Consortium GARR
8° BORSISTI DAY
"ORIO CARLINI"
MERCOLEDÌ 14 DICEMBRE 2016 - ROMA

Protocolli per sistemi cloud federati sicuri e affidabili

Borsista: Dott. Giacomo ODOARDI
Tutor: Ch.mo Prof. Luca SPALAZZI

UNIVERSITÀ POLITECNICA DELLE MARCHE
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
cini Cyber Security National Lab

Ambito

- Paradigma computazionale:
  - *cloud-storage*

- Scenario di riferimento:
  - *sanità*

  - Collaborazione del gruppo di ricerca con Federazione Italiana Medici di Medicina Generale (F.I.M.M.G.)

Requisiti da soddisfare

- **Affidabilità**: capacità di garantire un corretto funzionamento nel tempo → massimizzazione della disponibilità agli utenti finali.
  - Il personale sanitario deve sempre essere in grado di accedere ai dati del paziente!
- **Sicurezza**: protezione dei dati da accessi illeciti (*confidenzialità*) e da modifiche non autorizzate (*integrità)*.
  - Ancor più importante nel mondo sanitario, ove si trattano dati SENSIBILI, che vanno protetti ai sensi di legge (cfr. D.Lgs. 196/03)
- **Prestazioni**: qualsiasi soluzione ingegneristica deve essere valutata anche sotto il profilo dell'efficienza.

Obiettivi del progetto

- **Progettazione** di protocolli di storage su *Interconnected-cloud* in grado di garantire livelli di sicurezza e affidabilità adeguati, con particolare riguardo allo scenario individuato.

- **Verifica** attraverso tecniche formali della rispondenza di tali protocolli ai requisiti prefissati.

- **Realizzazione** di un prototipo software per testare concretamente il funzionamento di tali protocolli.

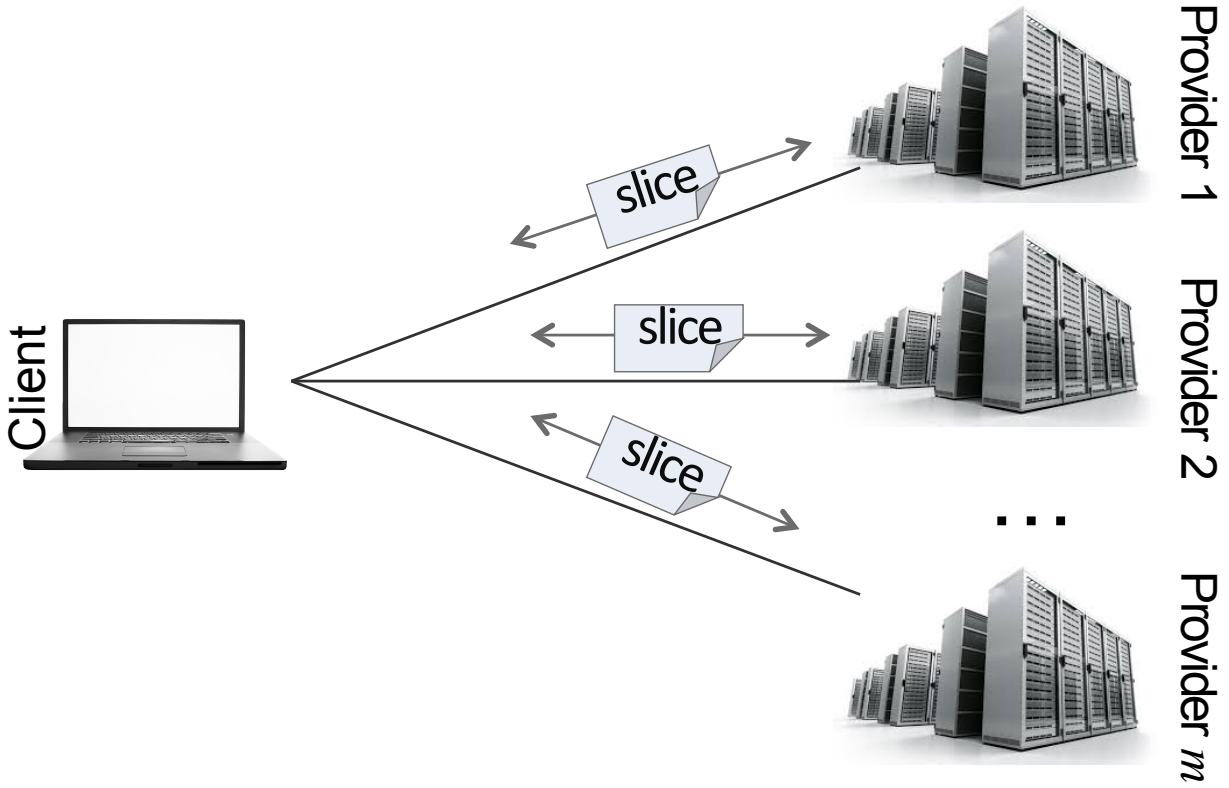
Idee chiave del progetto

1. All Or Nothing Transform per aumentare la confidenzialità e l'integrità.

2. Codifica a correzione di errore per aumentare l'affidabilità.

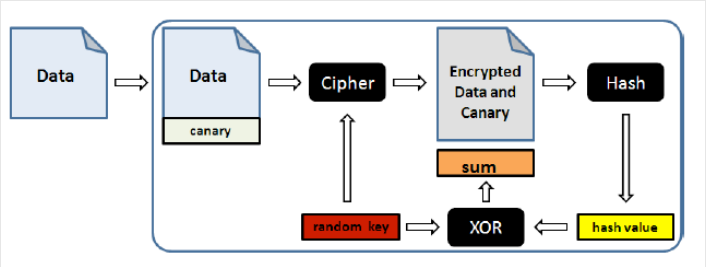
3. Interconnected cloud storage per aumentare affidabilità e prestazioni.

1. All Or Nothing Transform



1. All Or Nothing Transform

Cifratura + slicing



1. Aggiungere (in coda) al file un frammento "noto", che servirà a verificarne l'integrità
2. Cifrare il tutto con un algoritmo a chiave simmetrica (es. DES,AES)
3. Applicare al risultato una funzione di hash
4. Effettuare uno XOR tra l'output del passo 3 e la chiave e aggiungere il risultato ("sum") al messaggio cifrato.
5. Suddividere il risultato finale (messaggio cifrato+"sum") in *k* slice da distribuire tra i nodi

Decifratura

- Per poter recuperare la chiave di cifratura è indispensabile:
  1. Avere TUTTE le slice del messaggio originale, così da poter
  2. Ricalcolare l'hash e porlo in XOR col valore "sum" ($K_r=H_r \oplus (H_t \oplus K)$

  ✓ CONFIDENZIALITA'

- Conferma definitiva: nel messaggio decifrato è presente il *"canarino"*

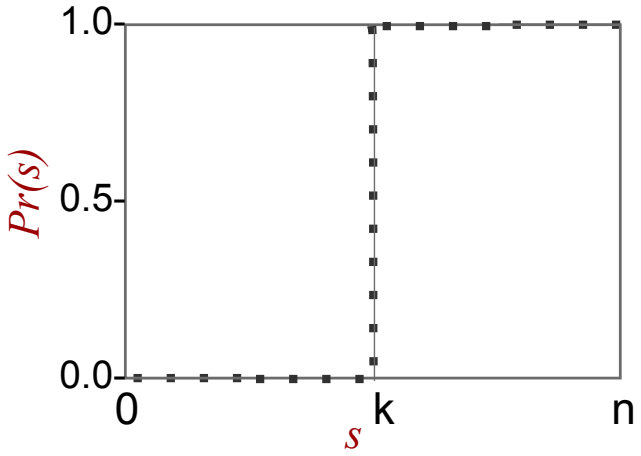
  ✓ INTEGRITA'

