modifica intenzionale e dannosa di un progetto di circuito che si traduce in un comportamento indesiderato quando il circuito viene distribuito [17]. I SoC "infettati" potrebbero subire modifiche nella loro funzionalità o specifiche. Ciò porterebbe alla divulgazione di informazioni sensibili dell'utente, o il circuito soffrirebbe di prestazioni degradate o inaffidabili. Il trojan hardware rappresenta una seria minaccia per qualsiasi progetto hardware implementato.

Essendo gli HT modifiche apportate direttamente al circuito, le contromisure software (*antivirus*) potrebbero essere inadeguate per affrontare la minaccia rappresentata dal trojan. Inoltre, il rilevamento di quest'ultimo in un progetto hardware non è banale dal momento che non sempre è disponibile una versione del circuito dove siamo sicuri non vi siano state manomissioni e con cui confrontare un determinato progetto durante la verifica.



La struttura di base di un HT può includere due parti principali, trigger e *payload* [16]. Il primo monitora segnali e/o una serie di eventi nel circuito. Il payload, invece, attinge ai segnali dal circuito originale e dall'uscita del trigger.

Nel momento in cui il trigger rileva un evento o una condizione per cui è stato predisposto l'innesco del trojan, il payload si attiva e viene boicottata l'esecuzione della funzione. Se il trojan è silente (ovvero quando il trigger non innesca il payload) il circuito si comporta come se fosse privo di manomissioni.

Tabella 1. Tabella della verità riferita al circuito in figura 2

A	B	C	D	Payload
0	0	1	1	Non attivo
0	1	0	0	Non attivo
1	0	0	0	Non attivo
1	1	0	1	Attivo

Come notiamo dalla tabella, il payload viene attivato nel momento in cui il valore di entrambi gli ingressi A e B è 1, altrimenti è silente. L'uscita del circuito (D) coincide con il segnale C, tranne quando il payload viene innescato. Un HT, quindi, può essere nascosto durante il normale funzionamento del chip e attivato solo quando viene applicata la condizione di innesco.

A questo proposito il trojan può essere attivato internamente (un evento verificatosi all'interno del circuito, o una particolare condizione circuitale, innesca il payload), o esternamente (una particolare valore di un input o un output di un componente del circuito).

Le possibili conseguenze del funzionamento di un circuito con un trojan attivo possono essere:

- **modifica delle funzionalità originarie:** il payload può modificare la funzionalità del dispositivo di destinazione e causare errori che potrebbero essere difficili da rilevare durante il test di produzione. Ad esempio, un trojan potrebbe far sì che un modulo di rilevamento degli errori accetti input che dovrebbero essere rifiutati;
- **fuga di informazioni:** un trojan può far trapelare dati attraverso canali sia nascosti che palesi. I dati sensibili possono essere diffusi tramite radiofrequenza, potenza ottica o termica, canali laterali di temporizzazione e interfacce I/O. Ad esempio, un trojan potrebbe far fruire all'attaccante la chiave di un algoritmo crittografico attraverso interfacce di uscita del sistema della vittima.

A quest'ultimo proposito, potrebbero essere estrapolati dati sensibili e/o privati, andando ad analizzare canali laterali. Anche se un circuito non è stato manomesso, un utente malintenzionato potrebbe fruire informazioni sui dati elaborati, andando ad osservare caratteristiche apparentemente non correlate all'esecuzione della funzione dell'hardware [18] (potenze dissipate, temperatura del chip, tempi di esecuzione, radiazioni elettromagnetiche).

4. Attacchi di tipo Side-Channel

Gli attacchi di tipo side-channel (SCA) sfruttano le informazioni fisiche che si osservano da fonti o canali indiretti, che apparentemente non dipendono dal funzionamento del circuito. Le informazioni fruite dall'analisi di questi parametri dipendono da valori intermedi, calcolati durante l'esecuzione di un algoritmo sull'hardware, e sono correlate con gli input del circuito stesso [19]. Un attaccante se osservando uno o più parametri "laterali" riuscisse a ricostruire una chiave crittografica segreta, potrebbe decriptare dati sensibili e privati. Per questi motivi, gli SCA rappresentano una grave minaccia per i dispositivi crittografici, in particolare smart card e dispositivi IoT, per i quali un utente malintenzionato potrebbe avere accesso a dati sensibili privati.

Gli attacchi comuni di tipo side-channel, come ad esempio gli attacchi alla potenza, monitorano il consumo di energia del dispositivo. Se consideriamo un dispositivo che sta effettuando una funzione di cifratura, il suo consumo energetico dipende dagli input della funzione crittografica. Per cifrare sequenze di bit differenti vengono dissipati valori di potenze diversi [20].

