

Verifica Automatica e Formale di Protocolli di Sicurezza descritti attraverso Modelli UML

Candidato:

Alessandro Busatto

VR432075

Relatore:

Prof.ssa Mila Dalla Preda

Correlatore:

Marco Rocchetto

Indice

1. Analisi dello stato dell'arte
2. Presentazione Toolchain
3. Modellazione UML
4. Modello attaccante Dolev-Yao
5. ProVerif e Verifpal
6. Presentazione Tool di conversione
7. Casi d'uso: Needham-Schroeder Symmetric Key, ARP, RSA
8. Conclusioni

Stato dell'arte



Anni	Definizione	Legenda
1970	infosec = CIA	Confidenzialità, Integrità, Disponibilità
1980	infosec += (Au,nR)	Autenticazione e non-Ripudio
1990	infosec += CSpec	Correttezza delle specifiche
2000	infosec += RITE	Responsabilità, Integrità delle persone, fiducia, eticità

Nascono il modello simbolico e il modello computazione con l'obiettivo di definire formalmente le proprietà di sicurezza previste dal protocollo e dimostrarne rigorosamente la soddisfazione delle stesse, basandosi sulla matematica formale.

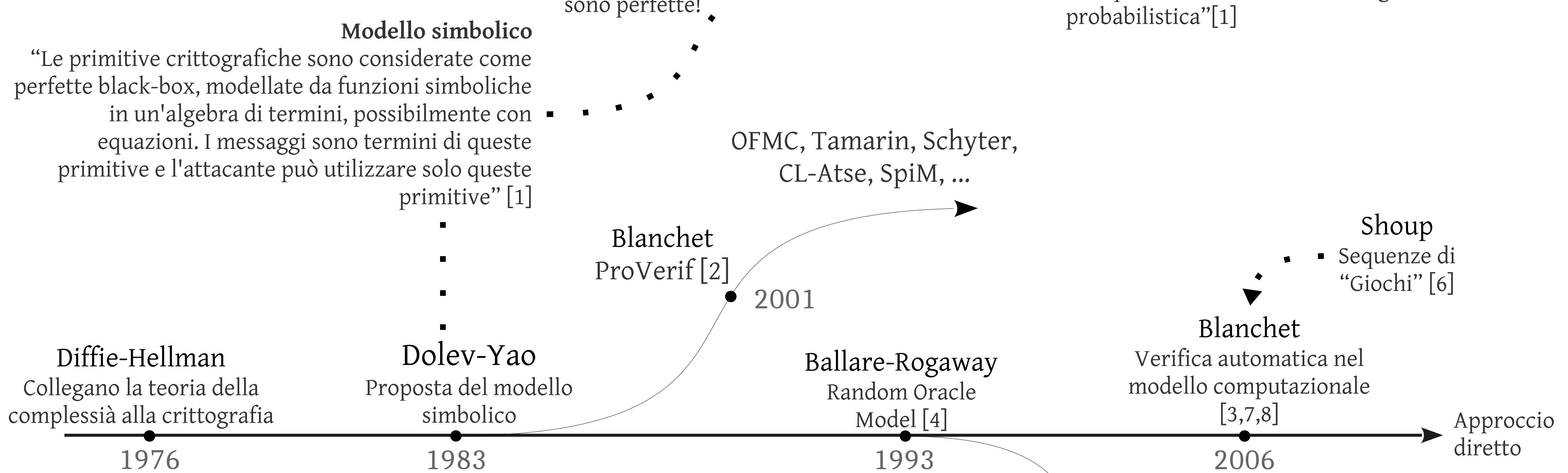
Modello computazionale:

- Messaggi dettagliati a livello di bit e attaccante potente
- Sicurezza valutata rispetto a macchine probabilistiche polinomiali
- Attaccante polinomiale rispetto al Security Parameter (lunghezza della chiave)
- Dimostrazioni lunghe, difficili e soggette ad errori anche con piccoli protocolli

Modello simbolico:

- I messaggi sono termini simbolici
- Visione astratta dell'esecuzione del protocollo
- Le primitive di sicurezza sono black box -> perfect cryptography
- Dimostrazioni semplici

Stato dell'arte



[1] Dolev, Yao. On the Security of Public Key Protocols. 1963

[2] Blanchet. Modeling and Verifying Security Protocols with the applied Pi Calculus and ProVerif. 2016

[3] Blanchet. An Efficient Cryptographic Verifier Based on Prolog Rules. 2001

[4] Ballare, Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. 1995

[5] Abadi, Rogaway. Reconciling Two Views of Cryptography. 2002

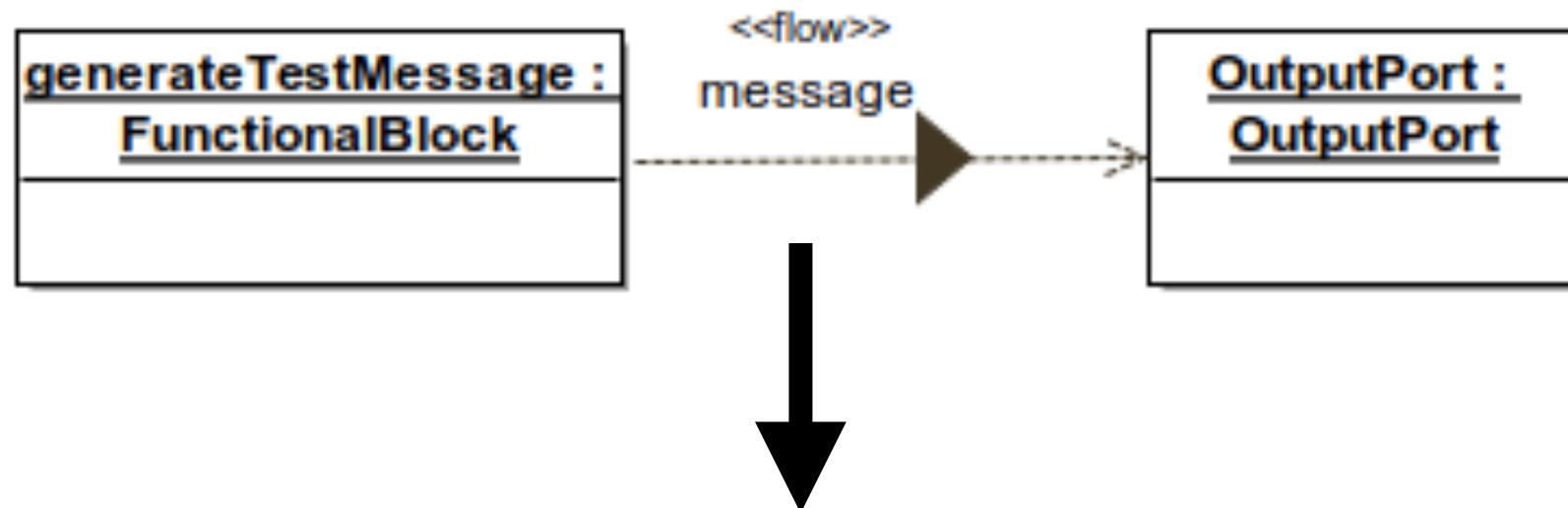
[6] Shoup. Sequences of Games: a Tool for Taming Complexity in Security Proofs. 2004

[7] Blanchet. A Computationally Sound Mechanized Prover for Security Protocols. 2006

[8] Blanchet, Pointcheval. Automated Security Proofs with Sequences of Games. Advances in Cryptology. 2006