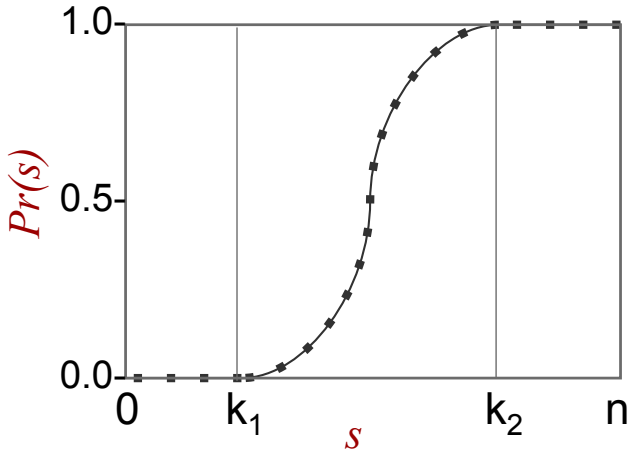
2. Codifica a correzione d'errore

- Obiettivo: migliorare la disponibilità del sistema.
- Strumento: codice a correzione d'errore
- Principio di funzionamento: introdurre nel messaggio ridondanza, che si traduce in *r* slice aggiuntive, per un totale di $n=k+r$

2. Codifica a correzione d'errore

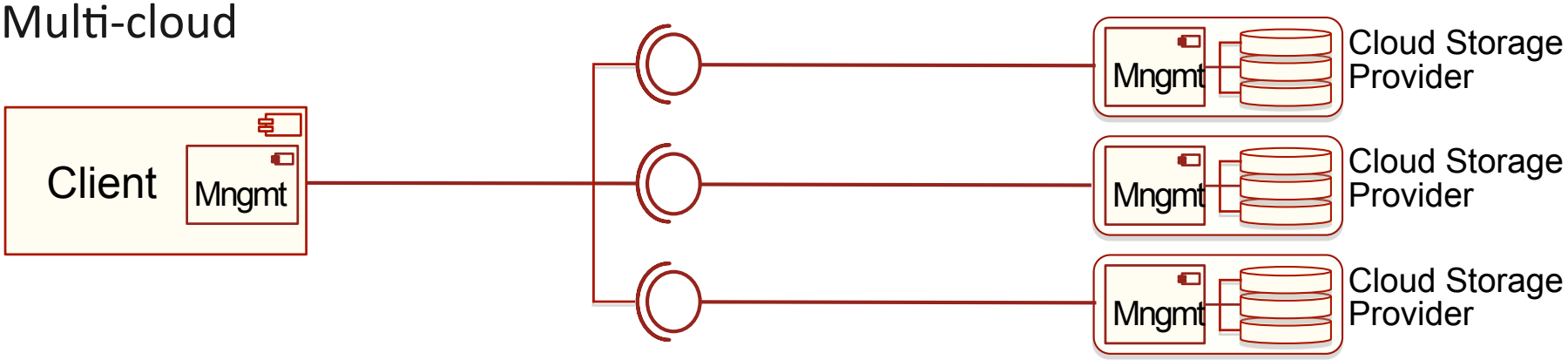
Codifica RS	**Codifica LT**
Basta recuperare *k* slice qualsiasi sul totale (*n*) per essere certi di ricostruire il messaggio	Detto ν il numero delle slice recuperate: - $\nu \in [0,k_1) \Rightarrow P_{ric}=0$ - $\nu \in [k_1, k_2) \Rightarrow P_{ric} \in (0,1)$ - $\nu \in [k_2,n] \Rightarrow P_{ric}=1$



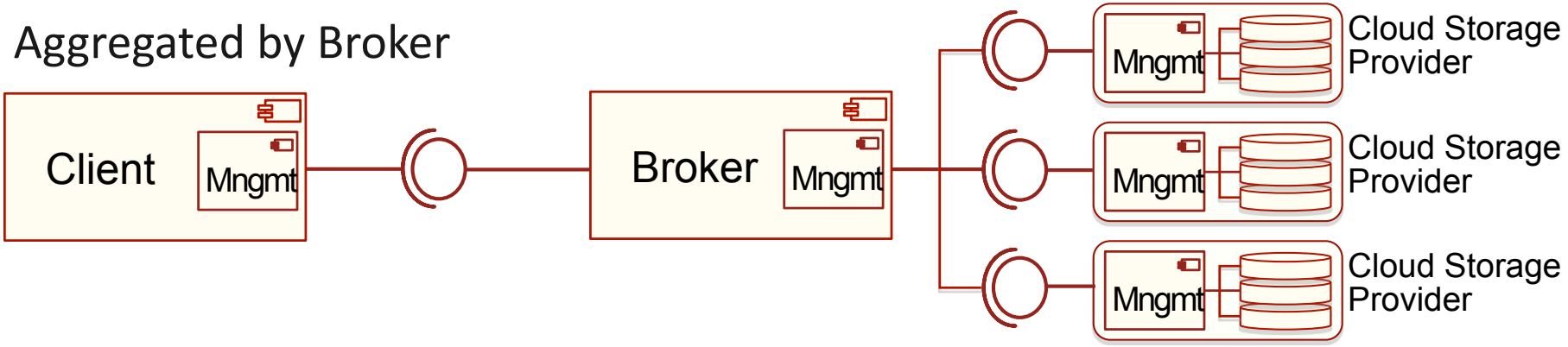


3. Interconnected Cloud Storage

- Multi-cloud



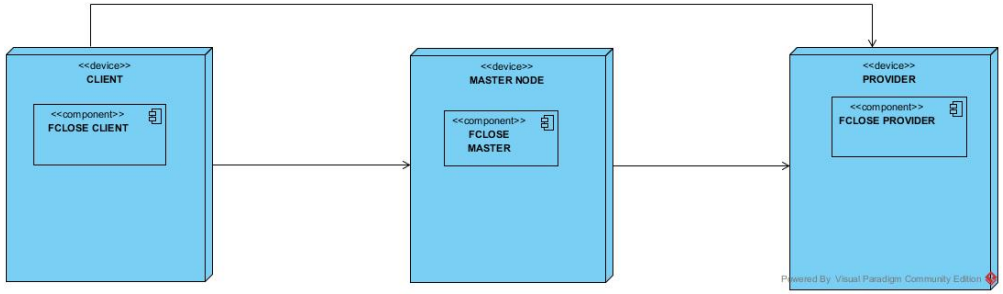
- Aggregated by Broker



Modello Write-Once Read-Many

PROGETTAZIONE

Diagrammi di deployment



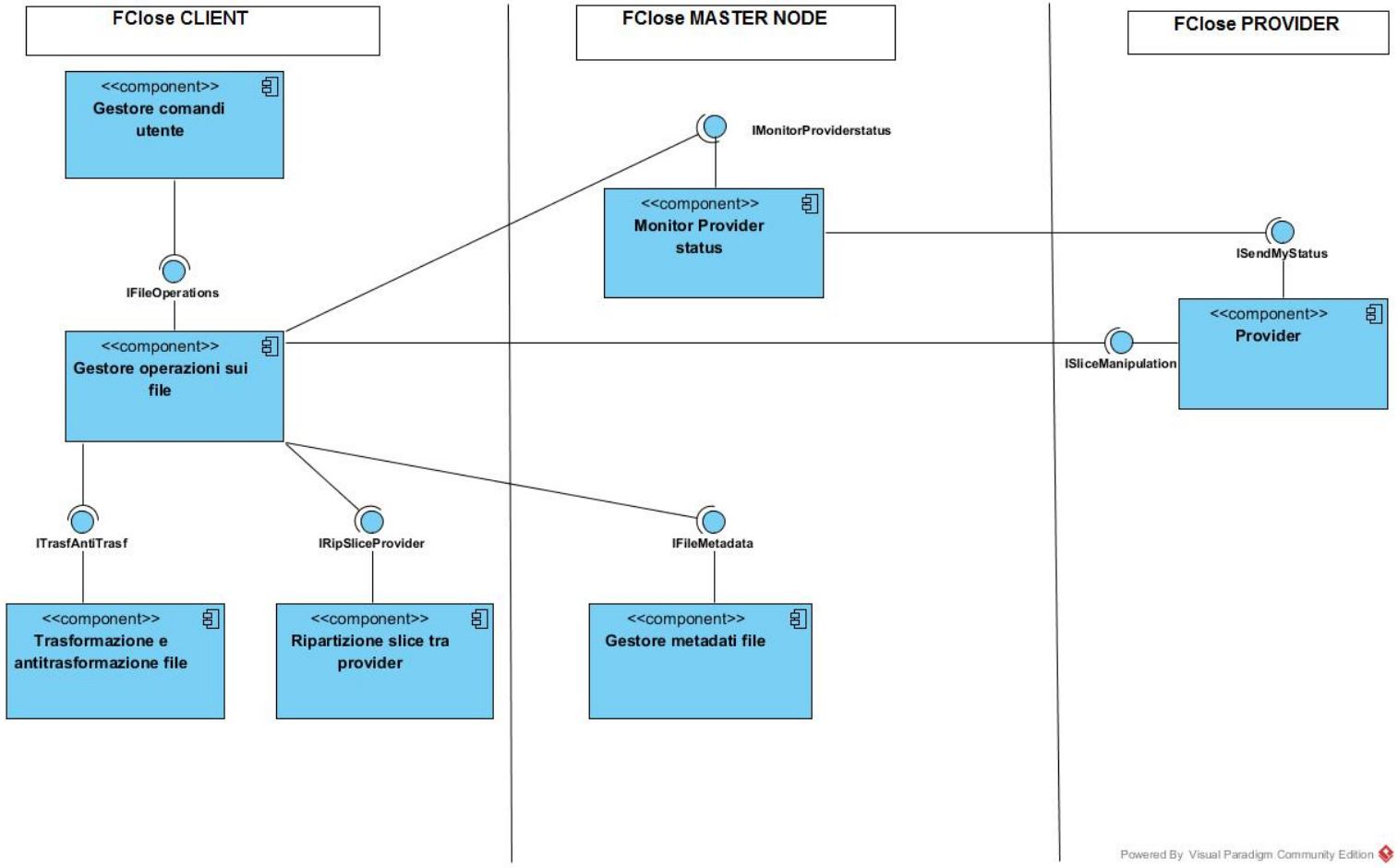
- ✓ Maggiore confidenzialità (nessun server ausiliario)
- × Tanti master-node quanti i client

