MINISTERO DELLO SVILUPPO ECONOMICO
DIREZIONE GENERALE PER LA LOTTA ALLA CONTRAFFAZIONE
UFFICIO ITALIANO BREVETTI E MARCHI

UIBM

DOMANDA DI INVENZIONE NUMERO	**102020000006475**
Data Deposito	**27/03/2020**
Data Pubblicazione	**27/09/2021**

Classifiche IPC

Sezione	Classe	Sottoclasse	Gruppo	Sottogruppo
H	04	L	9	30

Titolo

PROCEDIMENTO PER ESEGUIRE OPERAZIONI DI CRITTOGRAFIA SU DATI IN UN DISPOSITIVO DI ELABORAZIONE, CORRISPONDENTI DISPOSITIVO DI ELABORAZIONE E PRODOTTO INFORMATICO

DESCRIZIONE dell'invenzione industriale intitolata:
"Procedimento per eseguire operazioni di crittografia su dati in un dispositivo di elaborazione, corrispondenti dispositivo di elaborazione e prodotto informatico"
di: STMicroelectronics S.r.l., di nazionalità italiana, Via C. Olivetti, 2 - 20864 Agrate Brianza (MB)
Inventore designato: Matteo BOCCHI
Depositata il: 27 marzo, 2020

***

TESTO DELLA DESCRIZIONE

Campo tecnico

La presente descrizione si riferisce a tecniche per eseguire operazioni di crittografia su dati in un dispositivo di elaborazione comprendenti

eseguire uno scambio o swap condizionale su un primo ed un secondo operando basandosi su un valore di bit di controllo

tale scambio condizionale comprendendo

definire una maschera il cui valore dipende dal valore del bit di controllo.

Si possono applicare varie forme di attuazione ad es., a smartcard, microcontrollori, decodificatori o set-top-box utilizzando uno schema di criptaggio o di firma digitale.

Descrizione della tecnica anteriore

I protocolli di crittografia sono protocolli astratti o concreti che eseguono una funzione relativa alla sicurezza, ed applicano procedimenti di crittografia, spesso come sequenze di primitive di crittografia.

Nel campo della protezione contro Attacchi del Canale Laterale in dispositivi che utilizzano algoritmi di crittografia, sono noti ad es. microcontrollori che implementano algoritmi di criptaggio, quali ECC o RSA,

Attacchi del Canale Laterale (SCA) verticali, dove l'attaccante può criptare dati arbitrari (ingresso) utilizzando il dispositivo, al fine di ottenere la chiave di crittografia utilizzata dall'algoritmo di criptaggio. Gli attaccanti registrano le informazioni sul canale laterale durante il criptaggio di dati di ingresso noti, il canale laterale essendo rappresentato tramite consumo di potenza, radiazioni elettromagnetiche, o altre simili quantità.

Il canale laterale è collegato ai dati elaborati dal dispositivo, che sono la chiave di crittografia e i dati dell'attaccante alimentati come ingresso, che rappresentano perciò dati noti.

L'attaccante registra molte "tracce" con dati di ingresso noti diversi ed una chiave costante sconosciuta, facendo ipotesi sul valore di una porzione della chiave di crittografia, ed utilizza procedimenti statistici per verificare tali ipotesi utilizzando le tracce. Per applicare tali procedimenti statistici, l'attaccante ha l'esigenza di utilizzare molte tracce, ciascuna con dati di ingresso noti e diversi ed una chiave costante.

Tuttavia, nella crittografia asimmetrica vi sono modi matematici per modificare la chiave segreta per ciascuna esecuzione in modo che il risultato dell'operazione non cambi, sebbene cambino i dati utilizzati durante il calcolo. Di conseguenza, l'attaccante non può raccogliere molte tracce poiché la chiave non è più costante.

Per questa ragione, quando l'attaccante ha l'esigenza di lavorare su una singola traccia, l'attaccante dispiega i cosiddetti Attacchi Orizzontali.

In un dispositivo da proteggere, ad es. unità di elaborazione o microcontrollore, vi è solitamente una

memoria, che immagazzina dati di ingresso/uscita e valori intermedi, un controllore, che legge parole dalla memoria RAM e le immagazzina in registri, "richiama" successivamente il moltiplicatore sui registri, cioè un'unità moltiplicatrice che esegue l'operazione di moltiplicazione su operandi immagazzinati nei registri. Il moltiplicatore moltiplica gli operandi e scrive il risultato nella memoria RAM. In tale situazione oltre il 95% del calcolo è effettuato all'interno del moltiplicatore.

A tal proposito, è mostrato in figura 1 un esempio di unità di elaborazione 10 che comprende un'unità controllore 11, che legge parole, indicate come a e b in figura 1, da un'unità di memoria 13, ad es. una memoria RAM, e le immagazzina in registri forniti nell'unità controllore 11, richiama successivamente un'unità moltiplicatrice 12 sui contenuti dei registri, cioè le parole a, b. L'unità moltiplicatrice 12 esegue l'operazione di moltiplicazione sugli operandi a, b immagazzinati nei registri del controllore 11. Il moltiplicatore 12 moltiplica gli operandi, ad es. esegue una moltiplicazione di modulo n $R=a\cdot b \bmod n$, e scrive il risultato R nella memoria RAM 13.

Riguardo un'architettura del dispositivo di elaborazione convenzionale, l'unità controllore 11 e il moltiplicatore 12 possono corrispondere all'unità di controllo ed ALU di un'unità di elaborazione quale una CPU, mentre la memoria 13 può essere esterna rispetto al dispositivo di elaborazione 10. Si evidenzia che tali elementi 11, 12, 13 in forme di attuazione variabili possono essere implementate o tramite hardware o software.

Nella crittografia asimmetrica si può ad esempio utilizzare la crittografia RSA (Rivest-Shamir-Adleman) che

comporta un anello che implementa un'Esponenziazione Modulare, o ECC (Crittografia a Curva Ellittica) che implica un anello che implementa la Moltiplicazione Scalare della Curva Ellittica.

Nella crittografia ECC è utilizzato un loop principale che implementa in modo iterativo una moltiplicazione scalare di un dato operando P tramite lo scalare k, la chiave segreta. All'inizio, al di fuori di tale loop, è inizializzato un primo operando Q_0 che punta all'infinito ed un secondo operando Q_1 è impostato sul dato operando P.

Dopo che tutti i bit della chiave segreta k sono stati elaborati nel loop principale, il primo operando Q_0 è restituito come risultato di k*P. Tale loop è implementato come mostrato nel seguente gruppo di istruzioni (1), che rappresenta l'iterazione per il valore di bit della chiave segreta k_i:

```
L1.    if k_i = 0:
L2.    Q_1 = Add(Q_1,Q_0)
L3.    Q_0 = Double(Q_0)
L4.    else:
L5.    Q_0 = Add(Q_0,Q_1)
L6.    Q_1 = Double(Q_1)                    (1)
```

dove k_i è l'i-esimo bit della chiave segreta k, Q_0, Q_1 i primi e secondi operandi letti dalla memoria 13 o anche dai corrispondenti registri dell'unità controllore 11, Add o Aggiungi e Double o Raddoppia le operazioni di addizione del punto della curva ellittica e di raddoppio del punto della curva ellittica. Q_1,Q_0 sono punti espressi in formato proiettivo (X,Y,Z) dove ad esempio (x,y) = X/Z, Y/Z. In forme di attuazione variabili, possono essere utilizzati altri tipi di coordinate proiettive, quali $(x,y)=X/Z^2,Y/Z^3$.