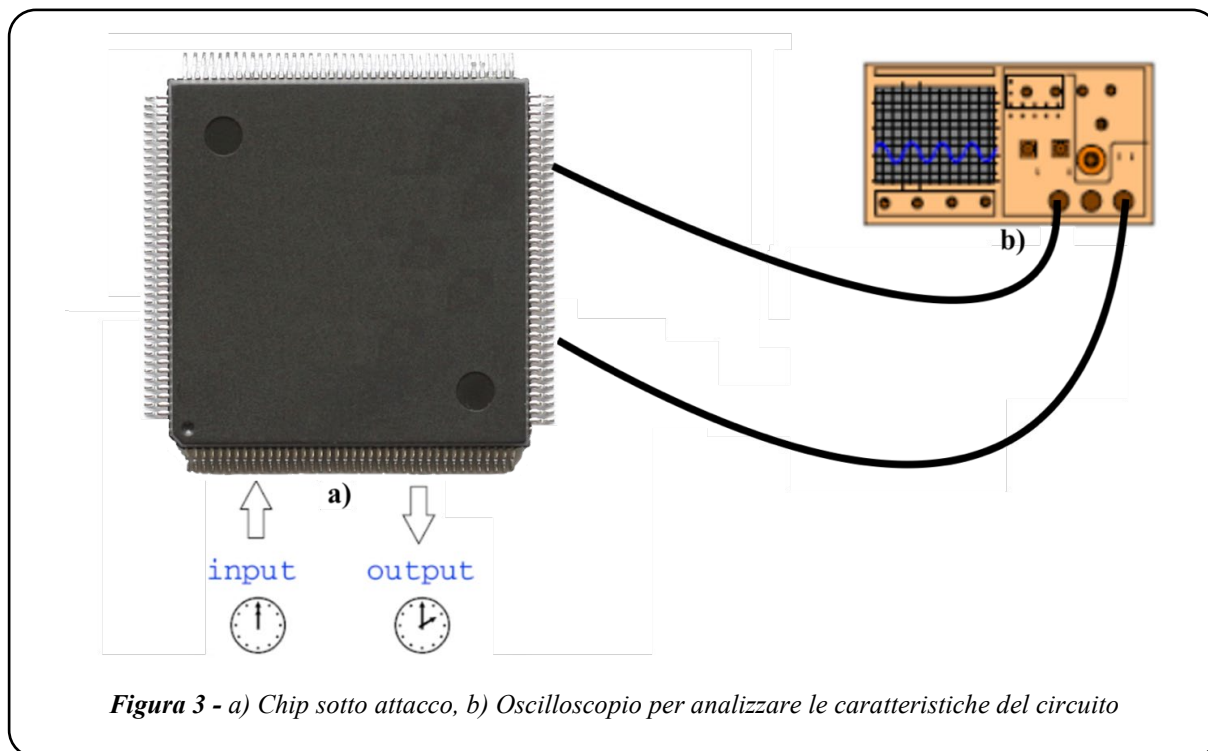


Figura 3 - a) Chip sotto attacco, b) Oscilloscopio per analizzare le caratteristiche del circuito

Un altro tipo di side-channel noto in letteratura è il *timing attack*. Ogni operazione eseguita in un dispositivo elettronico richiede una certa quantità di tempo per essere completata. Questo tempo può variare a seconda del tipo di operazione, dei dati di input, della tecnologia utilizzata per costruire il dispositivo e delle proprietà dell'ambiente in cui il dispositivo sta funzionando. Analizzando il tempo di esecuzione di ciascuna operazione in diverse configurazioni e schemi di input si possono ricavare informazioni cruciali [21][22].

Una possibile contromisura alle vulnerabilità a questo tipo di attacchi è la rimozione delle correlazioni tra gli input del circuito e le emissioni di tipo side-channel. Al variare dei dati in input al sistema possiamo pensare di mantenere costante sia la potenza dissipata dal chip [15], che i tempi di esecuzione delle istruzioni/operazioni [22]. Inoltre, pensando di partizionare il design dell'hardware, si potrebbero individuare due regioni diverse del circuito: una costituita da moduli che elaborano i dati cifrati ed un'altra che opera sui dati in chiaro.

5. Contromisure architetturali

Una nuova tendenza per garantire la sicurezza dell'hardware è l'inserzione nel circuito di moduli che implementano algoritmi probabilistici, servendosi di funzioni di hash. Questi possono essere aggiunti al circuito per rilevare l'eventuale presenza di trojan, o per segnalare attacchi di tipo side-channel [23][24][25]. Lo scopo di questi moduli è semplicemente quello di stimare alcuni parametri. La forza di questi approcci è il loro minimo overhead introdotto in termini di area e tempi per l'elaborazione.

Nei prossimi paragrafi vengono presentate due metodologie circuitali con le quali sarebbe possibile identificare con la prima HT un attacco di tipo *man in the middle* [28] e con la seconda un attacco di tipo side-channel.

Consideriamo, per esempio, il caso in cui un operatore malizioso, durante una delle fasi di produzione del chip, abbia manomesso il circuito. In particolare supponiamo abbia interposto un modulo hardware tra la memoria di un processore ed il core. Questa manomissione va a modificare le corrispondenze tra i dati e i relativi indirizzi di memoria in cui sono stati scritti precedentemente. In sede di scrittura della memoria si avranno determinate corrispondenze tra dati ed indirizzi; nel processo di lettura, con il payload del trojan attivo, la corrispondenza non sarà più la medesima. Per rilevare questo tipo di manomissioni è necessario, in primo luogo osservare a quali indirizzi della memoria vengono scritte le singole stringhe di dati ed in seguito controllare se vi è la medesima corrispondenza quando il core le legge. A questo proposito possono essere sfruttate le architetture basate sull'elaborazione di algoritmi probabilistici.