Difficoltà di allineamento tra le copie dei metadati dei file.

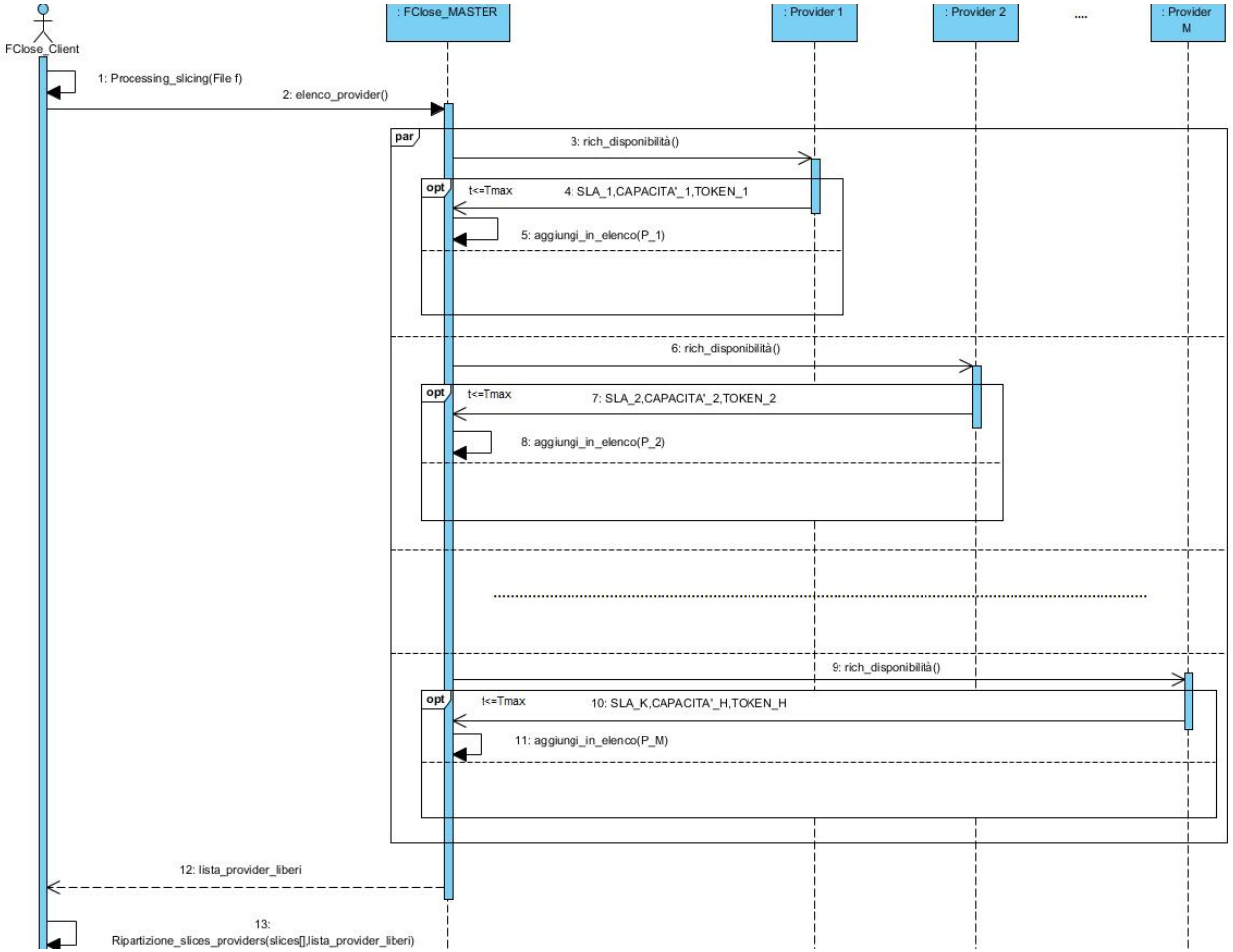
- ✓ No problemi di mantenimento della consistenza
- × Master-node è un single-point of failure