Il loop principale (1) della moltiplicazione del punto

di conseguenza comprende che

per ciascun bit k_i della chiave segreta k, specificatamente ciascun bit i se il bit è 1 bit in lunghezza,

se tale bit k_i è zero, è impostato il secondo operando Q_1, cioè scritto nello stesso indirizzo della memoria 13, corrispondente alla variabile dell'operando, uguale alla somma dei due operandi Q_1,Q_0 e il primo operando Q_0 è moltiplicato per due,

altrimenti o else il primo operando Q_0 è uguale alla somma dei due operandi Q_1,Q_0 e il secondo operando Q_1 è moltiplicato per due.

Inoltre, a titolo di esempio nella RSA il loop principale è implementato dal seguente gruppo di istruzioni (2), che rappresenta l'iterazione per ciascun bit della chiave segreta k_i:

```
M1.    if k_i = 0:
M2.    q_1 = s*q_1
M3.    s = s*s
M4.    else:
M5.    q_0 = s*q_0
M6.    s = s*s                                    (2)
```

dove s è un valore chiamato quadrato, q_0 o q_1 gli operandi letti dalla memoria 13, q_0 in generale che rappresenta una base dell'esponenziazione modulare e q_1 un valore intero fittizio.

L'anello principale della RSA della moltiplicazione del punto di conseguenza comprende che

per ciascun bit k_i della chiave segreta k, specificatamente ciascun bit k_i,

se tale bit k_i è zero, il secondo operando è impostato uguale q_1 al secondo operando q_1 moltiplicato per il

quadrato s, $s*q_1$, e il quadrato s è impostato uguale al quadrato moltiplicato per sé stesso, s*s,

altrimenti il primo operando $s*q_0$ è impostato uguale al primo operando q_0 moltiplicato per il quadrato s, $s*q_0$, e il quadrato s è impostato uguale al quadrato moltiplicato per sé stesso, s*s.

Sono solitamente identificate tre famiglie di Attacchi Orizzontali:

Attacchi sulla Manipolazione dei Bit Segreta

Attacchi su Indirizzi

Attacchi su Dati

Basandosi ad esempio su pattern di lettura/scrittura, per la crittografia asimmetrica ECC con l'anello di moltiplicazione (1), se l'attaccante può capire

- se l'operando Q_0 o Q_1 dell'ECC è letto o scritto durante l'operazione Double, o
- se l'operando Q_0 o Q_1 dell'ECC è scritto durante l'operazione Add,

successivamente l'attaccante può recuperare i bit della chiave segreta k_i.

Basandosi ad esempio su pattern di lettura/scrittura, per la crittografia asimmetrica ECC con l'anello di moltiplicazione (2), se l'attaccante può capire se l'operando q_0 o q_1 della RSA è letto o scritto durante la moltiplicazione, successivamente l'attaccante può recuperare i bit della chiave segreta k_i .

Negli Attacchi su Attacchi Orizzontali di Indirizzi (HA), questa famiglia di attacchi può far conto su una singola traccia e dividere la singola traccia in sottotracce, una per ciascuna iterazione dell'anello, successivamente l'attaccante cerca di trovare somiglianze (o differenze) tra le sottotracce con lo stesso (o diverso)

valore dei bit della chiave segreta k_i .

Con i procedimenti statistici è possibile identificare i cosiddetti "punti di perdita" dove possono essere estratti i bit della chiave segreta k_i.

Ad esempio, IF…ELSE crea rami che producono diverse sequenze di istruzioni eseguite, quindi diverse misure di temporizzazione. L'attaccante è in grado di reperire se il bit chiave era 0 o 1.

Dalla pubblicazione *"Localized Electromagnetic Analysis of Cryptographic Implementations"*, di Johann Heyszl, Stefan Mangard, Benedikt Heinz, Frederic Stumpf, e Georg Sigl è noto un procedimento di protezione da attacchi del canale laterale orizzontale dove si utilizza un'operazione di scambio condizionale aritmetico, o cswap, definito come segue, tramite le istruzioni (3).

I1. def cswap(Q_0,Q_1,c):
I2. T = Q_0+Q_1
I3. Q_0 = T-Q_{1-c} // Q_0=Q_0 if c = 0 else Q_0=Q_1
I4. Q_1 = T-Q_c // Q_1=Q_1 if c = 1 else Q_1=Q_0 (3)

dove T è the somma del valore degli operandi, c è un valore del bit di controllo, Q_{1-c} e Q_c sono valori di sottrazione che corrispondono rispettivamente agli operandi Q_0 o Q_1, a seconda del valore del valore di controllo c, il cui valore determina se il valore degli operandi Q_0 e Q_1 è scambiato, l'uno rispetto all'altro, oppure no.

Nel complesso l'operazione di moltiplicazione scalare, comprendente il loop di moltiplicazione principale, il loop ECC (1) qui nel seguito, può essere implementata come segue, con il gruppo di istruzioni (4).

Preferibilmente, è inizializzato una variabile di maschera swap, N0, su zero, swap = 0, successivamente per ciascun bit k_i, cioè per i=1 a len, len essendo il numero

di bit della chiave segreta k

N1 if swap $\oplus$ k_i = 0:
N2 Q_1 = Add(Q_1,Q_0)
N3 Q_0 = Double(Q_0)
N4 else:
N5 Q_0 = Add(Q_0,Q_1)
N6 Q_1 = Double(Q_1)
N7 r=random_bit()
N8 cswap(Q_0,Q_1,swap$\oplus$r)
N9 swap = r
N10 end for (4)

Come si può vedere una variabile swap è impostata su zero prima di eseguire un'iterazione, ed è successivamente sottoposto a XOR (fase N1) con l'i-esimo bit della chiave segreta k_i per formare lo scambio del bit di controllo mascherato $\oplus$ k_i, il cui valore determina quale dei due diversi insiemi di equazione è implementato nell'anello (1), N2, N3 o N5, N6. Successivamente in una fase N7 un valore del bit casuale r è impostato applicando una funzione di generazione casuale random_bit(), che è o uno zero logico o un uno logico. Il bit casuale r è successivamente sottoposto a XOR con la variabile di maschera swap per determinare il valore del bit di controllo c per la funzione cswap, che è eseguita successivamente (fase N8, corrispondente ad esempio alle istruzioni (3)). Successivamente, prima della successiva iterazione, nella fase N9 la variabile swap è impostata al valore del bit casuale r. Si osserva che l'iterazione è eseguita per ciascun bit della chiave segreta k_i, tuttavia l'operazione di lettura della chiave segreta k per acquisire tali bit dalla memoria può essere eseguita anche parola per parola.

Le fasi N1-N6 corrispondono al loop di moltiplicazione scalare della ECC (1) mostrato sopra, che può essere sostituito in modo equivalente dall'anello di esponenziazione modulare della RSA (2).

Di conseguenza, questo procedimento sfrutta una nota tecnica nello sviluppo del software per evitare problemi di temporizzazione di cache, il che prevede lo scambio in modo condizionale di due interi grandi basandosi sul valore del bit.

Le limitazioni di questo procedimento comprendono il fatto che Attacchi Orizzontali su indirizzi alla lettura del valore di sottrazione Q_c dall'operazione cswap recupereranno il bit casuale r. Inoltre è possibile utilizzare un Attacco Orizzontale per recuperare lo scambio del valore $\oplus k_i$, ed un altro Attacco Orizzontale per recuperare il valore di controllo c. Con entrambi gli attacchi è possibile recuperare i bit della chiave segreta k_i. Inoltre l'operazione $Q_0=Q_0$ per eseguire la funzione cswap è significativamente diversa, ad es. rispetto al consumo, dall'operazione $Q_0=Q_1$. Questo procedimento inoltre non protegge la prima iterazione.