5.1. Checker di hardware trojan basato su filtro di Bloom

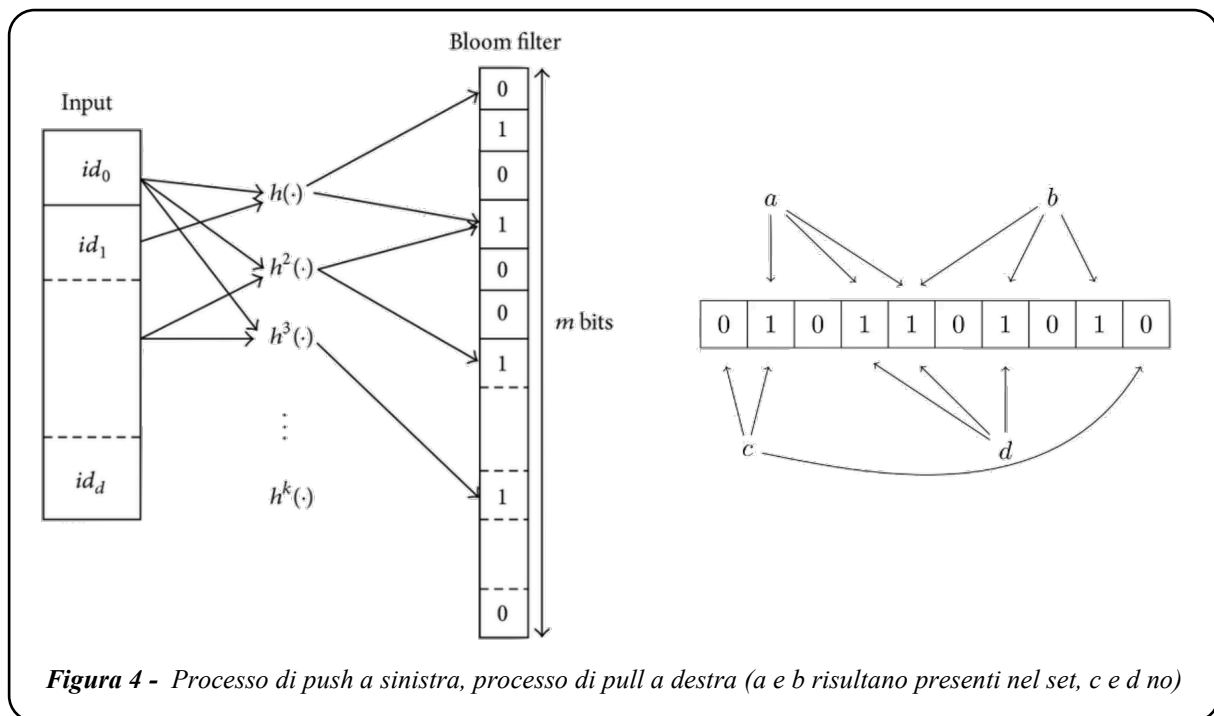


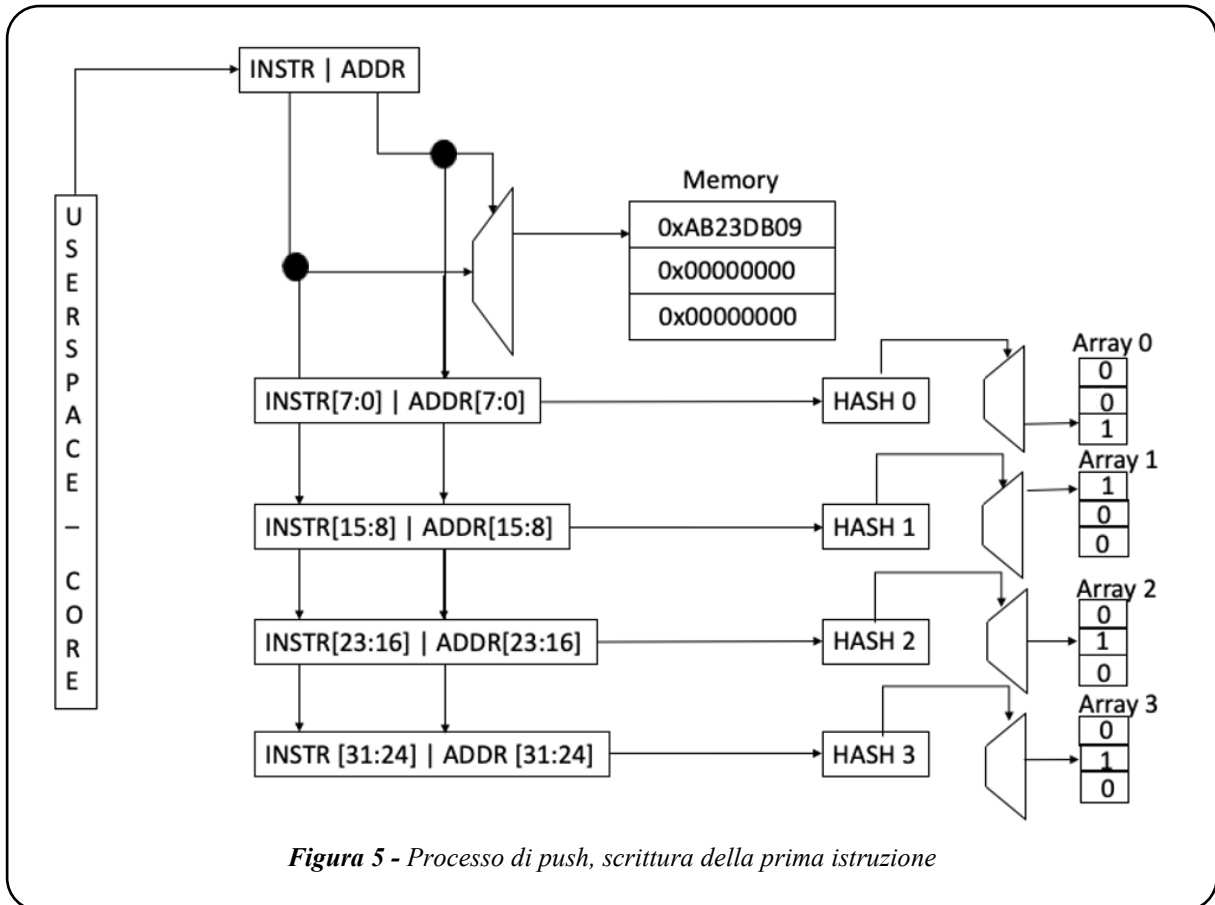
Figura 4 - Processo di push a sinistra, processo di pull a destra (a e b risultano presenti nel set, c e d no)

I filtri di Bloom sono strutture di dati probabilistiche utili per verificare se un elemento è presente in un determinato insieme di dati [26].

Questo tipo di struttura genera la presenza di falsi positivi, ma non consente la possibilità di falsi negativi. Se viene rilevato che un elemento non è nell'insieme, questo risultato è certamente vero, altrimenti abbiamo solo la possibilità che l'elemento sia realmente nel set. Pertanto gli elementi possono essere inseriti nel set, durante il processo di *push*, ma non possono essere rimossi. In secondo luogo, durante la fase di *pull* avviene il controllo dell'effettiva

presenza del dato nel set iniziale o meno. La probabilità di falsi positivi aumenta con il numero di inserimenti. Questi due processi sono illustrati nella figura 4.

Facciamo riferimento all'HT di tipo man in the middle sopracitato, in un contesto di scrittura e lettura di istruzioni di un determinato programma nella memoria istruzioni di un processore. Si può implementare un Filtro di Bloom che abbia come dataset in ingresso la concatenazione delle singole istruzioni e il loro relativo indirizzo di scrittura nella memoria (figure 5 e 6).



