Modellazione in SystemC di un cifrario basato su algoritmo XTEA con possibili utilizzi

Massimiliano Incudini - VR433300

***Sommario*—Il seguente documento documenta il lavoro svolto durante il la prima parte del modulo di laboratorio del corso Progettazione di Sistemi Embedded (aa 2018/2019). Il suo obiettivo è illustrare le scelte progettuali effettuate per ognuna delle diverse versioni del cifratore XTEA.**

I. Introduzione

Il progetto consiste nello sviluppare un cifrario il cui funzionamento è basato sull'algoritmo eXtended TEA fornito insieme alla consegna. Il cifrario dovrà essere implementato come modulo (insieme al proprio testbench) nelle versioni:

- RTL: il modulo viene diviso in due processi, uno che ne rappresenta la propria FSM e l'altro che ne rappresenta il datapath;
- TLM untimed: il modulo viene implementato attraverso le classi di TLM senza tenere traccia dell'informazione temporale della simulazione;
- TLM loosely timed: il modulo è simile al precedente, con la differenza che tiene parzialmente conto dell'informazione temporale ed utilizza la tecnica del temporal decoupling per ottimizzare le prestazioni della simulazione;
- TLM approximatively timed 4-phases: il modulo è progettato per essere interrogato attraverso chiamate TLM non bloccanti, ogni transazione attraversa quattro fasi che richiedono sincronizzazione esplicita.

Viene inoltre richiesto di implementare un sistema formato da un controllore, una valvola ed un serbatoio d'acqua. Il controllore riceve il livello dell'acqua dal serbatoio, e manda alla valvola il comando per aprire e chiudere il rubinetto di conseguenza. Il sistema dovrà essere sviluppato in SystemC-AMS.

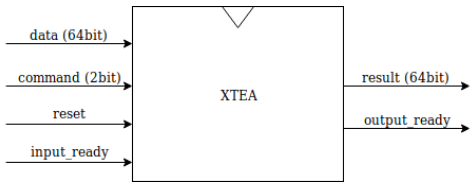
Infine viene richiesto di modificare il sistema precedente implementando il controllore con TLM ed assicurandosi che mandi le transazioni criptate con XTEA, aggiungengo transattori ai moduli AMS e ponendo tra il controllore e la valvola un decifratore XTEA in RTL (con proprio transattore).

II. Modellazione RTL

La prima versione del cifratore (RTL) viene implementata a partire dal sorgente fornito. Il primo compito svolto è stato schematizzare il codice in un diagramma di flusso (Figura 1).

Questo permette di estrarre gli input e gli output che servono al modulo per funzionare. Una prima soluzione potrebbe essere prende in input tutti i dati che servono in una volta sola: la chiave, i dati da elaborare, la modalità di esecuzione.

In realtà l'interfaccia risulta grossolana: il cifratore probabilmente dovrà elaborare un discreto numero di dati utilizzando sempre la stessa chiave (intesa come il gruppo di 128 bit). Risulta inutile passarla sempre quando per la maggior parte delle computazioni questa rimarrà identica. Si sceglie quindi di togliere l'input relativo alla chiave. Questa verrà passata all'inizio della computazione utilizzando la porta `data`. Poiché la porta `data` è di 64bit e la chiave è grande il doppio, occorrerà passarla in due volte. Otteniamo lo schema che segue:



Le porte hanno il seguente significato:

- `clock` (in input);
- `reset` (in input): quando attivo alto, riporta il modulo allo stato iniziale;
- `command` (in input, 2bit): comando da eseguire sul modulo;
- `data` (in input, 64bit): valore da processare;
- `input_ready` (in input): quando attivo alto, si possono leggere i dati in input;
- `result` (in output, 64bit): risultato della computazione;
- `output_ready` (in output): quando è attivo alto, la computazione è terminata;

L'azione corrispondente ai valori forniti in input è presente in Tabella I. Una volta che il modulo ha finito la computazione, viene attivato il segnale `output_ready` e viene mantenuto attivo un solo ciclo di clock. Durante questo periodo di tempo si può leggere il valore della porta `result`.

Si è deciso di dividere il modulo in due funzioni separate: una funzione è `fsm` e l'altra `datapath`. La prima ha il compito di leggere i dati in input e memorizzarli all'interno del modulo, scandire l'avanzamento della computazione modificando la variabile di stato, e scrivere i valori in output. La seconda si occupa di elaborare espressioni aritmetiche anche semplici.