Di conseguenza, lo SWAP condizionale scambia due variabili di ingresso ad es. a,b se il bit di controllo di ingresso o flag c è 1, altrimenti le lascia immutate.

In una forma generale il CSWAP può utilizzare una maschera, uint32_t mask:

```
cswap(a, b, flg):
  uint32_t mask = -flg
  for (i = 0; i < size; i++)
    x = mask & (a[i] XOR b[i])
    a[i] = a[i] XOR x
    b[i] = b[i] XOR x
```

Quando il bit = 0, l'ingresso non cambia

```
cswap(a, b, flg):
  uint32_t mask = -0 = 0
  for (i = 0; i < size; i++)
    x = 0 & (a[i] XOR b[i]) = 0
    a[i] = a[i] XOR 0 = a[i]
    b[i] = b[i] XOR 0 = b[i]
```

Quando il bit = 1, gli ingressi sono scambiati

```
cswap(a, b, flg):
  uint32_t mask = -1 = 0xFFFFFFFF
  for (i = 0; i < size; i++)
    x = a[i] XOR b[i]
    a[i] = a[i] XOR a[i] XOR b[i] = b[i]
    b[i] = b[i] XOR a[i] XOR b[i] = a[i]
```

Qui flg indica il flag, cioè il bit di controllo c dell'operazione CSWAP. I

Di conseguenza, se qualcuno capisce che le variabili sono scambiate oppure no, può rubare la chiave segreta.

Poiché CSWAP è nata per la protezione contro attacchi di temporizzazione, essa è tuttavia vulnerabile all'analisi della potenza, poiché l'analisi della potenza può rilevare se i valori finali cambiano o rimangono gli stessi. Inoltre, la maschera può essere identificata, se essa è 0x00000000 o 0xFFFFFFFF

Scopo e sintesi

Uno scopo di una o più forme di attuazione è di fornire un procedimento per un'operazione di crittografia di dati in un dispositivo di elaborazione comprendente un'esecuzione di un'operazione che risolve gli svantaggi della tecnica anteriore e che sia in particolare più resistente ad Attacchi Orizzontali combinati.

Secondo una o più forme di attuazione, quello scopo è conseguito grazie ad un procedimento avente le caratteristiche specificate nella Rivendicazione 1. Una o più forme di attuazione si possono riferire ad un corrispondente dispositivo di elaborazione che esegue il procedimento e ad un prodotto informatico che può essere caricato nella memoria di almeno un computer e che comprende parti di codice software che sono in grado di eseguire le fasi del procedimento quando il prodotto è eseguito su almeno un computer. Come qui utilizzato, il riferimento ad un tale prodotto informatico è inteso come essere equivalente al riferimento a mezzi leggibili su computer contenente istruzioni per controllare il sistema di elaborazione affinché si coordini l'implementazione del procedimento secondo le forme di attuazione. Il riferimento ad "almeno un computer" è evidentemente inteso per evidenziare la possibilità che le presenti forme di attuazione siano implementate in forma modulare e/o distribuita.

Le rivendicazioni formano parte integrante dell'insegnamento tecnico qui fornito riguardo le varie forme di attuazione.

Secondo la soluzione qui descritta, il procedimento per eseguire operazioni di crittografia su dati in un dispositivo di elaborazione comprende

eseguire uno scambio condizionale sul primo e sul secondo operando basandosi sul valore del bit di controllo

detto scambio condizionale comprendendo

definire una maschera il cui valore dipende dal valore del bit di controllo,

in cui

detto scambio condizionale comprende

convertire il valore del bit di controllo in una prima o seconda stringa di bit di una data lunghezza a seconda del valore del bit di controllo, detta prima o seconda stringa di bit essendo complementare ed avente lo stesso peso di Hamming

impostare come prima maschera la prima stringa e come seconda maschera la seconda stringa

immagazzinare in una variabile temporanea il valore di un'operazione XOR tra il primo e secondo operando come valore temporaneo,

eseguire un'operazione AND bit per bit o bitwise tra la prima maschera e il primo operando ottenendo un primo sottoinsieme di bit

eseguire un'operazione AND bitwise tra la seconda maschera e il secondo operando ottenendo un secondo sottoinsieme di bit

ripetere dette due operazioni AND bitwise scambiando le maschere applicate agli operandi, ottenendo un terzo e quarto sottoinsieme,

eseguire un'operazione OR bitwise tra il primo e il secondo sottoinsieme di bit impostando un quinto sottoinsieme di bit ed un'operazione OR bitwise tra il terzo e il quarto sottoinsieme di bit ottenendo un sesto sottoinsieme di bit,

eseguire un'operazione AND bitwise del quinto sottoinsieme con la prima stringa ottenendo un settimo sottoinsieme di bit ed un'operazione AND bitwise del sesto sottoinsieme con il fattore della seconda stringa ottenendo un ottavo sottoinsieme,

eseguire una AND del settimo ed ottavo sottoinsieme di bit e immagazzinare il risultato come nuovo valore del primo operando,

impostare il nuovo valore del secondo operando come il risultato di uno XOR bitwise tra detto nuovo valore del primo operando calcolato nella fase precedente e detto valore temporaneo.

In forme di attuazione variabili, detta conversione del valore del bit di controllo in una prima o seconda stringa di bit di una data lunghezza dipende dal valore del bit di controllo, detta prima o seconda stringa di bit essendo complementare ed avente lo stesso peso di Hamming comprende

impostare una prima maschera come il valore di un fattore pari alternando un numero uguale di gruppi di uno e zero che terminano con uno zero spostato a destra del valore del bit di controllo,

impostare una seconda maschera sul valore di un fattore dispari, alternando un numero uguale di gruppi di zero e uno che terminano con un uno.

In forme di attuazione variabili, impostare una seconda maschera sul valore di un fattore dispari comprende commutare i bit di una stringa con tutti uno secondo il valore della prima maschera, in particolare applicando operazioni XOR bitwise, in modo che la prima maschera e la seconda maschera scambino il loro valore a seconda del valore del bit di controllo.

In forme di attuazione variabili, il procedimento comprende

immagazzinare il flag in un registro di flag,

immagazzinare il primo operando in un primo registro immagazzinare il secondo operando in un secondo registro,

il procedimento prevede successivamente che

un terzo registro sia inizializzato su zero

un quarto registro sia inizializzato su zero

nel terzo registro sia immagazzinata la prima maschera di detto fattore pari spostato a destra del valore del bit di controllo,

nel quarto registro sia immagazzinata la prima maschera di detto fattore dispari ottenuto commutando i bit di una stringa con tutti uno secondo il valore della prima maschera e sottoposto a XOR.

il primo sottoinsieme di bit sia immagazzinato in un quinto registro,

il secondo sottoinsieme di bit sia immagazzinato in un sesto registro,

il quinto sottoinsieme di bit sia immagazzinato nel quinto registro,

il settimo sottoinsieme sia immagazzinato nel quinto registro,

il terzo sottoinsieme sia immagazzinato in un sesto registro,

la variabile temporanea sia immagazzinata nel secondo registro il cui valore è ottenuto da un'operazione XOR bitwise tra il primo e il secondo operando,

il terzo sottoinsieme sia immagazzinato nel primo registro,

il sesto sottoinsieme sia immagazzinato nel sesto registro,

l'ottavo sottoinsieme sia immagazzinato nel sesto registro,

come nuovo primo operando sia immagazzinato nel primo registro l'AND bitwise del settimo ed ottavo sottoinsieme di bit

come nuovo valore del secondo operando sia immagazzinato nel secondo registro lo XOR bitwise tra detto nuovo valore del primo operando calcolato nella fase

precedente e detto valore temporaneo.