Necessità di un master-node secondario

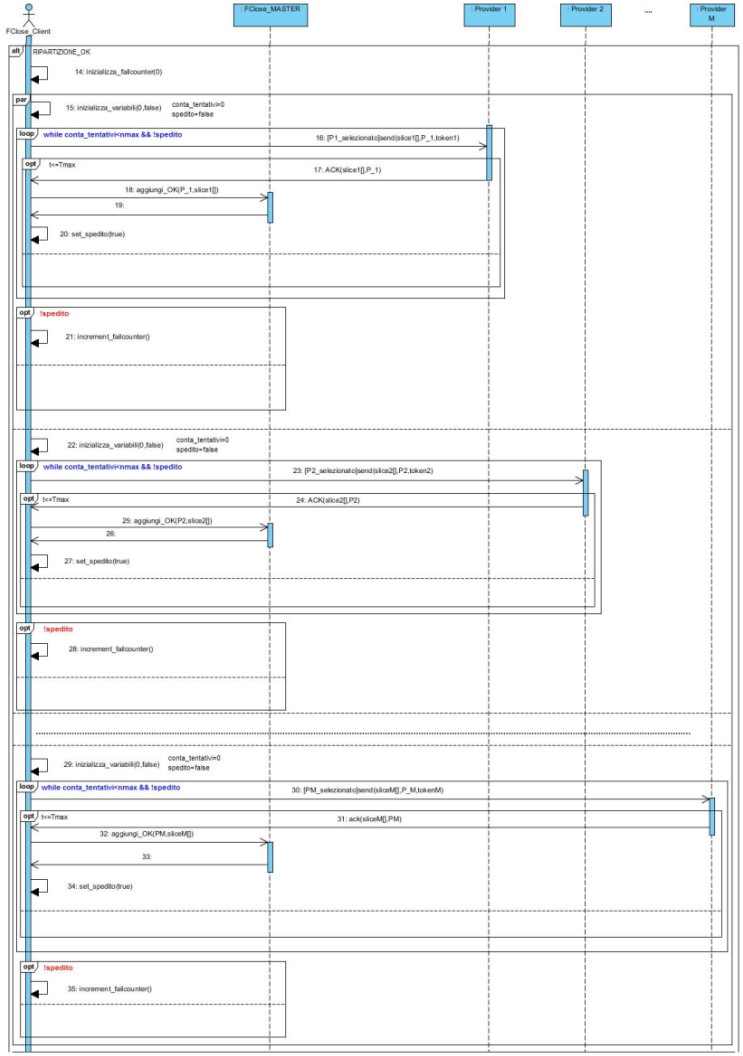
Diagramma dei componenti



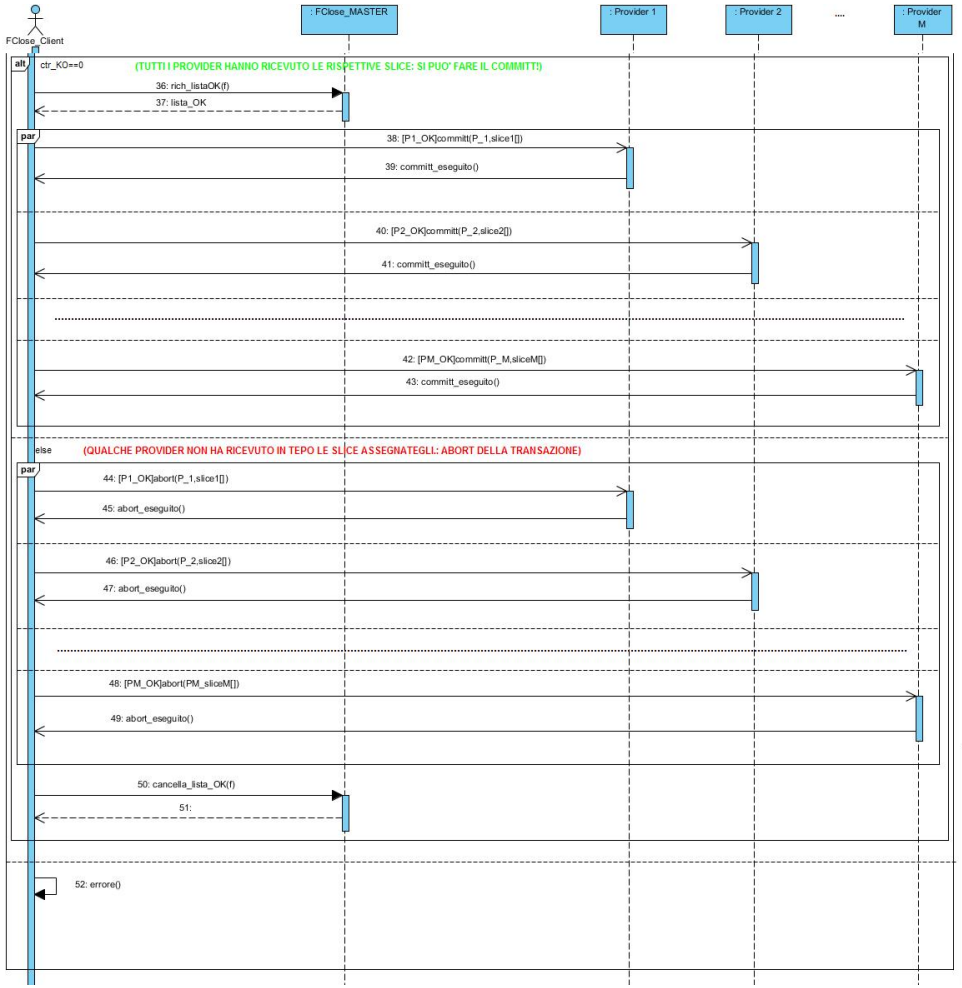
Write – Diagramma delle sequenze (1)



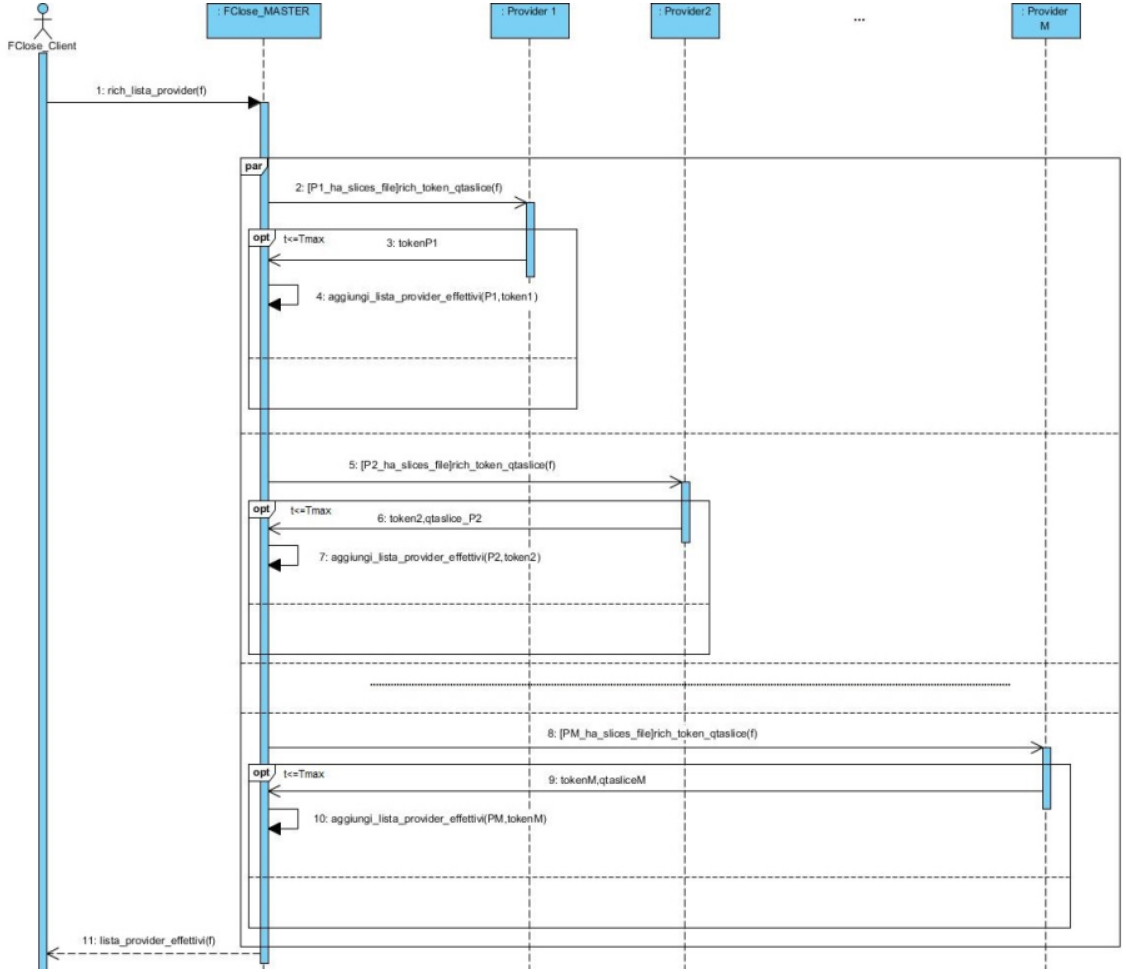
Write – Diagramma delle sequenze (2)