Si può notare nell'algoritmo in Figura 1 che buona parte dei due blocchi più grandi, quelli contenenti i calcoli, si sovrappone. Per questo il modulo cercherà di sfruttare quando può questa sovrapposizione per minimizzare l'area (del datapath, nella quale vengono effettuate queste computazioni).

Inoltre le operazioni index $= x\&3$, index $= (x \gg 11)\&3$ usate per calcolare quale parte di chiave si traducono in una semplice estrazione di bit dall'operando, quindi non serve passare dal datapath.

Il compito della FSM è legge gli input e li salva all'interno delle propri registri, gestisce lo stato, comunica col datapath e scrive gli output. Il datapath legge i dati dai registri della FSM e li sposta nei propri registri effettua le operazioni aritmetiche.

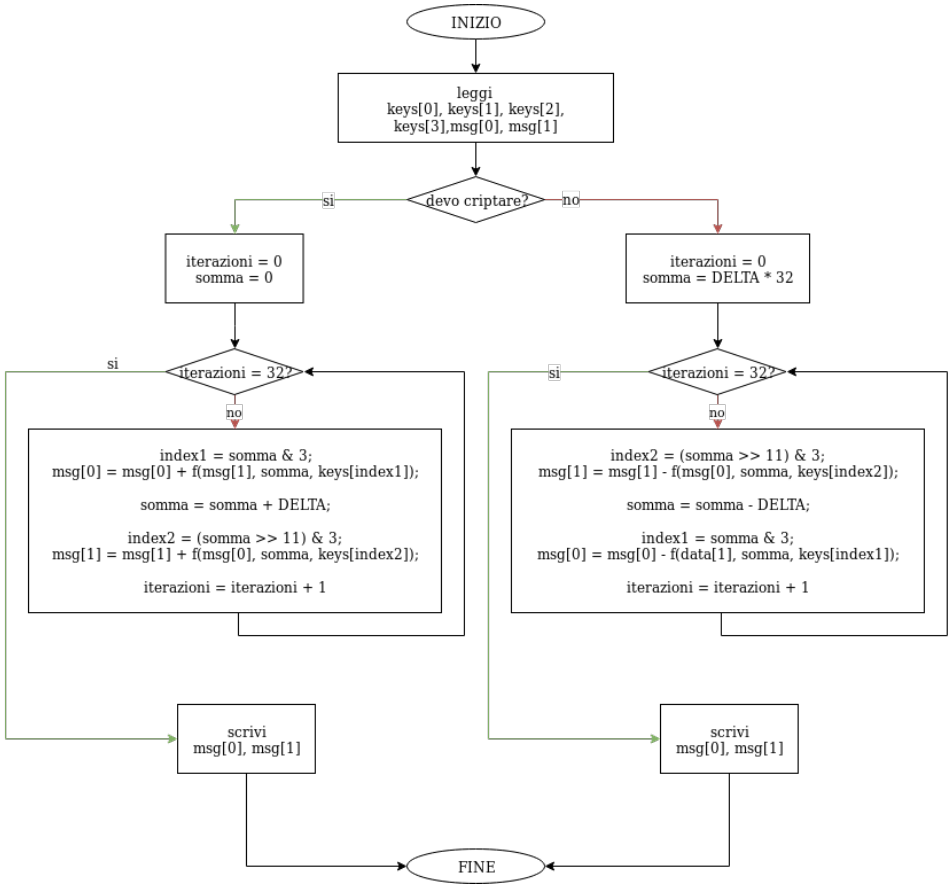


Figura 1. Algoritmo ad alto livello del cifratore. NB: La funzione $f(a, b, c)$ è un'espressione che corrisponde a $(((a \ll 4) \oplus (a \gg 5)) + a) \oplus (b + c)$.

reset	input_ready	command	azione
1	-	--	riporta il sistema allo stato iniziale
0	0	--	non fare nulla
0	1	00	memorizza il valore su data all'interno del sistema come i primi 64bit della chiave
0	1	01	memorizza il valore su data all'interno del sistema come i secondi 64bit della chiave
0	1	10	avvia la criptazione, il messaggio è il valore su data
0	1	11	avvia la decriptazione, il messaggio è il valore su data

Tabella I
TABELLA DELLE AZIONI

In Figura 2 puoi vedere la mappa dei registri. Ogni registro è scritto o solo dalla FSM o solo dal datapath.