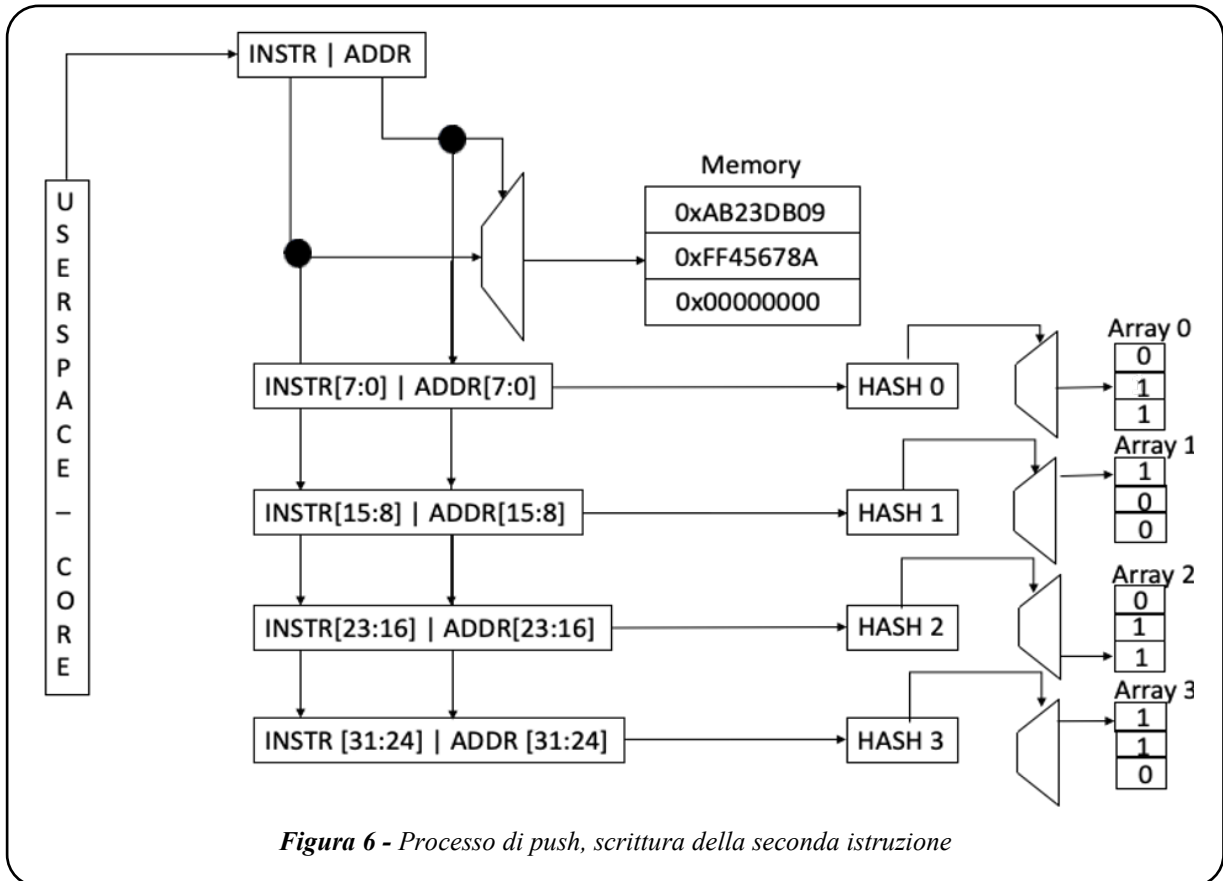


Figura 6 - Processo di push, scrittura della seconda istruzione

Più in dettaglio, consideriamo istruzioni ed indirizzi, provenienti dall'*userspace* o dal core, ciascuno a 32 bit. Dividiamo le singole concatenazioni [istruzione | indirizzo], in quattro segnali da 16 bit ciascuno, 8 bit del valore dell'indirizzo e 8 bit del valore dell'istruzione. Costituiamo il Filtro di Bloom con quattro vettori (inizializzati a 0) ed altrettante funzioni di hash che avranno in input le diverse concatenazioni.

Durante il processo di push, l'output della funzione di hash andrà ad indirizzare il relativo vettore e porrà un 1 in corrispondenza della locazione puntata; lascerà 0 nelle locazioni che non vengono indizzate, ovvero:

$$Array_i[Address_j] = 1$$

$$Address_j = Hash_i(Instr[m:n]_j | AddrOfInstr[m:n]_j)$$

Set 1 di equazioni

$$BitArray_i[Address_j] = Array_i[Address_j]$$

$$Address_j = Hash_i(DataMemory[m:n]_j | AddrOfInstr[m:n]_j)$$

$$AddressOfInstr = AddressOfDataMemory$$

$$Warning = \prod \overline{BitArray_i} = 0$$

Set 2 di equazioni

Una volta terminata l'intera scrittura delle istruzioni in memoria, in sede di esecuzione del programma, ha luogo la fase di pull del filtro di Bloom. In questo processo vi è una differente concatenazione: gli indirizzi continuano ad essere considerati quelli provenienti da userspace e/o core, ma le istruzioni provengono dall'output della memoria. In assenza di manomissioni verranno puntate le locazioni dei vettori dove vi è stato posto precedentemente il valore 1 (la corrispondenza coincide con quella in fase di scrittura, come notiamo nella figura 7 e nel set 2 di equazioni).

Nel caso in cui vi fosse un trojan, che ad esempio somma un valore costante all'indirizzo dal quale si vuole leggere, in output alla memoria sarà posta l'istruzione relativa all'indirizzo specificato più l'offset dato dal trojan, ovvero:

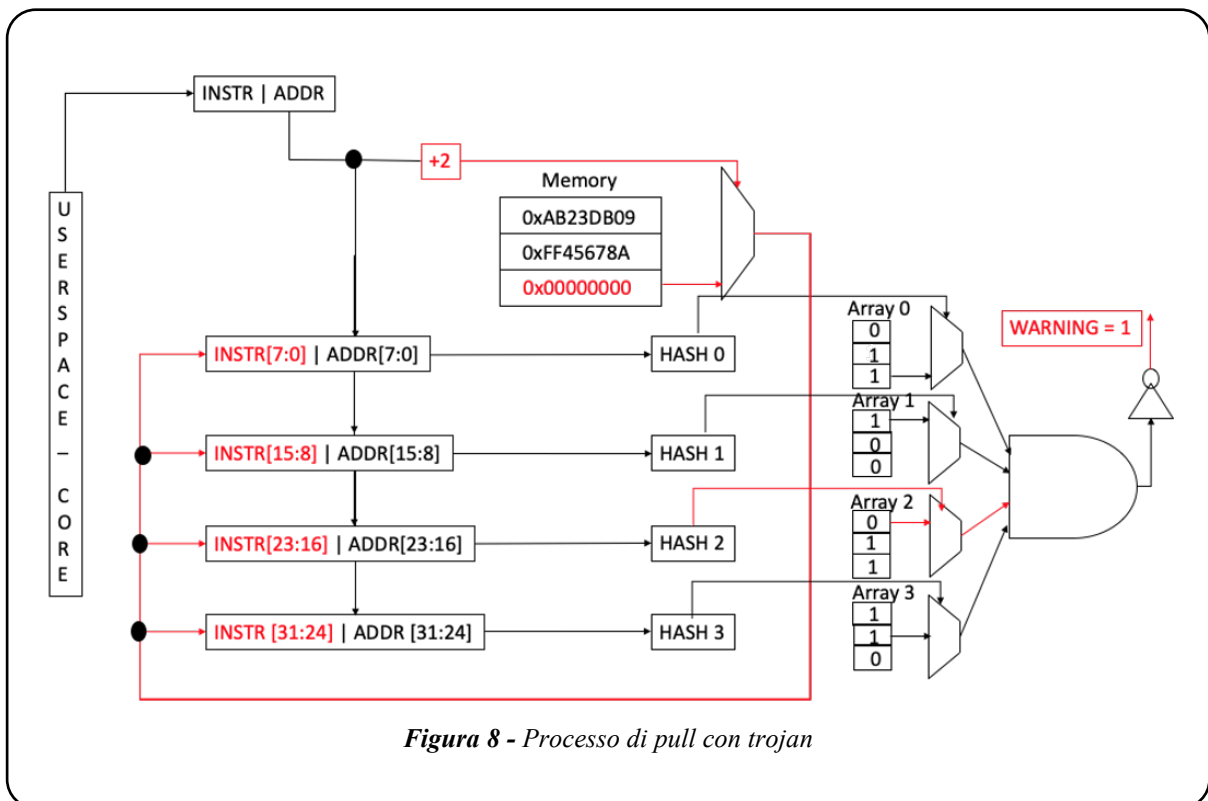
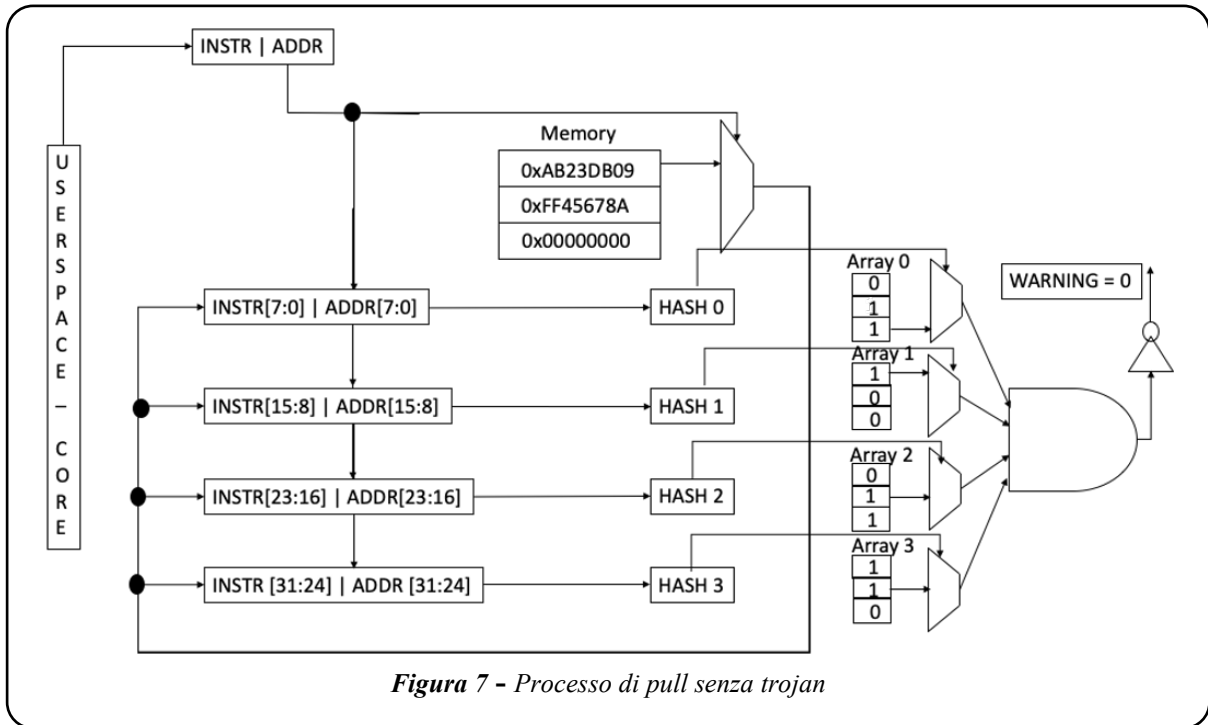
$$\begin{aligned} BitArray_i[Address_j + T] &= Array_i[Address_j] \\ Address_j &= Hash_i(DataMemory[m:n]_j | AddrOfInstr[m:n]_j) \\ AddressOfInstr &\neq AddressOfDataMemory \end{aligned}$$

$$Warning = \prod \overline{BitArray_i} = 1$$

Set 3 di equazioni

In questo scenario sicuramente non vi sarà corrispondenza e nel filtro di Bloom verrà puntata almeno una locazione dove non è stato scritto 1 nel processo di push. Un segnale di warning che valga 1 quando viene puntata almeno una locazione degli array contenente uno 0, può essere implementato ponendo in ingresso ad una porta *NAND* bit presenti nei vettori (figure 7 e 8).