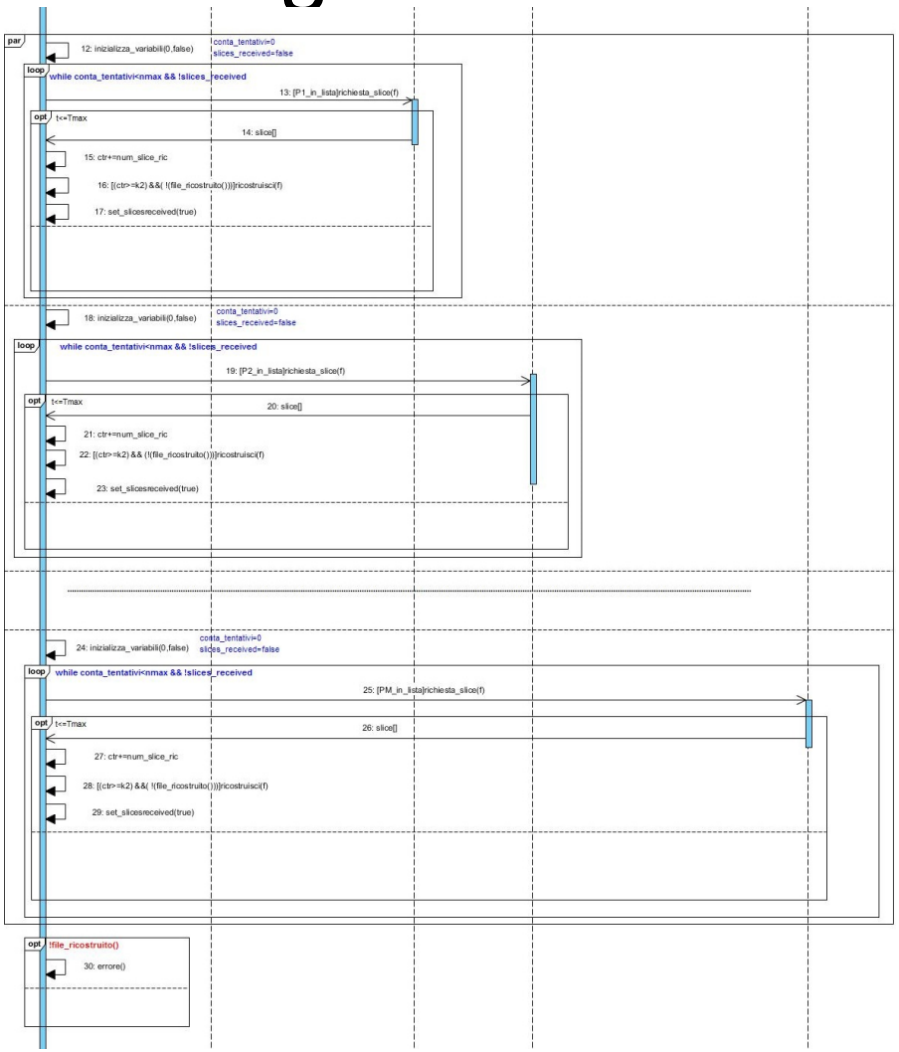
Write – Diagramma delle sequenze (3)