Lo schema compatto di FSM e datapath è presente sottoforma di schema in Figura 3. I cerchi sono gli stati, le etichette sulle freccie indicano la condizione con la quale viene cambiato lo stato. In rettangoli verdi sono i calcoli svolti dal datapath. Come si può notare, le operazioni fatte in alcuni stati sono uguali, il che ci permette di ottimizzare per area il nostro modulo.

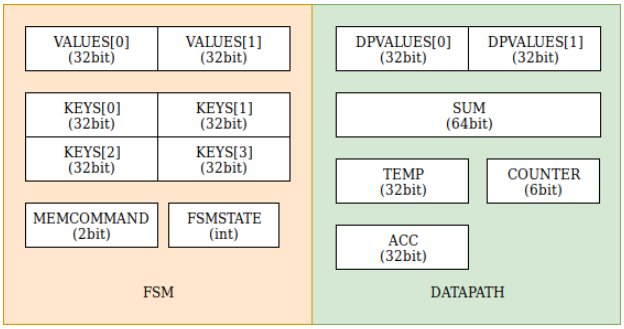


Figura 2. Registri interni al modulo

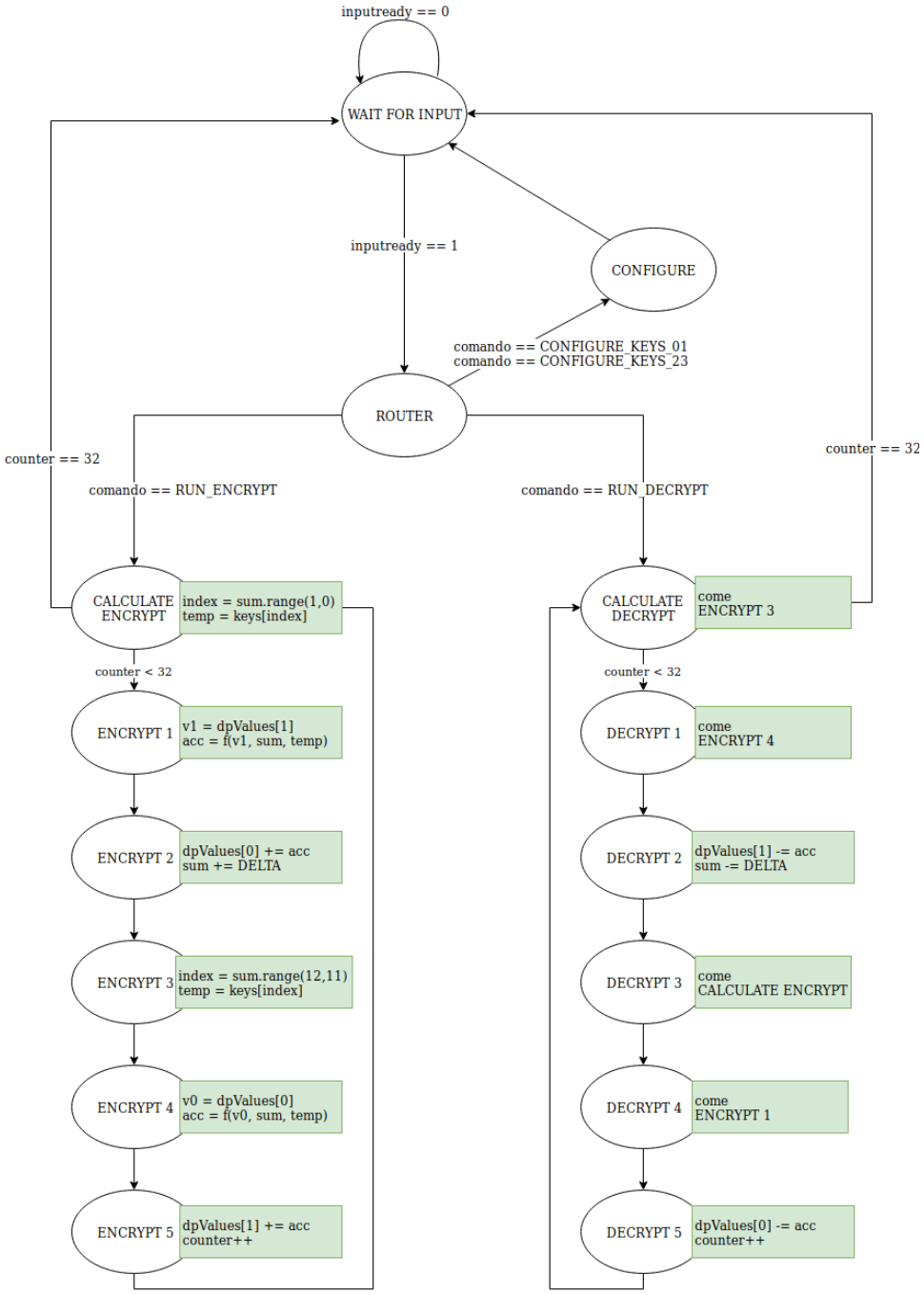


Figura 3. Diagramma degli stati. Le transazioni sono comandate dalla FSM. In verde sono presenti le elaborazioni fatte dal datapath

III. Modellazione TLM

Le successive versioni del cifratore sono modellate usando il TLM. Il framework è stato sviluppato al di sopra di SystemC, ogni modulo TLM è un modulo di SystemC che eredita anche ulteriori interfacce. Il modulo XTEA (target) implementa anche l'interfaccia `tlm::tlm_fw_transport_if<>` (forward) mentre il suo testbench implementa l'interfaccia `tlm::tlm_bw_transport_if<>` (backward). I due moduli comunicano attraverso porte dette dette socket. Le interfacce forward devono avere al loro interno almeno una socket *target* (per ricevere transazioni) e quelle backward devono avere almeno una socket *initiator* (per iniziare transazioni).

Le transazioni si scambiano sempre una struttura `tlm::tlm_generic_payload` che al suo interno contiene un puntatore ad un oggetto definito dall'utilizzatore, che rappresenta il dato da scambiare durante la transazione. Nel nostro caso questo oggetto è un'istanza di `struct Packet`. Essa ha al suo interno il comando per il target ed l'operando su cui lavorare.

A. Modellazione TLM Untimed

Il comportamento del target è codificato nella funzione `b_transport` che prende in input il `tlm_generic_payload`

ed una variabile che rappresenta il tempo. La chiamata, che verrà effettuata dall'initiator, è bloccante.