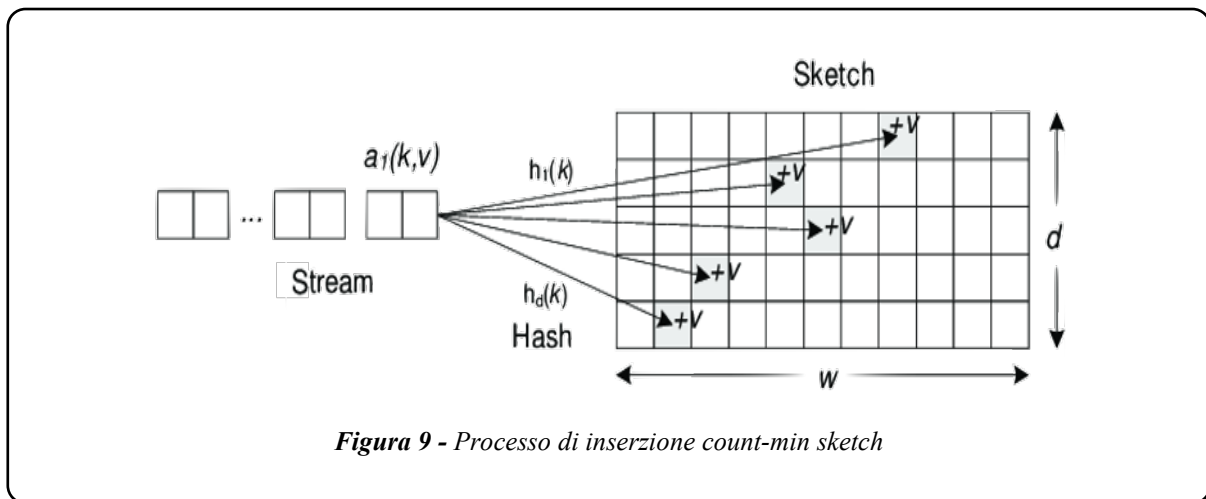
In [27] viene riportato un'altra implementazione, basata su un filtro di Bloom, per la rilevazione di errori nelle istruzioni.



5.2. Checker di attacchi di tipo side-channel basato su Count-min sketch

La struttura *count-min sketch* consiste nell'elaborazione probabilistica di dati [29]. Viene adottata per contare il numero di volte che un elemento è presente in un set [30].

Gli elementi costitutivi del dataset vengono inseriti in una matrice. L'obiettivo di questa architettura è tenere traccia di un flusso di eventi e contare la frequenza dei differenti eventi appartenenti al medesimo flusso. Nel momento in cui si presenta un nuovo evento di tipo w , per ciascuna riga d della matrice, viene puntata la locazione corrispondente all'output della funzione di hash relativa, ovvero $K = Hash_d(w)$. Quindi viene incrementato di uno il valore della locazione K . Questo tipo di architetture può essere utile per rilevare precisi pattern di stream: ogni volta che viene presentato un dato in ingresso verranno incrementate determinate locazioni dello sketch. Inoltre viene registrato anche quante volte si è presentato quel particolare pattern.



Inserendo un'architettura basata su count-min sketch in un processore si possono andare, ad esempio, a contare quante (e quali) istruzioni vengono eseguite. Consideriamo il caso in cui un attaccante abbia installato un software maligno nella memoria del mio core. In particolare facciamo riferimento ad attacchi dove, per diversi scopi, vengono eseguite molteplici volte le medesime istruzioni: Spectre [1] e Orchestration [31] effettuano accessi alla cache in *loops*; Battery Drain [32] esegue la medesima operazione (anche una *NOP* ad esempio), affinché il sistema non vada in *sleep mode* e di conseguenza dissipi potenza.

L'idea consiste nel disporre di una circuiteria che rilevi differenti pattern di istruzioni, quindi che sia programmabile. Il programmatore può specificare di quali opcode si vogliono contare le ricorrenze, settarne il valore massimo (*threshold*) ed impostare ogni quanto tempo resettare i valori dei contatori. Nella figura 10 vi è il *workflow* dell'architettura implementata (figura 11). In primis vengono resettati i valori dei contatori, dopo la programmazione del modulo (4) della figura 11. Durante l'esecuzione del programma, le istruzioni vengono passate al side-channel checker, che effettuerà differenti hash function per la ciascuna istruzione. Gli output di queste funzioni punteranno a determinati contatori che vengono incrementati istruzione dopo

istruzione. Una volta scaduto il *timeout*, se si verifica che i valori dei contatori eccedono il *threshold* precedentemente impostato ed inoltre vi è corrispondenza tra gli opcode definiti dal programmatore e quelli delle istruzioni eseguite, verrà attivato un segnale di warning, altrimenti si ripete il workflow iniziando nuovamente il valore dei contatori.