Toolchain

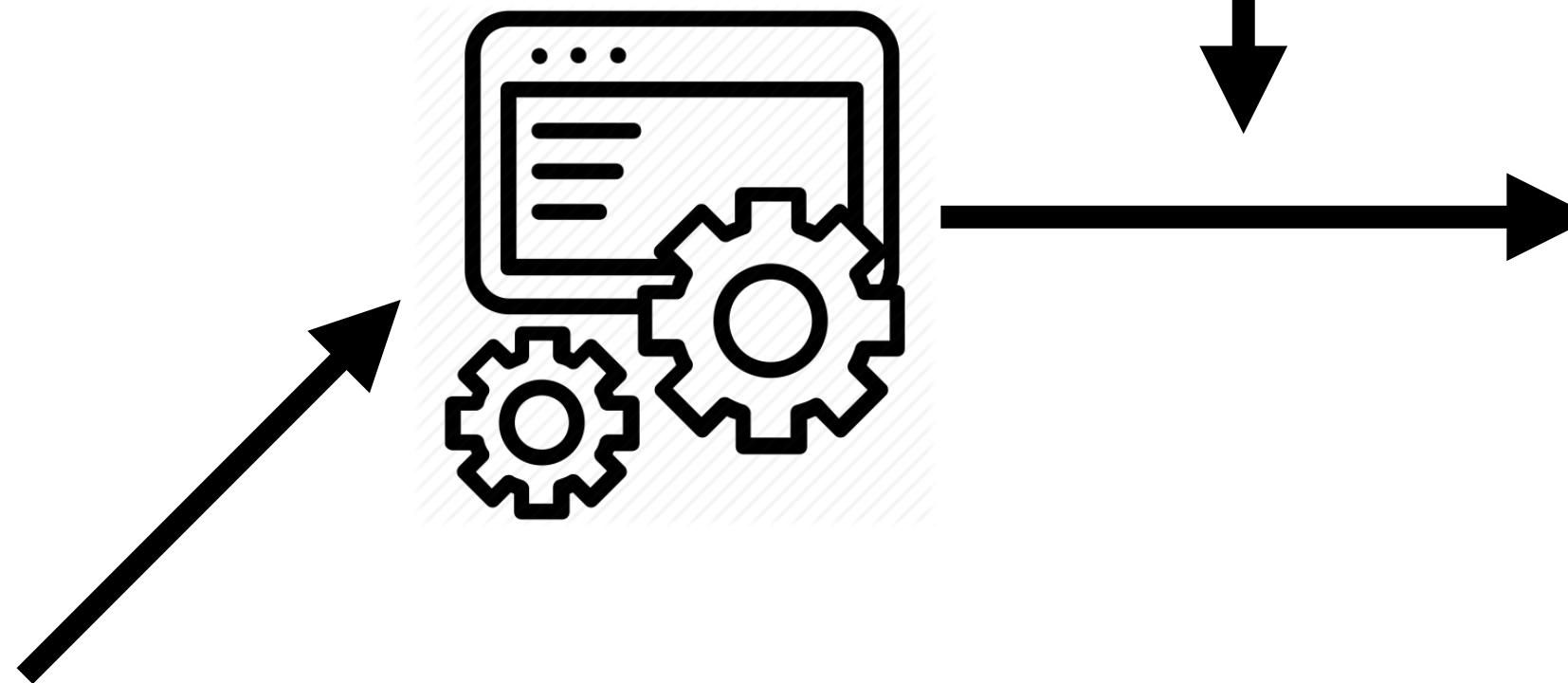
1. Modello UML



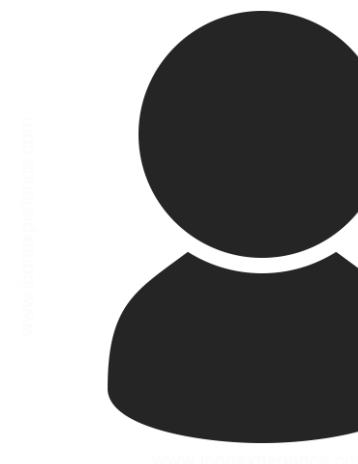
2. XMI

```
<packagedElement
  xmi:type="uml:Package"
  xmi:id="_6KYCwF_CEuUS4KSsiMLRg"
  name="ExampleThesis"
>
<packagedElement
  xmi:type="uml:InstanceSpecification"
  xmi:id="_6KYCwV_CEuUS4KSsiMLRg"
  name="generateTestMessage"
  classifier="_6KWNiV_CEuUS4KSsiMLRg"
/>
<packagedElement
  xmi:type="uml:InstanceSpecification"
  xmi:id="_6KYCwl_CEuUS4KSsiMLRg"
  name="OutPort"
  classifier="_6KWNqV_CEuUS4KSsiMLRg"
/>
<packagedElement
  xmi:type="uml:InformationFlow"
  xmi:id="_6KYCw1_CEuUS4KSsiMLRg"
  name="message"
  informationSource="_6KYCwV_CEuUS4KSsiMLRg"
  informationTarget="_6KYCwl_CEuUS4KSsiMLRg"
/>
</packagedElement>
```