Il payload può essere di lettura o scrittura. Nel nostro caso è sempre sia di scrittura (il target esegue l'operazione) che di lettura (il target ritorna l'eventuale risultato).

In questo caso l'informazione temporale non viene aggiornata.

B. Modellazione TLM Loosely Timed

Questa versione differisce dalla precedente perchè viene parzialmente introdotta la gestione dell'informazione temporale. Il target nella funzione `b_transport` aggiornerà il tempo di esecuzione sommando il tempo, reale o presunto, dell'operazione richiesta dalla transazione. Si è a conoscenza del tempo reale solamente se si è già implementato il modulo a livello RTL. Nel nostro caso, possiamo settare il tempo pari a 798 cicli di clock. Il tempo di un ciclo di clock è richiesto come parametro del costruttore del target.

La simulazione perfettamente sincronizzata dei componenti è costosa. Essa può essere velocizzata attraverso il *temporal decoupling*. I processi possono essere eseguiti oltre il tempo della simulazione (non cedendo quindi il controllo allo scheduler) per un tempo massimo fissato detto *quanto di tempo* o fino a che non è richiesta una sincronizzazione esplicita. Il guadagno di tempo si ottiene quindi riducendo il numero di context-switching. L'oggetto che mantiene la sincronia è detto *quantum keeper*. Il metodo `sync` del quantum keeper cede il controllo allo scheduler.

Il testbench conterrà quindi un oggetto della classe `tlm_utils::tlm_quantumkeeper` che ne gestisce la sincronizzazione.

C. Modellazione TLM Approximatively Timed (4 phases)

La transazione è implementata tramite una sequenza di chiamate non bloccanti `nb_transport_[fw|bw]`, ognuna delle quali è una certa *fase* della transazione. Solitamente le fasi sono quattro: `BEGIN_REQ`, `END_REQ`, `BEGIN_RESP`, `END_RESP`. Si possono comunque mettere un numero arbitrario di fasi, fino a rendere la simulazione anche precisa fino al clock.