In forme di attuazione variabili, detta conversione del valore del bit di controllo in una prima o seconda stringa di bit di una data lunghezza dipende dal valore del bit di controllo, detta prima o seconda stringa di bit essendo complementare ed avente lo stesso peso di Hamming comprende

impostare una prima maschera come il valore di un fattore pari alternando un numero uguale di gruppi di uno e zero che terminano con uno zero, detto fattore pari essendo spostato a destra del valore del bit di controllo, a sua volta spostato a sinistra di un valore uguale al logaritmo alla base due di detta dimensione del gruppo,

impostare una seconda maschera sul valore di un fattore dispari, alternando un numero uguale di gruppi di zero ed uno che terminano con un uno, detto fattore dispari essendo spostato a sinistra del valore del bit di controllo, a sua volta spostato a sinistra di un valore uguale al logaritmo alla base due di detta dimensione del gruppo,

In forme di attuazione variabili, le operazioni del suddetto procedimento sono comprese in una procedura di crittografia asimmetrica comprendente l'esecuzione di un'operazione iterativa tra un primo operando ed un secondo operando utilizzando una chiave segreta,

detta operazione iterativa comprendendo almeno, per ciascun bit della chiave segreta, le fasi di:

- eseguire operazioni matematiche applicando due diversi insiemi di operazioni al primo operando e al secondo operando a seconda del valore di una funzione del bit della chiave segreta,

- eseguire uno scambio condizionale suddetto primo e

secondo operando basandosi su un valore del bit di controllo corrispondente al valore di detto bit della chiave segreta,

In forme di attuazione variabili, le operazioni del suddetto procedimento sono comprese in una procedura di copia condizionale che assegna in modo condizionale una variabile ad un'altra.

In forme di attuazione variabili, detta procedura di crittografia asimmetrica che comprende eseguire un'operazione iterativa (comprende un anello di moltiplicazione scalare della ECC o un anello esponenziale modulare della RSA).

In forme di attuazione variabili, detta operazione bitwise è implementata tramite un'operazione di rotazione.

In forme di attuazione variabili, comprende immagazzinare nei registri, comprende eseguire ulteriori operazioni di immagazzinamento di ulteriori valori in uno o più ulteriori registri.

La soluzione qui descritta si riferisce anche a forme di attuazione di un dispositivo di elaborazione configurato per eseguire le fasi del procedimento secondo qualsiasi delle rivendicazioni precedenti.

La soluzione qui descritta si riferisce anche ad un prodotto informatico che può essere caricato nella memoria di almeno un computer e che comprende parti di codice software che sono in grado di eseguire le fasi del procedimento secondo qualsiasi delle rivendicazioni precedenti quando il prodotto è eseguito su almeno un computer.

Breve descrizione dei disegni

L'invenzione verrà ora descritta puramente a titolo di esempio non limitativo riguardo ai disegni annessi, in cui:

- la Figura 1 è stata descritta in precedenza;

- la Figura 2 rappresenta un diagramma di flusso di una forma di attuazione del procedimento qui descritto.

Descrizione dettagliata di forme di attuazione

La seguente descrizione illustra vari dettagli specifici mirati ad una comprensione approfondita delle forme di attuazione. Le forme di attuazione possono essere implementate senza uno o più dei dettagli specifici, o con altri procedimenti, componenti, materiali, ecc. In altri casi, note strutture, materiali, o operazioni non sono illustrati o descritti nel dettaglio tale che vari aspetti delle forme di attuazione non verranno resi poco chiari.

Il riferimento ad "una forma di attuazione" nella struttura della presente descrizione intende indicare che una particolare configurazione, struttura, o caratteristica descritta in relazione alla forma di attuazione è compresa in almeno una forma di attuazione. Analogamente, frasi quali "in una forma di attuazione" che possono essere presenti in vari punti della presente descrizione, non si riferiscono necessariamente alla singola e alla stessa forma di attuazione. Inoltre, particolari conformazioni, strutture, o caratteristiche possono essere combinate in modo appropriato in una o più forme di attuazione.

I riferimenti qui utilizzati sono intesi semplicemente per comodità e pertanto non definiscono la sfera di protezione o l'ambito delle forme di attuazione.

Si è osservato che le perdite nella precedente soluzione si basavano sul fatto che il consumo di potenza di un'istruzione dipende dal Peso di Hamming delle variabili coinvolte. Il Peso di Hamming è il numero di 1 binari in una variabile, di conseguenza il procedimento qui

descritto mira a mantenere costante il numero di 1 (e 0) in ciascuna fase della funzione di scambio condizionale, non importa se il flag è 1 o 0.

Quindi, descrivere un'implementazione della funzione di scambio condizionale con operatori bitwise come

```
C1 tmp = a^b
C2 a = [(-flg)&tmp]^a
C3 b = a^tmp
```

in una fase C1, dati gli operandi a,b una variabile temporanea tmp immagazzina il valore di uno XOR bitwise tra gli operandi a e b. Successivamente il primo operando a è impostato uguale allo XOR bitwise tra il primo operando a ed un AND bitwise tra la variabile -flg, che corrisponde al bit di controllo c, e la variabile temporanea tmp.

Il secondo operando b è impostato sullo XOR bitwise tra il primo operando e la variabile temporanea tmp.

Ciò, in generale, corrisponde all'esecuzione di uno scambio condizionale su un primo operando a e su un secondo operando b basandosi sul valore del bit di controllo, sul valore del flag, tale scambio condizionale comprendendo la definizione di una maschera il cui valore dipende dal valore del bit di controllo, in questo caso il valore del flag che agisce direttamente come maschera per la variabile temporanea tmp.

Secondo una forma di attuazione, possono essere invece eseguite le seguenti fasi:

```
S1 m1 = 0xAAAAAAAA >> flg
S2 m2 = 0xFFFFFFFF ^ m1
C1 tmp = a^b
S3 a = {[(m1&a)|(m2&b)]&0xAAAAAAAA}o
  | {[(m1&b)|(m2&a)]&0x55555555}
C4 b = a^tmp
```

In altre parole, una prima maschera m1 è impostata nell'operazione S1 sul valore 0xAAAAAAAA spostato a destra del valore del flag flg o bit di controllo. 0xAAAAAAAA in binario alterna uno e zero che terminano con uno zero, 1010 1010 1010 1010 1010 1010 1010 1010.

Una seconda maschera m2 è impostata nell'operazione S2 sul valore 0xFFFFFF sottoposto a XOR con il valore della prima maschera m1. 0xFFFFFF in binario è rappresentato da una stringa di uno consecutivi, 1111 1111 1111 1111 1111 1111, eseguire lo XOR corrisponde sostanzialmente a commutare i bit di una stringa con tutti uno a seconda del valore della prima maschera m1, in modo che la stringa complementare della prima maschera sia ottenuta come seconda maschera m2.

Di conseguenza, se il flag flg è zero la prima maschera m1 è 0xAAAAAAAA e la seconda maschera m2 0x55555555, se il flag flg è 1 la prima maschera m1 è 0x55555555 e la seconda maschera m2 0xAAAAAAAA. Di conseguenza secondo il valore del flag flg, le maschere m1, m2 scambiano il loro valore tra 0xAAAAAAAA e 0x55555555, che sono, in una rappresentazione binaria, valori complementari, cioè il bitwise di ciascun bit di una stringa è il complementare del corrispondente bit nell'altra stringa, o, come indicato sopra, è il corrispondente bit nell'altra stringa, commutato.

Successivamente, una variabile temporanea tmp immagazzina il valore dello XOR bitwise tra il primo operando a e il secondo operando b.