3. Tool di conversione



4. Inserimento specifiche di sicurezza



5. Modello VerifPal

```
attacker [active]
principal A [
  knows private message
]
principal B []
A -> B : message
queries [
  confidentiality? message
]
```

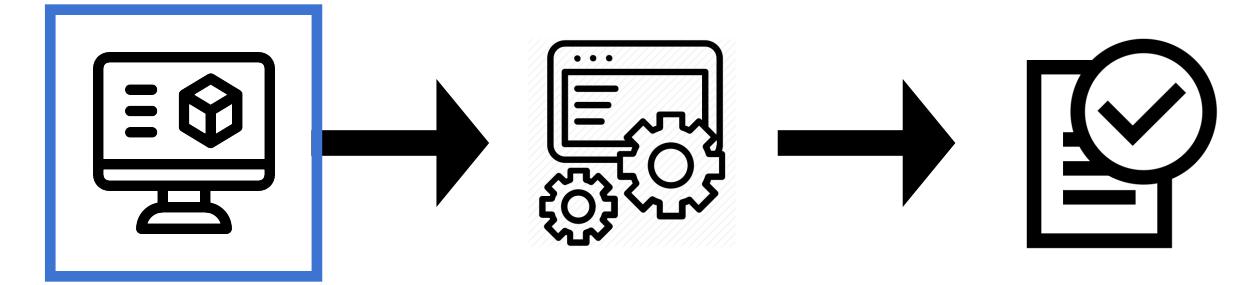


6. Risultati:

confidentiality? Message → **violata dall'Attaccante**.

Modellazione UML

Modellazione



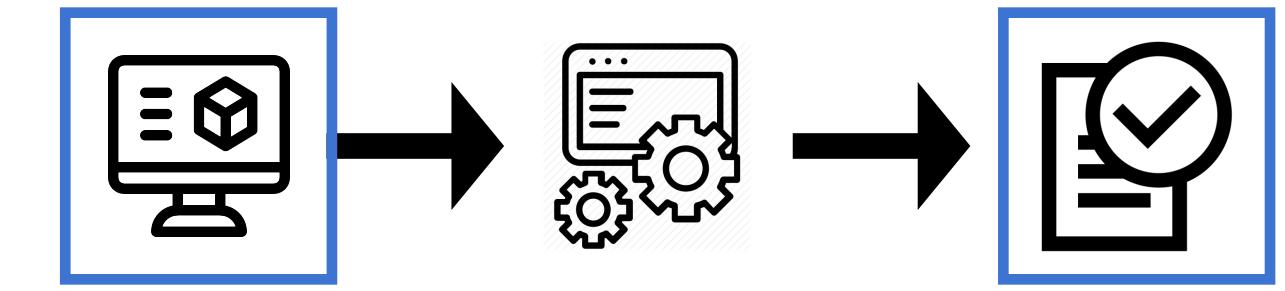
Esportazione XMI



```
<packagedElement  
xmi:type="uml:Package"  
xmi:id="_6KYCwF_CEuUS4KSsiMLRg"  
name="ExampleThesis"  
>  
<packagedElement  
xmi:type="uml:InstanceSpecification"  
xmi:id="_6KYCwV_CEuUS4KSsiMLRg"  
name="generateTestMessage"  
classifier="_6KWNiV_CEuUS4KSsiMLRg"  
/>  
<packagedElement  
xmi:type="uml:InstanceSpecification"  
xmi:id="_6KYCw1_CEuUS4KSsiMLRg"  
name="OutPort"  
classifier="_6KWNqV_CEuUS4KSsiMLRg"  
/>  
<packagedElement  
xmi:type="uml:InformationFlow"  
xmi:id="_6KYCw1_CEuUS4KSsiMLRg"  
name="message"  
informationSource="_6KYCwV_CEuUS4KSsiMLRg"  
informationTarget="_6KYCw1_CEuUS4KSsiMLRg"  
/>  
</packagedElement>
```