L'initiator chiama la forward con fase `BEGIN_REQ`, e il target termina la transazione con fase `END_REQ`, nel frattempo inizia a fare i calcoli per completare l'operazione richiesta. Una volta completata, il target chiama la backward con fase `BEGIN_RESP` e i dati richiesti, ed l'initiator chiude la transazione con fase `END_RESP`.

D. Confronto delle prestazioni

Procediamo a fare un confronto delle prestazioni delle quattro implementazioni. Ogni volta le chiavi vengono inizializzate una volta sola, poi vengono criptate e decriptate un milione di word generate casualmente. I risultati sono visibili in Tabella II.

Possiamo concludere che le prestazioni peggiori si ottengono con il modello RTL che è il più preciso. I tempi sono misurati con l'utility `time`.

	UT	LT	AT4	RTL
Real time	1.149s	1.757s	1.672s	174.062s
User time	1.144s	1.752s	1.672s	174.062s
System time	0.004s	0.004s	0.000s	0.004s

Tabella II
CONFRONTO DELLE PRESTAZIONI

IV. MODELLAZIONE AMS

Il sistema è composto da una tanica d'acqua, dalla valvola che regola l'apertura della tanica e dal proprio controllore.

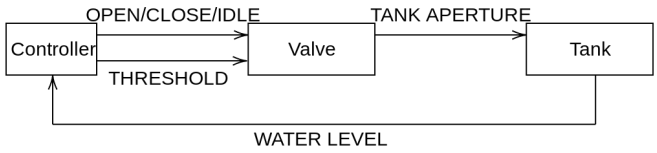


Figura 4. Schema del sistema tanica d'acqua

La tanica mantiene al suo interno l'acqua. Il livello di riempimento è dato dalla funzione $x(t)$. Occorre che $5.0 \leq x(t) \leq 8.8$. Il livello di riempimento è dato dal sistema

$$\begin{cases} \dot{x} = 0.6a - 0.03x \\ x(0) = 0 \end{cases}$$

La valvola aggiorna il livello proprio livello di apertura a a seconda del comando che riceve dal controllore. I valori esatti sono presenti in Tabella III. In ogni caso occorre che $0 \leq a(t) \leq T$ con T threshold in input dal controllore. Il valore iniziale si suppone sia $a(0) = 0$.

Il controllore monitore il livello dell'acqua e comanda la valvola di conseguenza. I comandi generati sono `OPEN`, `IDLE` e `CLOSE`. Un threshold (inteso come valore massimo) viene aggiornato secondo i valori di Tabella III e mandato alla valvola. Il suo valore iniziale è $T(0) = 0.7$.

Segnale	Min lev	Max lev	Molt. threshold	Molt. a
`OPEN`	0.0	5.0	1.1x	0.25x
`IDLE`	5.0	8.8	1.0x	0.00x
`CLOSE`	8.8	∞	0.7x	-0.25x

Tabella III
COMANDI DATI DAL CONTROLLORE ALLA VALVOLA, RANGE DI LIVELLO DI ACQUA DEL COMANDO, MOLTIPLICATORE PER L'AGGIORNAMENTO DEL THRESHOLD, MOLTIPLICATORE PER L'AGGIORNAMENTO DEL LIVELLO DI APERTURA

A. Valvola

La valvola è implementata tramite il modello discreto non conservativo AMS-TDF. Il suo compito è aggiornare il valore dell'apertura a seconda del comando ricevuto. La classe è definita tramite la macro `SCA_TDF_MODULE` e tutta la logica del componente è racchiusa nel metodo `processing`.

B. Controllore

Il controllore è implementato tramite il modello discreto non conservativo AMS-TDF. Il suo compito è ricevere il livello

dell'acqua dal serbatoio e generare i comandi ed il threshold da passare alla valvola. a classe è definita tramite la macro `SCA_TDF_MODULE` e tutta la logica del componente è racchiusa nel metodo `processing`. Viene settato un ritardo obbligatorio sulla porta in input per poter definire un ordine di esecuzione per i componenti (scheduling). Il ritardo viene settato nel metodo `set_attributes`.