Il primo operando a è successivamente impostato sul risultato del valore di (m1 AND a OR m2 AND b) AND 0xAAAAAAAA OR (M1 AND b OR m2 AND a) AND 0x55555555. 0x5555555555 in binario alterna gli uno e zero che

terminano con un uno, cioè è uno spostamento di 0xAAAAAAAA. Tutte le operazioni appena indicate sono operazioni bitwise.

Quindi l'operazione S3 è una combinazione fra operazioni logiche bitwise che restituisce a=a se flg=0 e a=b se flg=1.

Nuovamente, il secondo operando b è il risultato del primo operando a XOR la variabile temporanea tmp, cioè b=a XOR TMP (operazione C4).

Di conseguenza, in una forma di attuazione il procedimento per eseguire operazioni di crittografia su dati in un dispositivo di elaborazione comprende

eseguire uno scambio condizionale su un primo e su un secondo operando a,b basandosi sul bit di controllo c o valore di flg

dove lo scambio condizionale comprende definire una maschera il cui valore dipende dal valore del bit di controllo,

comprende il fatto che detto scambio condizionale comprende

convertire il valore del bit di controllo c in una prima e seconda stringa di bit di una data lunghezza, cioè le maschere m1, m2 a seconda del valore del bit di controllo c o flg, detta prima o seconda stringa di bit essendo complementare ed avente lo stesso peso di Hamming, dove la conversione comprende

impostare S1 come prima maschera m1 la prima stringa ed impostare S2 come seconda maschera m2 la seconda stringa, nell'esempio è 0xFFFFFFFF ^ m1, per ottenere 0x55555555 come seconda stringa,

immagazzinare C1 in una variabile temporanea tmp il valore di un'operazione XOR bitwise tra il primo operando a

e il secondo operando b,

eseguire S3 una combinazione fra operazioni logiche bitwise che producono il primo operando a o il secondo operando b a seconda del valore del flag flg, tale combinazione comprendendo:

eseguire un'operazione AND bitwise tra la prima maschera m1 e il primo operando a ottenendo un primo sottoinsieme di bit m1&a,

eseguire un'operazione AND bitwise tra la seconda maschera m2 e il secondo operando b ottenendo un secondo sottoinsieme di bit m2&b

ripetere dette due operazioni AND bitwise scambiando le maschere m1, m2 applicate agli operandi a,b, ottenere un terzo sottoinsieme m1&b e quarto sottoinsieme m2&a, cioè eseguire un'operazione AND bitwise tra la prima maschera m1 e il secondo operando b ed eseguire un'operazione AND bitwise tra la seconda maschera m2 e il primo operando a,

eseguire un'operazione OR bitwise tra il primo e secondo sottoinsieme di bit impostando un quinto sottoinsieme di bit, (m1&a)|(m2&b), ed un'operazione OR bitwise tra il terzo e quarto sottoinsieme di bit ottenendo un sesto sottoinsieme di bit, (m1&b)|(m2&a),

eseguire un'operazione AND bitwise del quinto sottoinsieme con la prima stringa 0xAAAAAAAA, ottenendo un settimo sottoinsieme di bit {[(m1&a)|(m2&b)]&0xAAAAAAAA} ed un'operazione AND bitwise del sesto sottoinsieme con la seconda stringa 0x55555555 ottenendo un ottavo sottoinsieme {[(m1&b)|(m2&a)]&0x55555555},

eseguire un AND bitwise del settimo ed ottavo sottoinsieme di bit ed immagazzinare il risultato,

{[(m1&a)|(m2&b)]&0xAAAAAAAA}|{[(m1&b)|(m2&a)]&0x55555555}, come nuovo valore del primo operando a,

impostare C4 il nuovo valore del secondo operando b come il risultato di uno XOR bitwise tra tale nuovo valore del primo operando, cioè variabile a, calcolato nella fase precedente e tale valore in tale valore temporaneo tmp.

Nella forma di attuazione esemplificativa mostrata sopra

convertire il flg del bit di controllo o valore c in una prima o seconda stringa di bit di una data lunghezza a seconda del valore del bit di controllo (c), detta prima o seconda stringa di bit essendo complementare ed avente lo stesso peso di Hamming comprende

impostare una prima maschera m1 come il valore di un fattore pari 0xAAAAAAAA alternando un numero uguale di gruppi di uno e zero che terminano con uno zero spostato a destra del valore del bit di controllo,

impostare una seconda maschera m2 sul valore di un fattore dispari, 0xFFFFFF, alternando un numero uguale di gruppi di zero e uno che terminano con un uno.

Quindi, indicando successivamente con

R0 ← flg, cioè immagazzinare il flag flg in un registro di flag R0,

R1 ← a, cioè immagazzinare il primo operando in un primo registro R1

R2 ← b, .e. Immagazzinare il secondo operando in un secondo registro R2,

i registri R0, R1, R2 di un'unità di elaborazione che implementano il procedimento qui descritto, in cui sono immagazzinati il flag flg e gli operandi, la soluzione descritta richiede le seguenti operazioni di registro:

R3 ← 0, cioè registro R3 è inizializzato su zero

R4 ← 0, cioè registro R4 è inizializzato su zero

R3 ← 0xAAAAAAAA >> R0, cioè nel registro R3 è immagazzinata la prima maschera m1 che è impostata sul valore: 0xAAAAAAAA spostato a destra del valore del flag (fase S1).

R4 ← 0xFFFFFFFF ^ R3, cioè nel registro R4 è immagazzinata la seconda maschera m2 che è impostata su 0xFFFFFFFF sottoposto a XOR con la prima maschera m1 (fase S2).

R5 ← R3 & R1, cioè è immagazzinato nel registro R5 m1&a

R6 ← R4 & R2, cioè è immagazzinato nel registro R6 m2&b

R5 ← R5 | R6, cioè è immagazzinato nel registro R5 (m1&a)|(m2&b)

R5 ← R5 & 0xAAAAAAAA, cioè è immagazzinato nel registro R5 (m1&a)|(m2&b)]&0xAAAAAAAA}

R6 ← R3 & R2, cioè è immagazzinato nel registro m1&b

R2 ← R1 ^ R2, cioè è immagazzinato nel registro R2 a XOR b, cioè la variabile temporanea tmp

R1 ← R4 & R1, cioè è immagazzinato nel registro R1 m2&a

R6 ← R1 | R6, cioè è immagazzinato nel registro R6 (m1&b)|(m2&a)

R6 ← R6 & 0x55555555, cioè è immagazzinato nel registro R6 {[(m1&b)|(m2&a)]&0x55555555}

R1 ← R5 | R6 cioè è immagazzinato nel registro R1 {[(m1&a)|(m2&b)]&0xAAAAAAAA}or|{[(m1&b)|(m2&a)]&0x55555555}

R2 ← R1 ^ R2 cioè è immagazzinato nel registro R2, a^tmp come valore del secondo operando b (fase C4).

Come mostrato sopra, il procedimento comporta due assegnazioni di variabili, uno spostamento, sei AND

bitwise, per un totale di 15 operazioni, tre OR bitwise, e tre XOR bitwise, che coinvolgono sette registri R0…R6.

Nella tabella qui sotto è indicato il Peso di Hamming HW delle suddette operazioni.