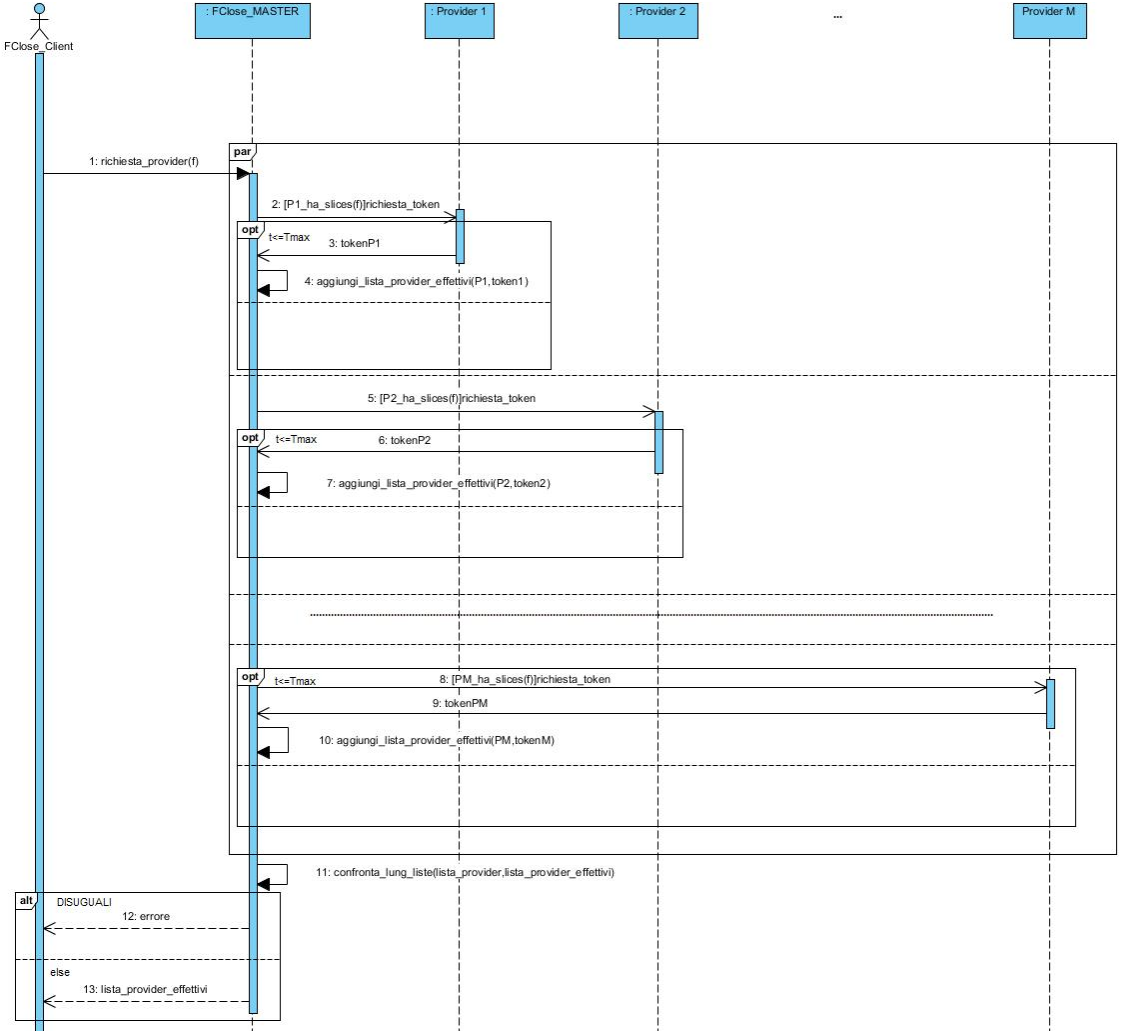
Read – Diagramma delle sequenze



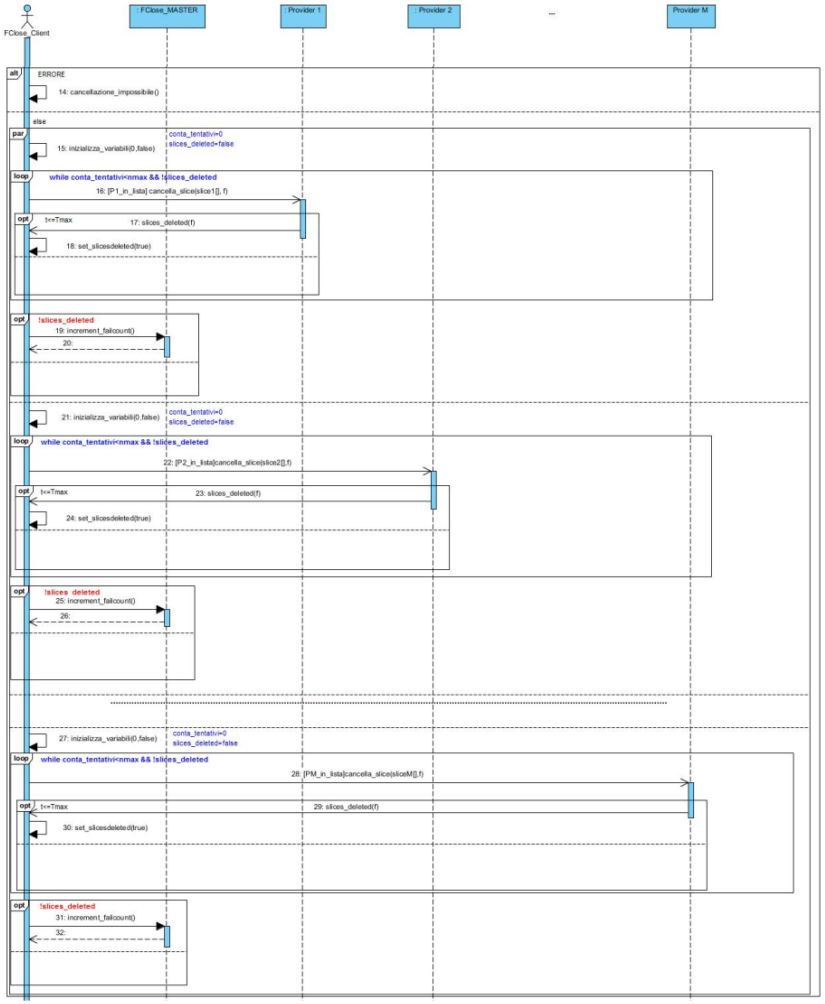
Read – Diagramma delle sequenze



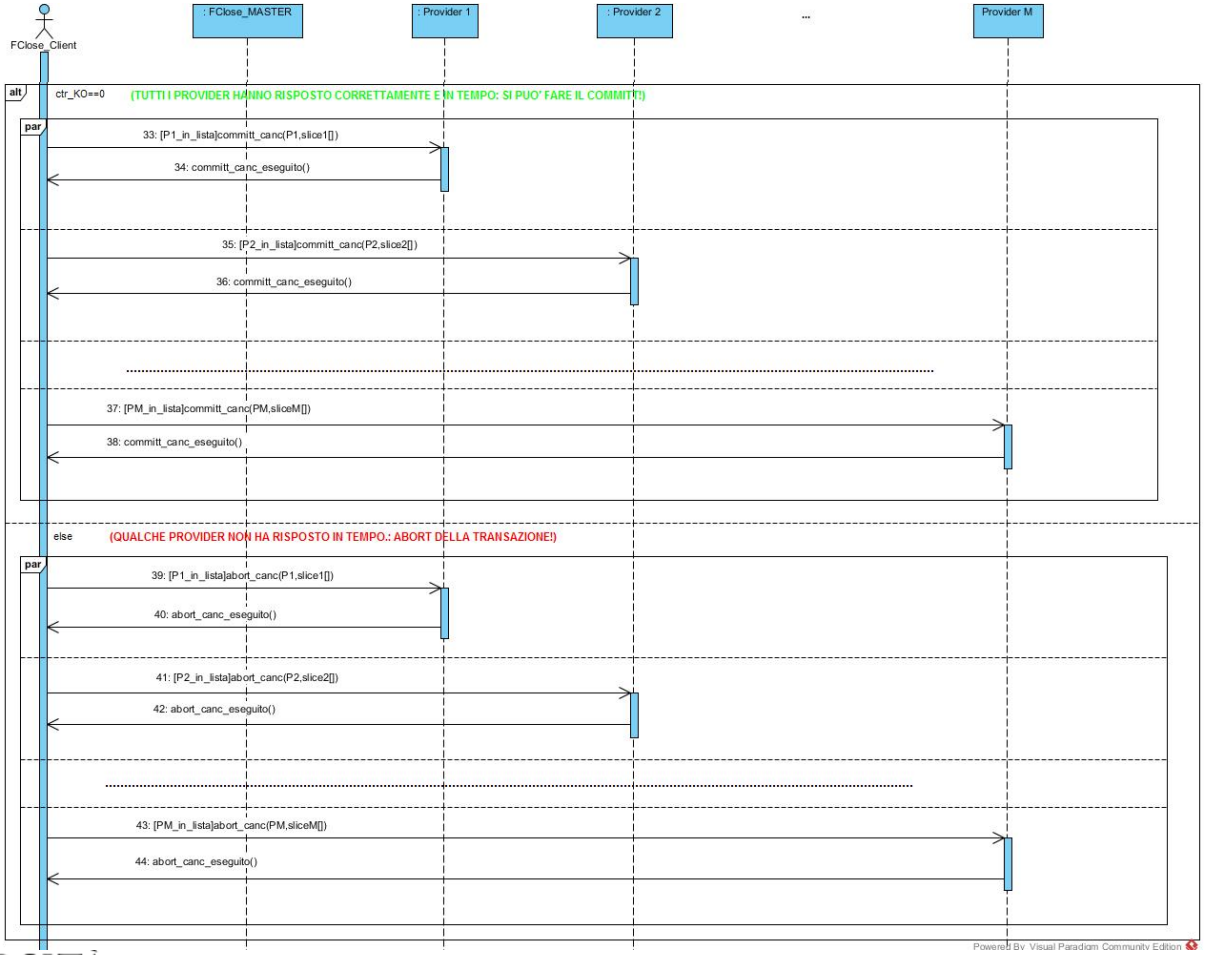
Delete – Diagramma delle sequenze (1)



Delete – Diagramma delle sequenze (2)



Delete – Diagramma delle sequenze (3)



VERIFICA

Verifica formale del sistema – Obiettivi e metodo

- Obiettivo: stimare il livello di confidenzialità garantito dagli algoritmi di cifratura-codifica-dispersione su indicati ai dati scambiati in un *Interconnected-cloud*, al variare del numero di slice e di provider.

- Metodologia utilizzata: **model checking parametrico probabilistico**:

  - Il sistema viene modellato attraverso un opportuno strumento formale

  - Le proprietà di interesse vengono espresse in un linguaggio formale

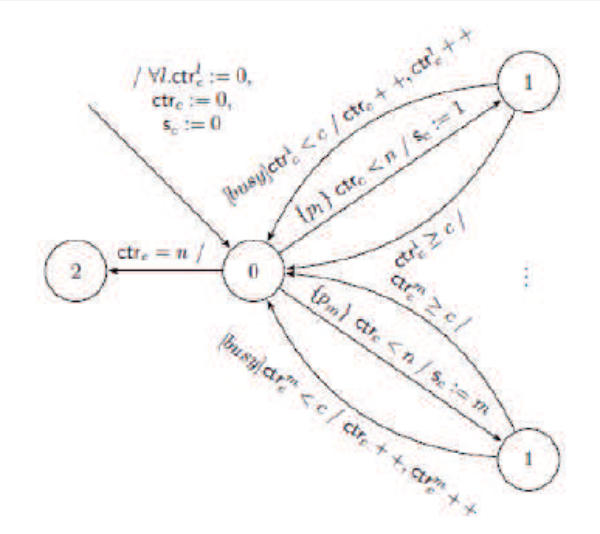
  - Un software (nel caso in esame PRISM) provvede ad elaborare il tutto per verificare con quale probabilità minima e massima (P_{min} e P_{max}) ognuna di tali proprietà risulti vera

Verifica formale del sistema – Metodo (2)

1. Caratterizzazione parametrica del sistema
2. Individuazione di due tipologie di attaccanti:
   - *"Slice-attacker" (SA)*: intercetta le slice in transito sulla rete indipendentemente dal provider a cui sono dirette (*"sniffing"*)
   - *"Provider-attacker" (PA)*: attacca uno o più provider per poi catturare tutte le slice da esso/i ospitate
3. Modellazione del client e di ognuno dei due attaccanti con processi probabilistici
4. Formulazione in logica PCTL* della proprietà di interesse per entrambi i "processi composti" CLIENT||SA e CLIENT||PA:

con quale P_{min} e P_{max} l'intruso in esame riesce a ricostruire fraudolentemente il messaggio inviato dal client ai server dell'Interconnected-cloud

Client: modello probabilistico



Variabili di stato:

- pc_c: posizione corrente nel processo (indicato in ogni nodo)
- sp_c: storage-provider al quale il client invia slice in un certo istante
- $\forall i \in [1,m]\ ctr_c^i$:contatore slice inviate al provider i-esimo
- ctr_c: contatore delle slice complessivamente inviate a tutti i provider

Processo:

Fintantoché tutte le slice non sono state inviate:

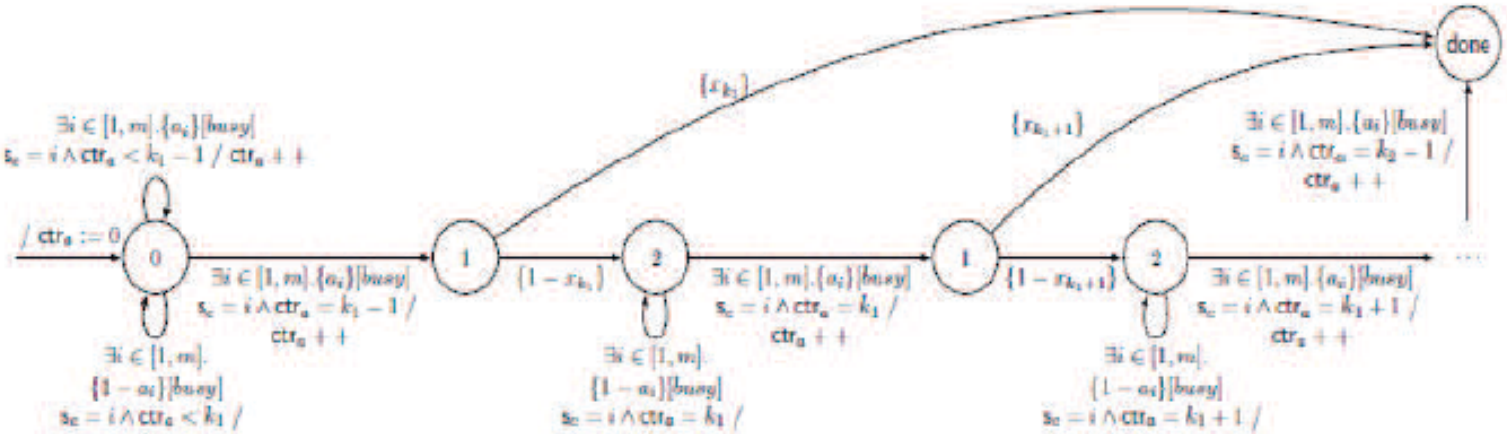
1) scelta con prob. p_i di un provider:

   a) se ha superato la propria capacità massima di archiviazione la slice non viene inviata
   b) altrimenti passaggio in modalità "busy", invia la slice ed incrementa i suoi due contatori di un'unità ciascuno