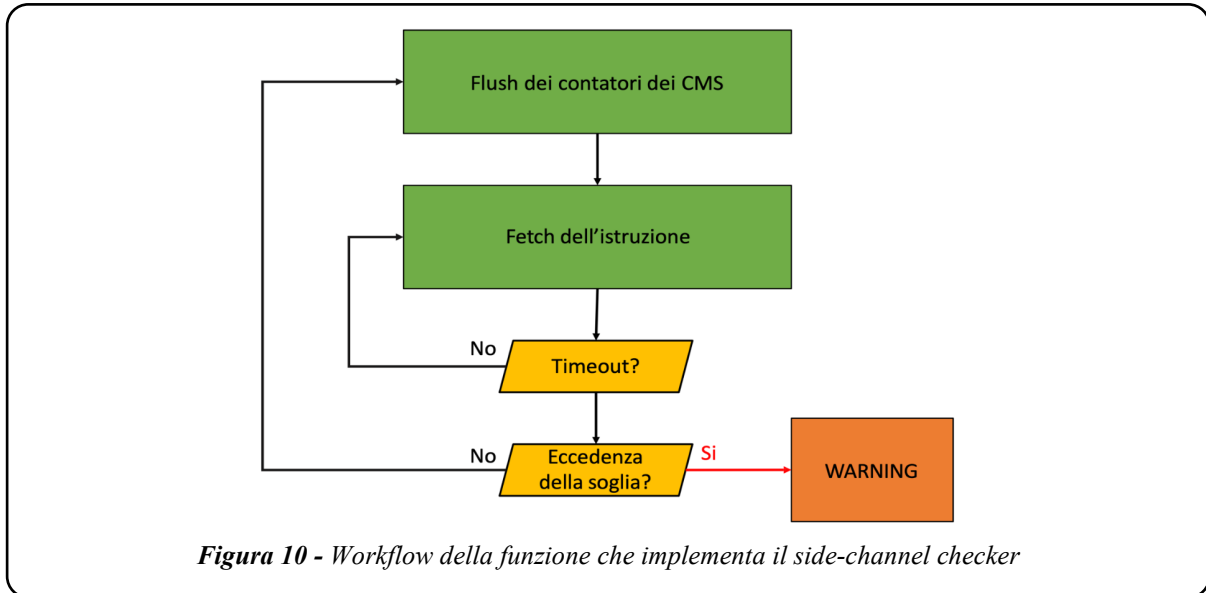


Figura 10 - Workflow della funzione che implementa il side-channel checker

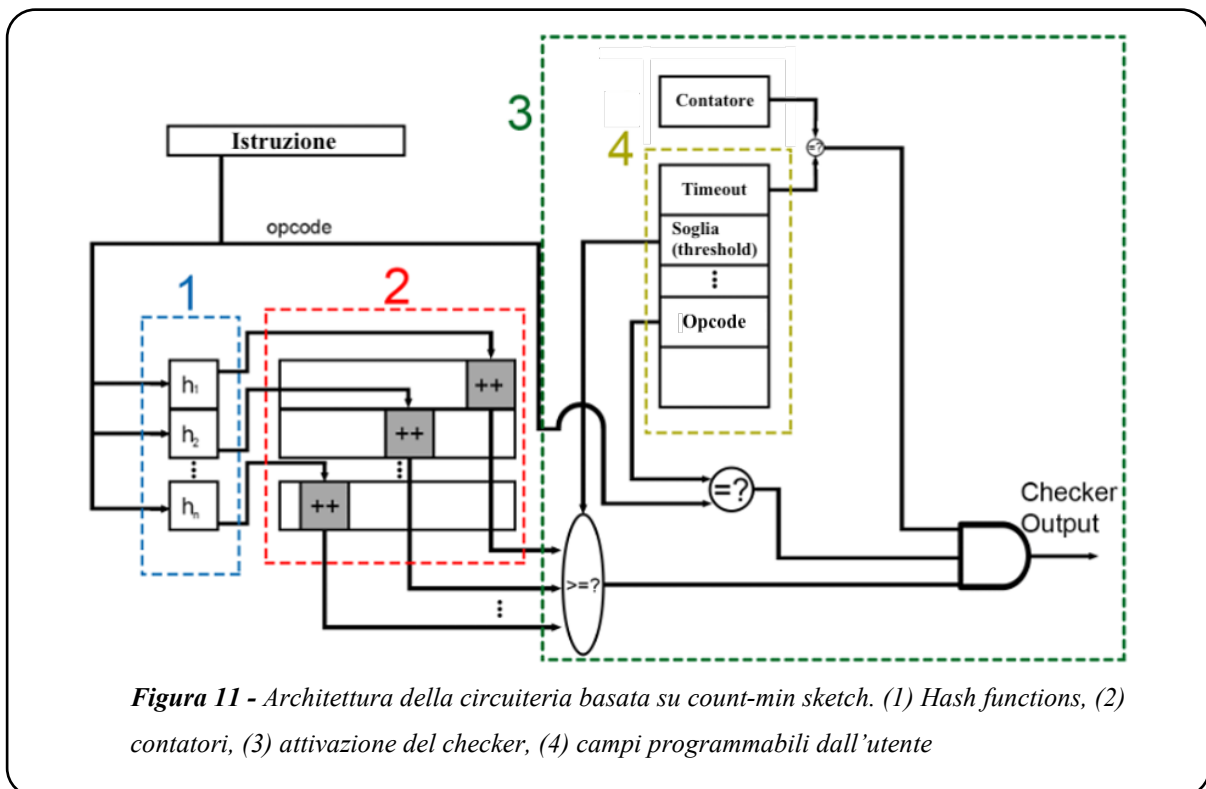


Figura 11 - Architettura della circuiteria basata su count-min sketch. (1) Hash functions, (2) contatori, (3) attivazione del checker, (4) campi programmabili dall'utente

5.3. Analisi delle implementazioni dei checker

Entrambe le circuiterie dei checker sono state integrate in *ibex* [33]: un prototipo di processore a 32 bit basato su un'architettura di tipo RISC-V, con due stage di pipeline e una memoria (istruzioni e dati) totale di 256Kbit. In particolare sono state posizionate delle istruzioni tra il core e la memoria. Le simulazioni sono state effettuate in un primo momento su Verilator [34], successivamente su Vivado [35]. Tramite quest'ultimo tool i checkers sono stati implementati su FPGA. Di seguito vengono riportati i numeri delle implementazioni:

Tabella 2 risorse implementate

	LUT	Flip Flop	Celle BRAM
Core	2519 (70%)	1590 (57.5%)	256 (96.5%)
Filtro di Bloom	91 (2.5%)	39 (1.5%)	6.5 (2.6%)
Count-min sketch	992 (27.5%)	936 (36.5%)	2.5 (0.9%)
Totale	3602 (100%)	2565 (100%)	265 (100%)

Sia l'architettura basata sul filtro di Bloom che quella basata sull'algoritmo di count-min sketch, sono indipendenti dalla piattaforma e possono essere impiegate tra il core e la memoria di processore, indipendentemente dagli stadi di pipeline, il numero di bit e le dimensioni della memoria.

6. Conclusioni

Con l'aumento esponenziale del volume dei dispositivi elettronici connessi ad Internet, garantire la sicurezza dei dati digitali è cruciale. L'hardware gioca un ruolo sempre più importante e fondamentale in questo scenario. Non si può dare per scontato che la circuiteria dove viene eseguito un algoritmo sia affidabile. Infatti, il circuito potrebbe essere stato manomesso e/o presentare molteplici vulnerabilità.

A questo proposito abbiamo proposto due architetture hardware da integrare in un processore, dove la prima (basata sui filtri di Bloom) rileva una possibile manomissione della circuiteria tra la memoria e core; la seconda (basata sull'algoritmo di count-min sketch) segnala l'eventuale fruizione di informazioni, non lecite, da parte di un utente malintenzionato.

Per essere certi che il chip prodotto svolga effettivamente la funzione desiderata possono essere integrati al suo interno moduli come quelli proposti.

Così come in una catena, la sicurezza di un sistema informatico dipende dalla sicurezza del suo componente più vulnerabile. Tali vulnerabilità spesso si nascondono dietro ai dettagli implementativi del circuito e dell'algoritmo in esecuzione e vanno accuratamente tenute in considerazione in fase progettuale con approcci di design for security.