Operazione di registro	HW
R3 ← 0	0
R4 ← 0	0
R3 ← 0xAAAAAAAA >> R0	50%
R4 ← 0xFFFFFFFF ^ R3	50%
R5 ← R3 & R1	~25%
R6 ← R4 & R2	~25%
R5 ← R5 | R6	~50%
R5 ← R5 & 0xAAAAAAAA	~25%
R6 ← R3 & R2	~25%
R2 ← R1 ^ R2	~50%
R1 ← R4 & R1	~25%
R6 ← R1 | R6	~50%
R6 ← R6 & 0x55555555	~25%
R1 ← R5 | R6	~50%
R2 ← R1 ^ R2	~50%

Tabella 1

In generale, data una dimensione di parola W degli operandi a, b, indicato un parametro di dimensione N come 1,2,4,8,16…,W/2, un fattore pari Feven, se, ad es. la dimensione di parola W è 32, può essere selezionata fra i cinque valori per la dimensione N=1, 2… 16: 0x….A…A/0x….C…C/0x….F0…F0/0xFF00..FF000/0xFFFF0000, cioè un'alternanza di uno e zero che terminano con zero in gruppi di dimensione di N bit, N=1, 2, 4, 8, 16.

Allo stesso modo, il fattore dispari Fodd può essere

0x5…5/0x….3…3/0x….OF…OF/0x00FF..00FF/0x0000FFFF, cioè un'alternanza di uno e zero che terminano con uno in gruppi di dimensione di N bit, N=1, 2, 4, 8, 16.

Di conseguenza, la prima maschera m1 può essere

m1=Feven>>(flg<<$\log_2$N)

e la seconda maschera m2

m2=Fodd<<(flg<<$\log_2$N)

a={[(m1&a)|(m2&b)]&Feven} |{[(m1&b)|(m2&a)]&Fodd}

b=a^(a^b)

Di conseguenza, in questo caso il fatto di convertire il valore del bit di controllo o flag flg in una prima o seconda stringa di bit di una data lunghezza a seconda del valore del bit di controllo, detta prima o seconda stringa di bit essendo complementare ed avente lo stesso perso di Hamming comprende

impostare una prima maschera m1 come il valore di un fattore pari Feven alternando un numero uguale di gruppi di uno e zero che terminano con uno zero, detto fattore pari Feven essendo spostato a destra del valore del bit di controllo, cioè flag flg, a sua volta spostato a sinistra di un valore uguale al logaritmo binario, o logaritmo alla base due, di detta dimensione N del gruppo,

impostare una seconda maschera m2 sul valore di un fattore dispari Fodd, alternando un numero uguale di gruppi di zero e uno che terminano con un uno, detto fattore dispari Fodd essendo spostato a sinistra del valore del flag flg, a sua volta spostato a sinistra di un valore uguale al logaritmo alla base due di detta dimensione N del gruppo.

Nell'impostare il valore del bit di controllo, eseguire `flg<<$\log_2$N` significa che il flag flg è spostato a sinistra di un numero di bit che è il logaritmo alla base

2 di N.

Ad esempio, se si sceglie N=16, allora Feven=0xFFFF0000 e Fodd=0x0000FFFF. Successivamente:

se flg=0, allora m1=0xFFFF0000>>(0<<4)=0xFFFF0000

Se flg=1, allora m1=0xFFFF0000>>(1<<4) = 0xFFFF0000>>16 = 0x0000FFFF

In figura 2 è mostrato un diagramma esemplificativo di una forma di attuazione dello scambio condizionale qui descritto.

Con S0 è indicata una fase aggiuntiva di selezione di un parametro di dimensione N per selezionare il numero di bit nei gruppi del fattore pari Feven e del fattore dispari Fodd.

Successivamente il procedimento comprende un blocco S1 che rappresenta il fatto di impostare come prima maschera m1 la prima stringa, ad es. fattore pari Feven, ed un blocco S2 che rappresenta il fatto di immagazzinare come una seconda maschera m2 la seconda stringa, ad es. fattore dispari Fodd

Successivamente un blocco C1 rappresenta ripristinare in una variabile temporanea tmp il valore di un'operazione XOR bitwise tra un primo a e secondo b operando fornito come ingresso nel blocco C1. Il blocco S3 rappresenta il fatto di eseguire una combinazione S3 di operazioni logiche bitwise che producono il primo operando o il secondo operando a seconda del valore del flag (flg), secondo la suddetta espressione, ad es. a={[(m1&a)|(m2&b)]&Feven} |{[(m1&b)|(m2&a)]&Fodd}.

Infine il blocco C4 rappresenta il fatto di impostare il nuovo valore del secondo operando come risultato di uno XOR bitwise tra il precedente valore del primo operando e il precedente valore del secondo operando.

Le maschere potrebbero anche essere due sequenze complementari di 0 e 1 in modo che ciascuna maschera abbia lo stesso numero di 0 e 1

In forme di attuazione variabili il procedimento qui descritto può anche essere utilizzato per assegnare in modo condizionale una variabile ad un'altra (Copia Condizionale)

- if (flag == 1)
- x = y

In forme di attuazione varianti, poiché l'immagazzinamento di R3 ← 0xAAAAAAAA >> R0 può essere diverso se R0, ad es. il flag, è 0 o 1, successivamente può essere utilizzata la rotazione a destra di 30 o 31 bit, cioè R3 ← ROR(0xAAAAAAAA, R0 + 30)

Una rotazione tramite il valore 30 corrisponde al fatto di non eseguire alcun cambiamento.

Una rotazione del valore 31 corrisponde al fatto di eseguire uno spostamento a destra di una posizione.

Successivamente, l'operazione di spostamento bitwise può essere implementata tramite un'operazione di rotazione.

Inoltre, se vi sono operazioni di scrittura nello stesso registro troppo vicine alla precedente operazione di scrittura nello stesso registro, ad es. R5 ← R3 & R1 e R5 ← R5 & 0xAAAAAAAA, possono essere utilizzate istruzioni inutili che eliminano la perdita, come ad esempio scrivere in un'ulteriore registro R7 valori inutili.

Qui di seguito vi è un esempio della soluzione che implementa tali due tecniche:

R3 ← 0

R4 ← 0

R3 ← ROR(0xAAAAAAAA, R0 + 30)

R4 ← 0xFFFFFFFF ^ R3

R5 ← R3 & R1

R6 ← R4 & R2

R5 ← R5 | R6

R6 ← R3 & R2

R2 ← R1 ^ R2

R1 ← R4 & R1

R6 ← R1 | R6

R7 ← R7 + R7

R5 ← R5 & 0xAAAAAAAA

R7 ← R7 + R7

R6 ← R6 & 0x55555555

R1 ← R5 | R6

R2 ← R1 ^ R2

Di conseguenza, l'immagazzinamento nei registri può comprendere eseguire ulteriori operazioni di immagazzinamento di ulteriori valori in uno o più ulteriori registri

La soluzione secondo le varie forme di attuazione qui descritte consente di ottenere i seguenti vantaggi.

Questa soluzione impiega in modo vantaggioso una cswap che protegge in modo intrinseco contro l'analisi della potenza poiché mantiene costante il consumo di potenza, grazie all'invarianza del peso di hamming in ogni fase del processo.

Il mascheramento condizionale fa perdere informazioni sul flag condizionale attraverso l'analisi della potenza. Di fatto, le maschere consentono di continuare il calcolo solo per il valore corretto, e il modo in cui è scelto permette di recuperare il flag.

Nelle forme di attuazione, la soluzione qui descritta divide gli ingressi in due parti, e mette insieme le corrispondenti parti appartenenti ai diversi ingressi.

In questo modo, le maschere non devono scegliere tra

parole diverse, ma tra spostamenti della stessa parola, estraendo/eliminando una quantità costante di bit da ciascuna parola.

Naturalmente, senza pregiudizio ai principi delle forme di attuazione, i dettagli di costruzione e le forme di attuazione possono variare notevolmente rispetto a ciò che è stato qui descritto ed illustrato puramente a titolo di esempio, senza così discostarsi dall'ambito delle presenti forme di attuazione, come definito dalle seguenti rivendicazioni.