C. Serbatoio

Il serbatoio è implementato tramite AMS-LSF (a tempo continuo, non conservativo). Poichè il resto del sistema è a

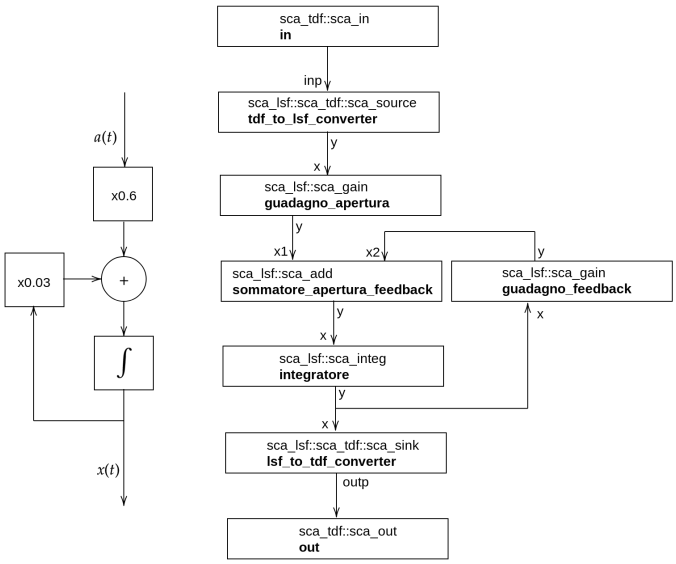
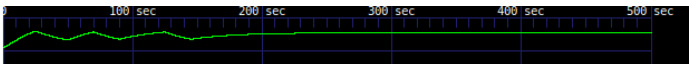


Figura 5. Serbatoio. A sinistra lo schema ad alto livello, a destra lo schema con i componenti LSF. Per ogni blocco, la prima riga nel rettangolo è il tipo del componente, la seconda riga il nome, le etichette vicino alle frecce i nomi delle porte del componente.

tempo discreto abbiamo due elementi che convertono il segnale TDF in LSF continuo e viceversa. I componenti utilizzati sono visibili in Figura 5.

D. Risultato

Vediamo di seguito che dopo diverse oscillazioni il livello dell'acqua si stabilizza ad 8.507.



V. Modellazione Heterogeneous Platform

La piattaforma eterogenea contiene componenti scritti con diversi formalismi e connessi tutti insieme. Essi sono:

- controller, non più implementato con AMS ma come TLM (scelto modello untimed per semplicità;
- valvola, in AMS-TDF;
- serbatoio, in AMS-LSF internamente e AMS-TDF come interfaccia esterna;
- decrifratore XTEA, implementato in RTL.

Sono necessari ulteriori componenti "collante":

- transattore che permette al sistema valvola-serbatoio di interfacciarsi con l'esterno tramite TLM;
- interfaccia RTL alla valvola, obbligatoria perchè TLM non riesce a comunicare direttamente con AMS;
- interfaccia RTL al serbatoio;
- transattore che collega controller, decifratore e sistema valvola-serbatoio; esso passa le richieste del controllore al resto del sistema, eventualmente decriptate.

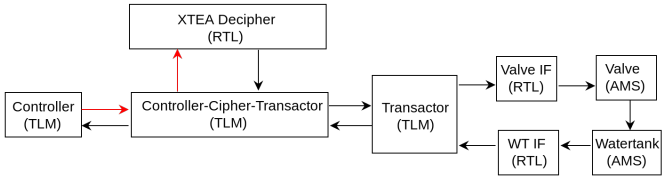


Figura 6. Schema della piattaforma eterogenea. Le freccie in rosso rappresentano il flusso di dati criptato.

Lo schema del sistema può essere visto in Figura 6. Ci sono due flussi di transazioni:

- da `controller` al `controller-cipher-transactor`;
- da `controller-cipher-transactor` a `transactor`.

Nel primo caso l'initiator è `controller`. Se la transazione è in scrittura allora questa conterrà i valori di threshold ed il comando per la valvola. I valori saranno criptati. Il target interroga il decifratore, ed una volta finita l'operazione crea una nuova transazione (secondo flusso). Non vengono ritornati valori all'initiator eccetto l'esito. Se la transazione è in lettura il target porterà avanti la richiesta con una transazione del secondo flusso. Il risultato viene ritornato all'initiator. In questo caso il decifratore non viene usato.

Nel secondo caso l'initiator è `controller-cipher-transactor` ed il target è il sistema valvola-serbatoio. Se la transazione è in scrittura allora passo i dati del payload alla valvola, altrimenti leggo i dati del serbatoio.