Riferimenti bibliografici

- [1] Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, “Spectre attacks: Exploiting speculative execution” in 40th IEEE Symposium on Security and Privacy (S&P’19), 2019.
- [2] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, “Meltdown: Reading kernel memory from user space” in 27th USENIX Security Symposium (USENIX Security 18), 2018.
- [3] L. Wang and S. Köse, “When hardware security moves to the edge and fog,” in 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP), 2018, pp. 1–5.
- [4] S. R. Chhetri, S. Faezi, N. Rashid, and M. A. Al Faruque, “Manufacturing supply chain and product lifecycle security in the era of industry 4.0,” *Journal of Hardware and Systems Security*, vol. 2, no. 1, pp. 51–68, 2018.
- [5] D. K. Oka, “Securing the automotive critical infrastructure,” in *Cyber-Physical Security*. Springer, 2017, pp. 267–281.
- [6] NIST-FIPS, “Announcing the advanced encryption standard (AES),” *Federal Information Processing Standards Publication*, vol. 197, no. 1-51, pp. 3–3, 2001.
- [7] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978. [Online]. Available: <http://doi.acm.org/10.1145/359340.359342>
- [8] “sha-3 standard: Permutation-based hash and extendable output functions.”
- [9] M. Joye and et al., *Fault analysis in cryptography* Springer, 2012, vol. 147.
- [10] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard, “Systematic classification of side-channel attacks: A case study for mobile devices,” *IEEE Communications Surveys and Tutorials*, vol. 20, no. 1, pp. 465–488, 2018.
- [11] F. Koeune, F.X. Standaert, A tutorial on physical security and side-channel attacks, in: *Foundations of Security Analysis and Design III*, 2005, pp. 78–108.
- [12] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: *CRYPTO*, 1999.
- [13] F. Wang, Formal Verification of Timed Systems: A Survey and Perspective, *Proceedings of the IEEE* (2004) 1283–1305.

- [14] A. Vijayakumar, V.C. Patil, D.E. Holcomb, C. Paar, S. Kundu, Physical design obfuscation of hardware: a comprehensive investigation of device and logic-level technique, *IEEE Transactions on Information Forensics and Security* (2017) 64–77.
- [15] D. Zoni, L. Cremona and W. Fornaciari, "All-Digital Control-Theoretic Scheme to Optimize Energy Budget and Allocation in Multi-Cores," in *IEEE Transactions on Computers*, vol. 69, no. 5, pp. 706-721, 1 May 2020, doi: 10.1109/TC.2019.2963859.
- [16] A. Nahiyani, M. Tehranipoor, Code coverage analysis for IP trust verification, in: *Hardware IP Security and Trust*, Springer, 2017, pp. 53–72.
- [17] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, M. Tehranipoor, Hardware Trojans: lessons learned after one decade of research, *ACM Transactions on Design Automation of Electronic Systems* 22 (2016) 6.
- [18] A. P. Fournaris, L. Pocero Fraile, and O. Koufopavlou, "Exploiting hardware vulnerabilities to attack embedded system devices: a survey of potent microarchitectural attacks", *Electronics*, vol. 6, no. 3, p. 52, 2017.
- [19] A. Barengi, L. Breveglieri, I. Koren, D. Naccache, Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures, *Proceedings of the IEEE* 100 (11) (2012) 3056–3076.
- [20] C. Luo, Y. Fei, P. Luo, S. Mukherjee, and D. Kaeli, "Side-channel power analysis of a gpu AES implementation," in 2015 33rd IEEE International Conference on Computer Design (ICCD). IEEE, 2015, pp. 281–288.
- [21] P. Kocher, Timing Attacks on Implementations of Diffie–Hellman, RSA, DSS, and Other Systems, in: *Advances in Cryptology CRYPTO96*, Springer, 1996, pp. 104–113.
- [22] Canvel B., Hiltgen A., Vaudenay S., Vuagnoux M. (2003) Password Interception in a SSL/TLS Channel. In: Boneh D. (eds) *Advances in Cryptology - CRYPTO 2003*. CRYPTO 2003. Lecture Notes in Computer Science, vol 2729. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-45146-4_34.
- [23] Pedro Reviriego Vasallo, "Bloom Filters: Dependability and Security", Seminar at Tor Vergata, 16 November 2020.
- [24] Reviriego Pedro, et al. "A method to protect Bloom filters from soft errors." 2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS). IEEE, 2015.
- [25] Reviriego Pedro, Jorge A. Martinez, and Marco Ottavi. "Soft Error Tolerant Count Min Sketches." *IEEE Transactions on Computers*, 2020.

- [26] Bloom Filters — A Tutorial, Analysis, and Survey (Blustein & El-Maazawi, 2002)
- [27] Atamaner, Mert & Ergin, Oguz & Ottavi, Marco & Reviriego, Pedro. (2017). Detecting errors in instructions with bloom filters. 1-4. 10.1109/DFT.2017.8244458.
- [28] What is man-in-the-middle attack (MitM)? - Definition from WhatIs.com, su IoT Agenda. URL consultato il 2 dicembre 2020.
- [29] Cormode, Graham (2009). "Count-min sketch". Encyclopedia of Database Systems. Springer. pp. 511–516.
- [30] Cormode, Graham; S. Muthukrishnan (2005) "An Improved Data Stream Summary: The Count-Min Sketch and its Applications". J. Algorithms. 55: 29–38. doi:10.1016/j.jalgor.2003.12.001.
- [31] M. R. Fadiheh, D. Stoffel, C. Barrett, S. Mitra, and W. Kunz.(2018, Dec.) Processor hardware security vulnerabilities and their detection by unique program execution checking. 1812.04975.pdf
- [32] R. Smith, D. Palin, P. Ioulianou, V. Vassilakis, and S. Shahandashti, "Battery draining attacks against edge computing nodes in IoT networks," 01 2020.
- [33] <https://github.com/lowRISC/ibex#ibex-risc-v-core>, 18 December 2020
- [34] <https://www.veripool.org/wiki/verilator/Manual-verilator>, 18 December 2020
- [35] <https://www.xilinx.com/products/design-tools/vivado.html>, 18 December 2020