RIVENDICAZIONI

1. Procedimento per eseguire operazioni di crittografia su dati in un dispositivo di elaborazione comprendente

eseguire uno scambio condizionale (I1-I4) su un primo (a) ed un secondo operando (b) basandosi su un valore del bit di controllo (c; flg)

detto scambio condizionale comprendendo

definire una maschera il cui valore dipende dal valore del bit di controllo (c; flg),

in cui

detto scambio condizionale comprende

convertire (S1, S2) il valore del bit di controllo (flg) in una prima (Feven, 0xAAAAAAAA) o seconda stringa (Fodd, 0X55555555) di bit di una data lunghezza a seconda del valore del bit di controllo (flg), detta prima o seconda stringa di bit essendo complementare ed avente lo stesso peso di Hamming, detta conversione comprendendo

impostare (S1) come prima maschera (m1) la prima stringa e immagazzinare (S2) come seconda maschera (m2) la seconda stringa,

immagazzinare (C1) in una variabile temporanea (tmp) il valore di un'operazione XOR bitwise tra il primo (a) e secondo (b) operando come valore temporaneo,

eseguire una combinazione (S3) fra operazioni logiche bitwise che producono il primo operando o il secondo operando a seconda del valore del bit di controllo (flg), detta combinazione comprendendo:

eseguire un'operazione AND bitwise tra la prima maschera (m1) e il primo operando (a) ottenendo un primo sottoinsieme di bit,

eseguire un'operazione AND bitwise tra la seconda

maschera (m2) e il secondo operando (b) ottenendo un secondo sottoinsieme di bit

ripetere dette due operazioni AND bitwise scambiando le maschere (m1, m2) applicate agli operandi (a,b), ottenendo un terzo ed un quarto sottoinsieme,

eseguire un'operazione OR bitwise tra il primo e secondo sottoinsieme di bit impostando un quinto sottoinsieme di bit ed un'operazione OR bitwise tra il terzo e quarto sottoinsieme di bit ottenendo un sesto sottoinsieme di bit,

eseguire un'operazione AND bitwise del quinto sottoinsieme con la prima stringa (Feven) ottenendo un settimo sottoinsieme di bit ed un'operazione AND bitwise del sesto sottoinsieme con il fattore della seconda stringa (Fodd) ottenendo un ottavo sottoinsieme,

eseguire un AND bitwise del settimo ed ottavo sottoinsieme di bit e immagazzinare il risultato come nuovo valore del primo operando (a),

impostare (C4) il nuovo valore del secondo operando (b) come il risultato di uno XOR bitwise tra detto nuovo valore del primo operando (a) calcolato nella fase precedente e detto valore temporaneo (tmp).

2. Procedimento secondo la rivendicazione 1, in cui

detta conversione del valore del bit di controllo (flg) in una prima o seconda stringa di bit di una data lunghezza a seconda del valore del bit di controllo (flg), detta prima o seconda stringa di bit essendo complementare ed avente lo stesso peso di Hamming comprende

impostare una prima maschera (m1) come il valore di un fattore pari (Feven, 0xAAAAAAAA) alternando un numero

uguale di gruppi di uno e zero che terminano con uno zero spostato a destra del valore del bit di controllo (flg),

impostare una seconda maschera (m2) sul valore di un fattore dispari (Fodd, 0x55555555), alternando un numero uguale di gruppi di zero e uno che terminano con un uno.

3. Procedimento secondo la rivendicazione 2, in cui

impostare una seconda maschera (m2) sul valore di un fattore dispari (Fodd, 0x55555555) comprende commutare i bit di una stringa con tutti uno secondo il valore della prima maschera (m1), in particolare applicando un'operazione XOR bitwise, in modo che la prima maschera (m1) e seconda maschera (m2) scambino il loro valore a seconda del valore del bit di controllo (flg).

4. Procedimento secondo qualsiasi delle rivendicazioni 1 a 3, comprendente

immagazzinare il flag (flg) in un registro di flag (R0),

immagazzinare il primo operando (a) in un primo registro (R1)

immagazzinare il secondo operando (b) in un secondo registro (R2),

il procedimento che prevede successivamente che

un terzo registro (R3) sia inizializzato su zero

un quarto registro (R4) sia inizializzato su zero

nel terzo registro (R3) detto fattore pari (Feven, 0xAAAAAAAA) spostato a destra del valore del bit di controllo (flg) sia immagazzinata la prima maschera (m1),

nel quarto registro (R4) sia immagazzinato detto fattore dispari (Fodd, 0x55555555) ottenuto commutando i bit di una stringa con tutti uno secondo il valore della prima maschera (m1) e sottoposto a XOR con la prima maschera (m1).

il primo sottoinsieme di bit sia immagazzinato in un quinto registro (R5),

il secondo sottoinsieme di bit sia immagazzinato in un sesto registro (R6),

il quinto sottoinsieme di bit sia immagazzinato nel quinto registro (R5),

il settimo sottoinsieme sia immagazzinato nel quinto registro (R5),

il terzo sottoinsieme sia immagazzinato in un sesto registro (R6),

la variabile temporanea (tmp) sia immagazzinata nel secondo registro (R2) il cui valore è ottenuto da un'operazione XOR bitwise tra il primo (a) e il secondo (b) operando,

il terzo sottoinsieme sia immagazzinato nel primo registro (R1),

il sesto sottoinsieme sia immagazzinato nel sesto registro (R6),

l'ottavo sottoinsieme sia immagazzinato nel sesto registro (R6),

come nuovo primo operando (a) sia immagazzinato nel primo registro (R1) l'AND bitwise del settimo ed ottavo sottoinsieme di bit

come nuovo valore del secondo operando (b) sia immagazzinato nel secondo registro (R2) lo XOR bitwise tra detto nuovo valore del primo operando (a) calcolato nella fase precedente e detto valore temporaneo (tmp).

5. Procedimento secondo la rivendicazione 1, in cui

detta conversione del valore del bit di controllo (flg) in una prima o seconda stringa di bit di una data lunghezza a seconda del valore del bit di controllo (flg), detta prima o seconda stringa di bit essendo complementare

ed avente lo stesso peso di Hamming comprende

impostare una prima maschera (m1) come il valore di un fattore pari (Feven) alternando un numero uguale di gruppi di uno e zero che terminano con uno zero, detto fattore pari (Feven) essendo spostato a destra del valore del bit di controllo (flg), a sua volta spostato a sinistra di un valore uguale al logaritmo alla base due di detta dimensione (N) del gruppo,

impostare una seconda maschera (m2) sul valore di un fattore dispari (Fodd, 0xFFFFFF), alternando un numero uguale di gruppi di zero e uno che terminano con un uno, detto fattore dispari (Fodd) essendo spostato a sinistra del valore del bit di controllo (flg), a sua volta spostato a sinistra di un valore uguale al logaritmo binario, o logaritmo alla base due di detta dimensione (N) del gruppo.

6. Procedimento secondo qualsiasi delle rivendicazioni precedenti, dove le operazioni di detto procedimento sono comprese in

una procedura di crittografia asimmetrica che comprende eseguire un'operazione iterativa tra un primo operando (a) ed un secondo operando (b) utilizzando una chiave segreta (k),

detta operazione iterativa comprendendo almeno, per ciascun bit (k_i) della chiave segreta (k), le fasi di:

- eseguire operazioni matematiche applicando due diversi insiemi di operazioni al primo operando (a) e al secondo operando (b) a seconda del valore di una funzione del bit (k_i) della chiave segreta (k),

- eseguire uno scambio condizionale su detto primo (a) e secondo operando (b) basandosi su un valore del bit di controllo (c) corrispondente al valore di detto bit (k_i) della chiave segreta (k).