2) ritorno alla posizione iniziale (pc_c=0)

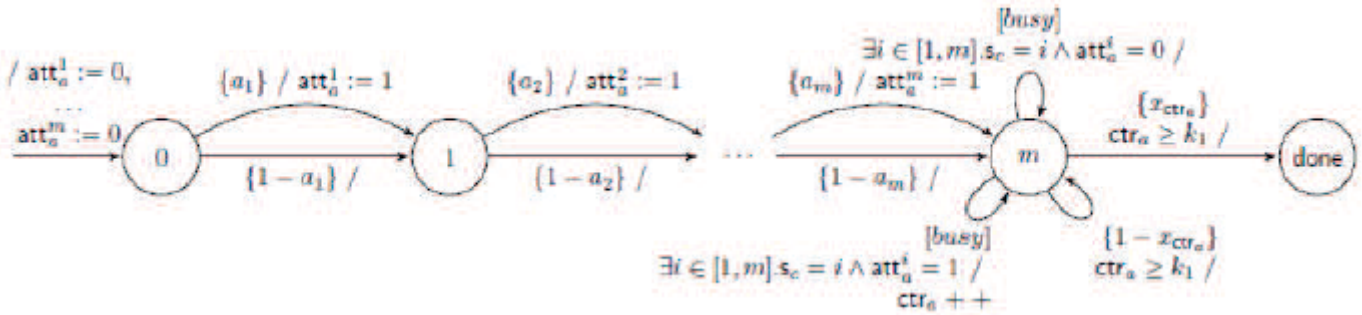
Compito del client concluso quando tutte le slice sono state inviate (pc_c=2, ctr_c=n)

Slice-attacker: modello probabilistico



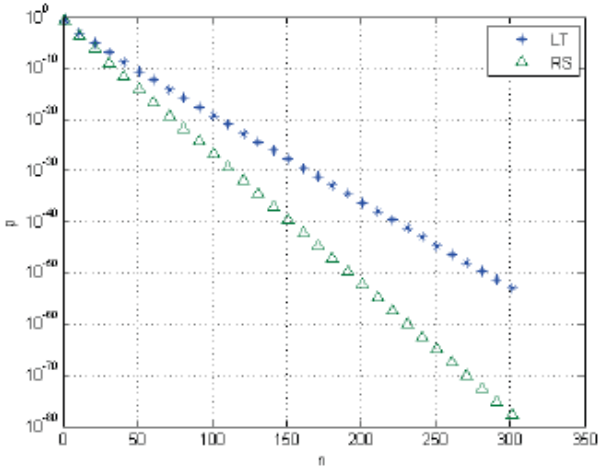
Variabili di stato	**Processo**
- pc_a= posizione corrente del processo (indicata in ogni nodo) - s_c= storage provider al quale il client sta inviando una slice in quel momento - ctr_a=contatore delle slice attualmente intercettate	- Tenta di intercettare le prime k_1 slice - Tenta di ricostruire il messaggio con prob. x_j (j=ctr_a) - Se riesce passa allo stato finale, altrimenti a) tenta di intercettare una nuova slice b) incrementa di uno il proprio contatore c) riprova a ricostruire il messaggio

Provider-attacker: modello probabilistico

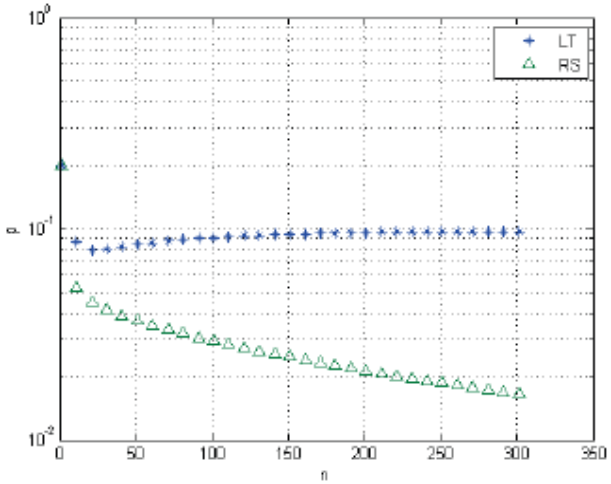


Variabili di stato	**Processo**
- pc_a= posizione corrente del processo (indicata in ogni nodo) - s_c= storage provider al quale il client sta inviando una slice in quel momento - ctr_a=contatore delle slice attualmente intercettate - $\forall i \in [1,m]\ att_a^i$: se settato ad 1 indica che l'attacco al provider i-esimo ha avuto successo, se settato a 0 insuccesso	- Prima fase: tentativo di attacco ad ogni provider, culmina nell'arrivo al nodo "*m*" - Tutte le slice inviate a provider già attaccati vengono intercettate (con incremento di volta in volta di ctr_a). - Allorché $ctr_a \geq k_1$ può tentare di ricostruire il messaggio: se riesce (prob. x_{ctr_a}) giunge allo stato finale ("*done*") altrimenti (prob. 1-x_{ctr_a}) riprende ad intercettare slice alternando con nuovi tentativi di ricostruzione.

Verifica formale del sistema – Risultati



AONT-RS e AONT-LT vs Slice-attacker



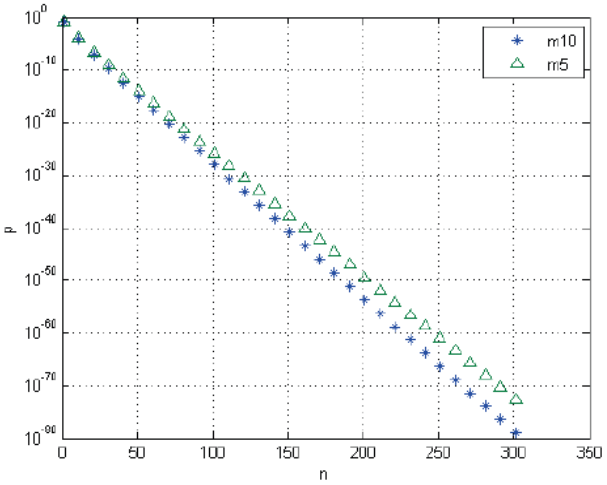
AONT-RS e AONT-LT vs Provider-attacker

Parametri impostati:

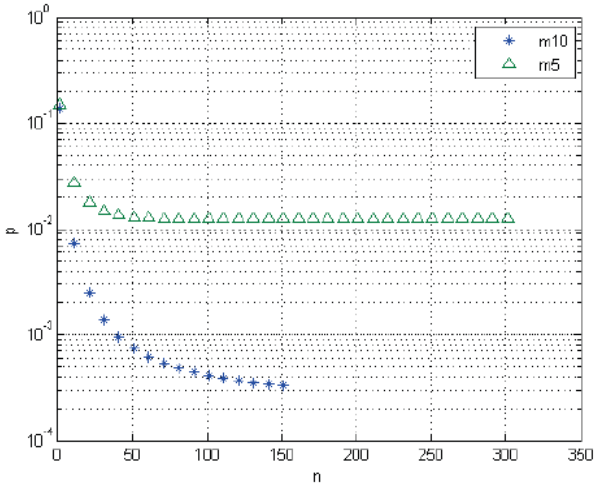
- **num. provider *(m)*=3**
- **prob. attacco: a_1=0.1 a_2=0.2 a_3=0.3**
- **soglie codice LT: k_1=0.6*n k_2=0.8*n**
- **Soglia codice RS: *k*=0.7**n***

- Codifica RS migliore della LT ma difficilmente scalabile rispetto ad *m*
- Migliore resistenza allo Slice-attacker rispetto al Provider-attacker

Verifica formale del sistema – Risultati (2)



AONT-LT vs Slice-attacker



AONT-LT vs Provider-attacker

Parametri impostati:
- **num. provider *(m): m_1=5, m_2=10***
- **prob. attacco a_i uniformi in [0,0.25]**
- **soglie codice LT: k_1=0.6*n k_2=0.8*n**

- Migliore resistenza allo Slice-attacker rispetto al Provider-attacker!
- Nel secondo grafico andamento si stabilizza al variare di *n*

REALIZZAZIONE

Implementazione prototipo

- Linguaggio utilizzato: JAVA
- Collocazione del protocollo nello stack ISO-OSI: livello di trasporto

- Astrazione software: socket TCP
- Librerie JAVA: java.net.Socket, java.net.ServerSocket

Conclusioni

- ✓Progettati protocolli e architettura per scrittura, lettura e cancellazione di file su un *Interconnected-cloud* previa applicazione di algoritmo AONT-RS/AONT-LT.