Modello di attaccante Dolev-Yao

Modellazione



È in grado di:

- intercettare, osservare ed eliminare i messaggi nella rete,
- costruire nuovi messaggi a partire da quelli osservati e iniettarli nella rete,
- disassemblare i messaggi non cifrati nelle parti che li compongono,
- iniettare nuovi messaggi nella rete costruiti con le informazioni di una sessione precedente del protocollo,
- decifrare messaggi solo se a conoscenza della chiave di cifratura

Primitive

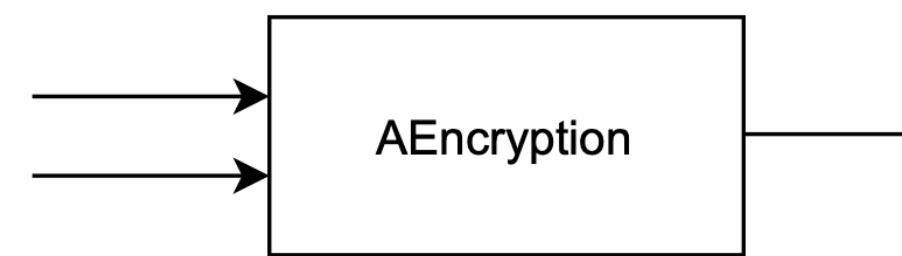
$$\frac{M_1 \in DY(IK) \quad M_2 \in DY(IK)}{[M_1, M_2] \in DY(IK)} \quad G_{concat}$$



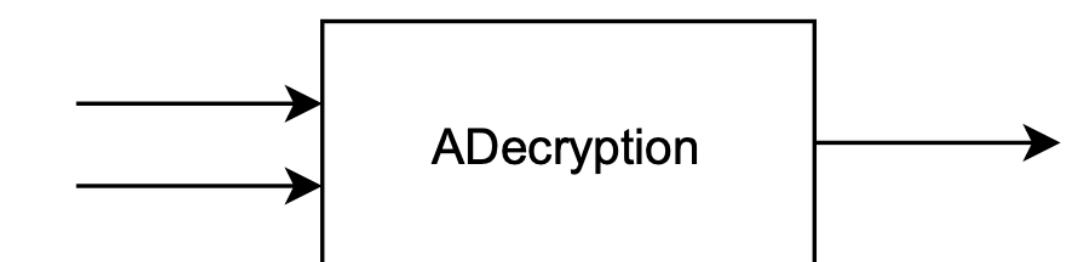
$$\frac{[M_1, M_2] \in DY(IK)}{M_i \in DY(IK)} \quad A_{concat_i}$$



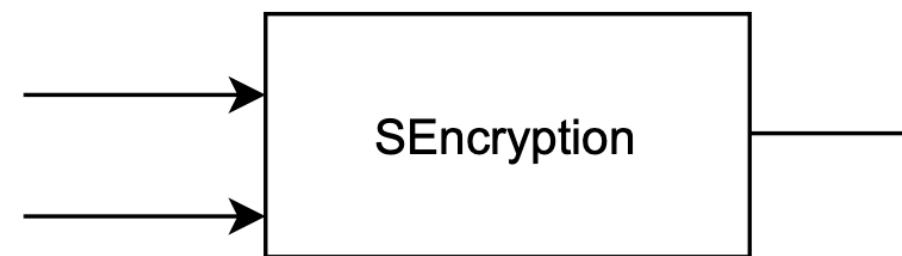
$$\frac{M_1 \in DY(IK) \quad M_2 \in DY(IK)}{\{M_1\}_{M_2} \in DY(IK)} \quad G_{critA}$$



$$\frac{\{M_1\}_{M_2} \in DY(IK) \quad inv(M_2) \in DY(IK)}{M_1 \in DY(IK)} \quad A_{critA}$$



$$\frac{M_1 \in DY(IK) \quad M_2 \in DY(IK)}{\{|M_1|\}_{M_2} \in DY(IK)} \quad G_{critS}$$