7. Procedimento secondo qualsiasi delle rivendicazioni precedenti, dove le operazioni di detto procedimento sono comprese in una procedura di copia condizionale che assegna in modo condizionale una variabile ad un'altra.

8. Procedimento secondo la rivendicazione 6, caratterizzato dal fatto che detta procedura di crittografia asimmetrica che comprende eseguire un'operazione iterativa (N1-N10; S1-S14) comprende un anello di moltiplicazione scalare della ECC o un anello di esponenziazione modulare della RSA.

9. Procedimento secondo qualsiasi delle rivendicazioni precedenti, caratterizzato dal fatto che detta operazione bitwise è implementata tramite un'operazione di rotazione.

10. Procedimento secondo qualsiasi delle rivendicazioni precedenti, caratterizzato dal fatto che l'immagazzinamento nei registri comprende inoltre eseguire ulteriori operazioni di immagazzinamento di ulteriori valori in uno o più ulteriori registri.

11. Dispositivo di elaborazione (10) configurato per eseguire le fasi del procedimento secondo qualsiasi delle rivendicazioni 1 a 10.

12. Prodotto informatico che può essere caricato nella memoria di almeno un computer e che comprende parti di codice software che sono in grado di eseguire le fasi del procedimento secondo qualsiasi delle Rivendicazioni 1 a 10 quando il prodotto è eseguito su almeno un computer.

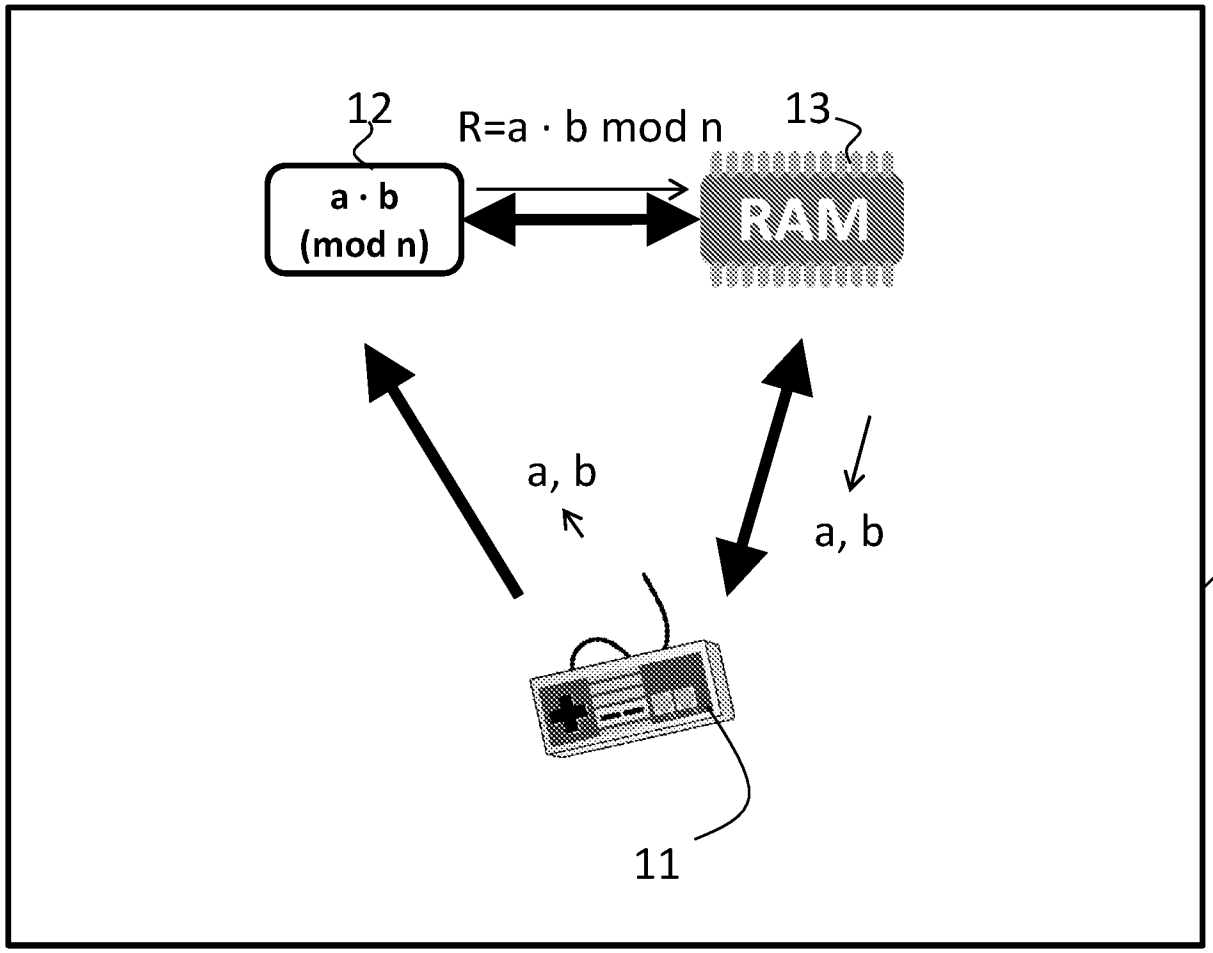

Fig. 1

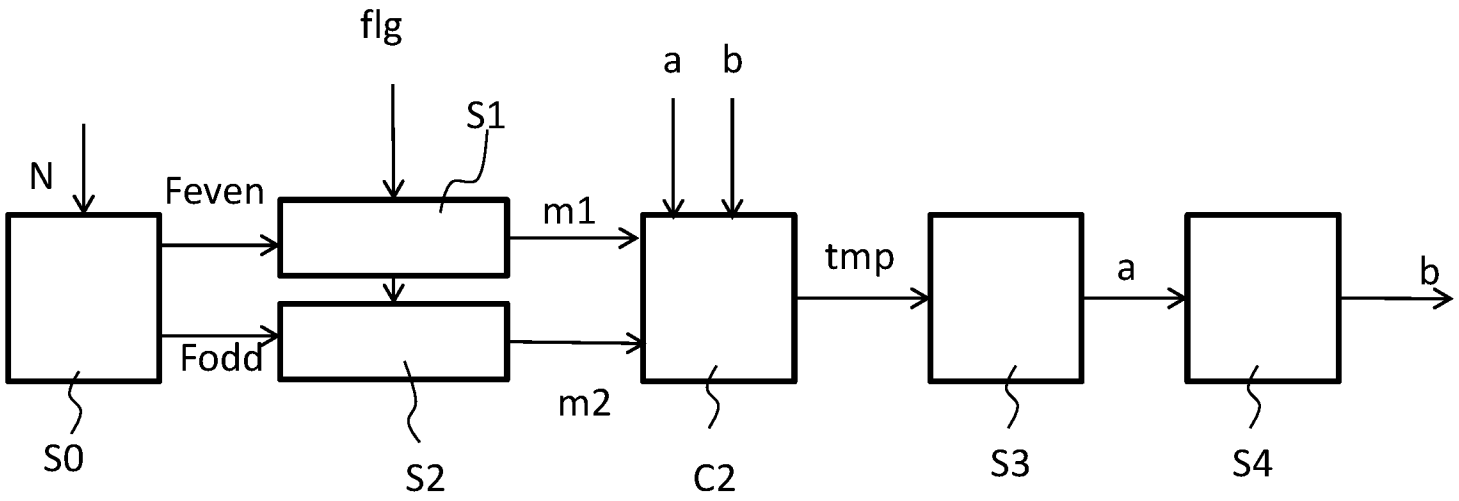

Fig. 2