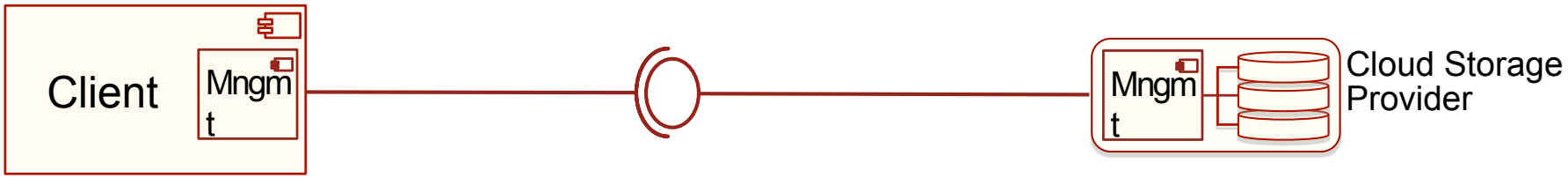
- ✓Progettazione e validazione di metodologia per la verifica sperimentale del sistema

- Implementazione di un prototipo avviata e tutt'ora in corso

Grazie per l'attenzione!

Stato dell'arte

- Single-cloud storage

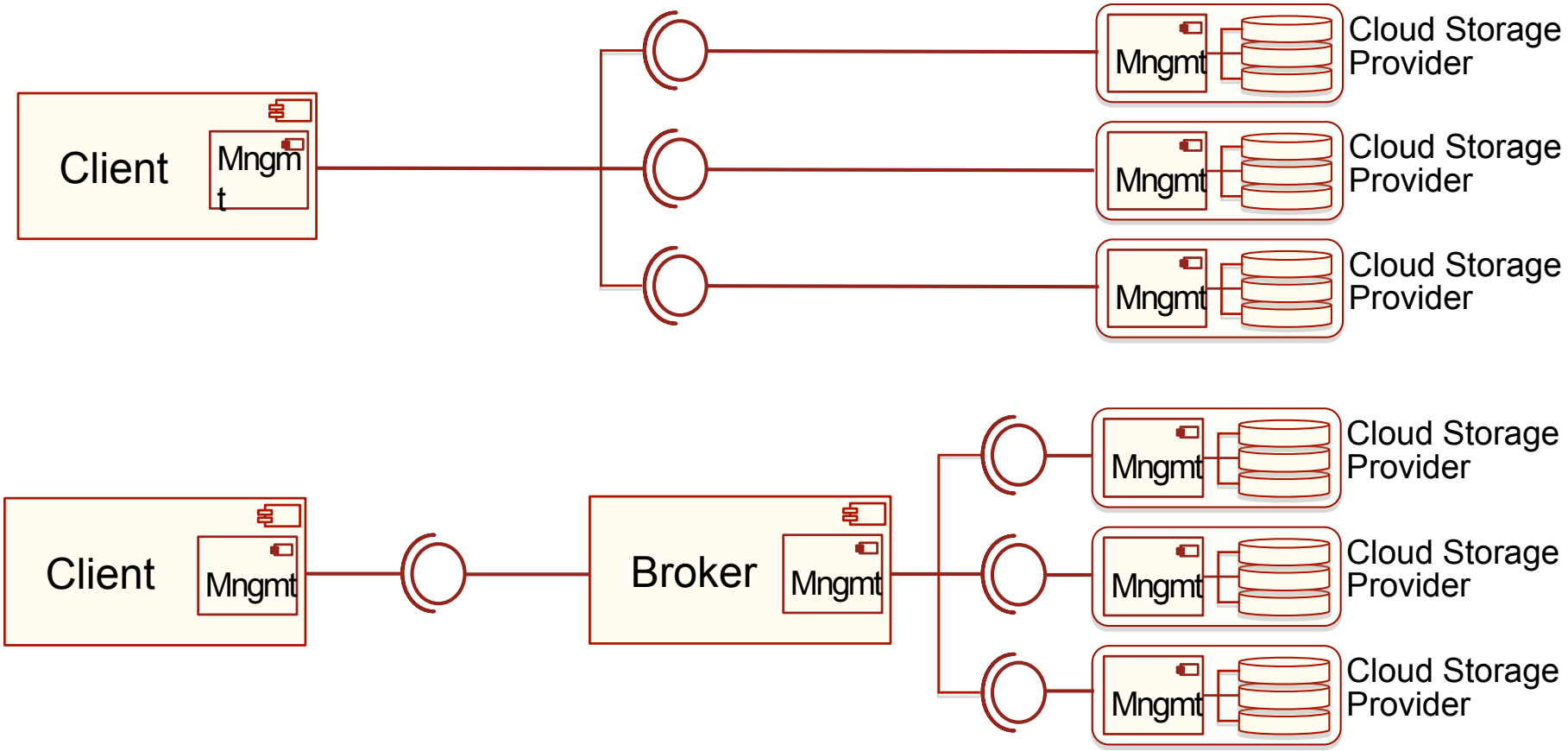


- Criticità:

1. Improvvisi picchi di richiesta di risorse possono sovraccaricare l'infrastruttura con rischio di penalizzazioni della QoS.
2. Esiste sempre il rischio di avarie che interrompano l'erogazione dei servizi.
3. Impossibilità di condividere informazioni o servizi tra organizzazioni diverse
4. Difficoltà di garantire adeguati tempi di risposta a tutti gli utenti.

Stato dell'arte (2)

- Interconnected-cloud storage (Multi-cloud & Aggregated by Broker)



Stato dell'arte (3)

- Interconnected-cloud storage – vantaggi:

  ✓ Migliore gestione dei picchi di carico di lavoro: un cloud sovraccarico può "prendere in prestito" risorse di calcolo da altri ad esso interconnessi ➡ non è più necessario sovradimensionare "a priori" con costi di sottoutilizzazione.

  ✓ Miglioramento dell'affidabilità complessiva, anche in termini di *fault-tolerance*

  ✓ Possibilità di condivisione di informazioni e servizi tra utenti di diversi cloud

  ✓ Più bassi tempi di risposta per gli utenti, specie se sparsi geograficamente.

Stato dell'arte (4)

Principi architetturali di riferimento (HDFS-GFS):

- modello write-once-read-many

- applicazione di algoritmi di hashing e/o codifica e/o cifratura sui file

- divisione dei file in *chunk* distribuiti nei vari nodi del sistema

- un nodo riservato a tenere traccia dei metadati dei file:

  - struttura gerarchica dello spazio dei nomi
  - corrispondenza tra file e blocchi che lo compongono
  - distribuzione di essi negli altri nodi

Cifratura tradizionale ->AONT

- Cifratura a chiave simmetrica: efficiente ma…

  come trasmettere la chiave in modo sicuro in un'Interconnected-cloud?

- Soluzione: All Or Nothing Transform:

PRINCIPI:

1. Utilizzare un algoritmo di cifratura simmetrica ma "mescolare" la chiave al messaggio cifrato
2. Suddividere il tutto in *n* slice da distribuire tra i vari nodi

Per ricostruire il messaggio è necessario recuperare TUTTE le slice!