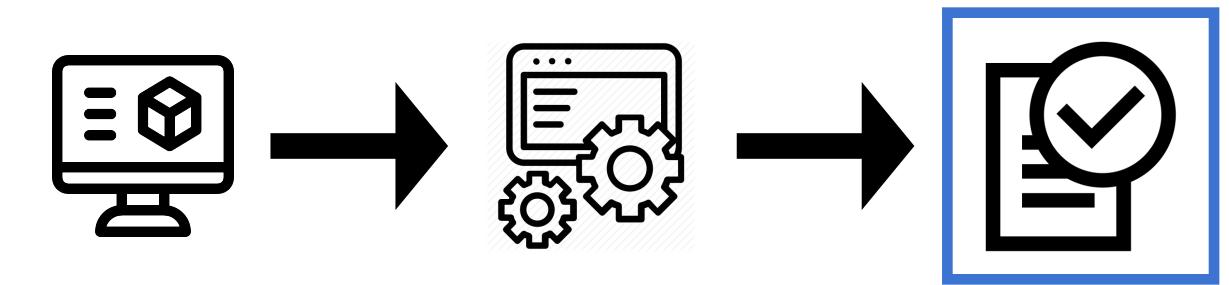


$$\frac{\{|M_1|\}_{M_2} \in DY(IK) \quad M_2 \in DY(IK)}{M_1 \in DY(IK)} \quad A_{critS}$$



$$\frac{\{M_1\}_{inv(M_2)} \in DY(IK) \quad M_2 \in DY(IK)}{M_1 \in DY(IK)} \quad A_{critA}^{-1}$$





Tool di verifica automatica basati sul modello simbolico

ProVerif

```

free c : channel .

free m : bitstring [ private ] .

query attacker(m) .

let Initiator (x : bitstring) = out (c, x) ; 0 .
let Recipient = in (c, m:bitstring); 0 .

process Initiator(m) | Recipient
  
```

VerifPal

```

attacker[active]

principal Initiator[
    knows private m
]

principal Recipient[]

Initiator -> Recipient : m

queries [
    confidentiality? m
]
  
```

Osservazioni primitive crittografiche:

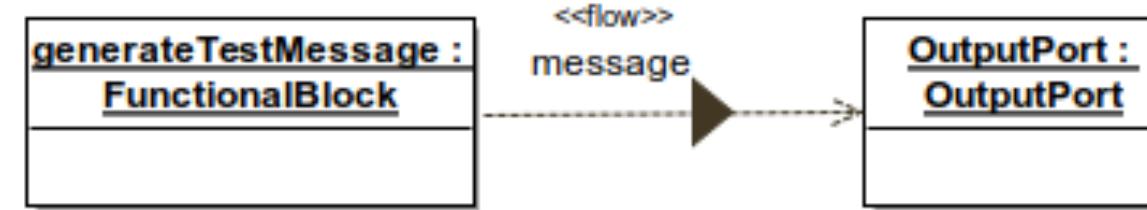
```

type key .
fun senc(bitstring, key) : bitstring .
reduc forall m: bitstring, k : key ;
  sdec(senc(m,k),k) = m .
  
```

- $\text{ENC}(\text{key}, \text{plaintext})$: ciphertext \rightarrow codifica nella crittografia a chiave simmetrica
- $\text{DEC}(\text{key}, \text{ENC}(\text{key}, \text{plaintext}))$: plaintext \rightarrow decodifica nella crittografia a chiave simmetrica

Tool di conversione

Parte 1



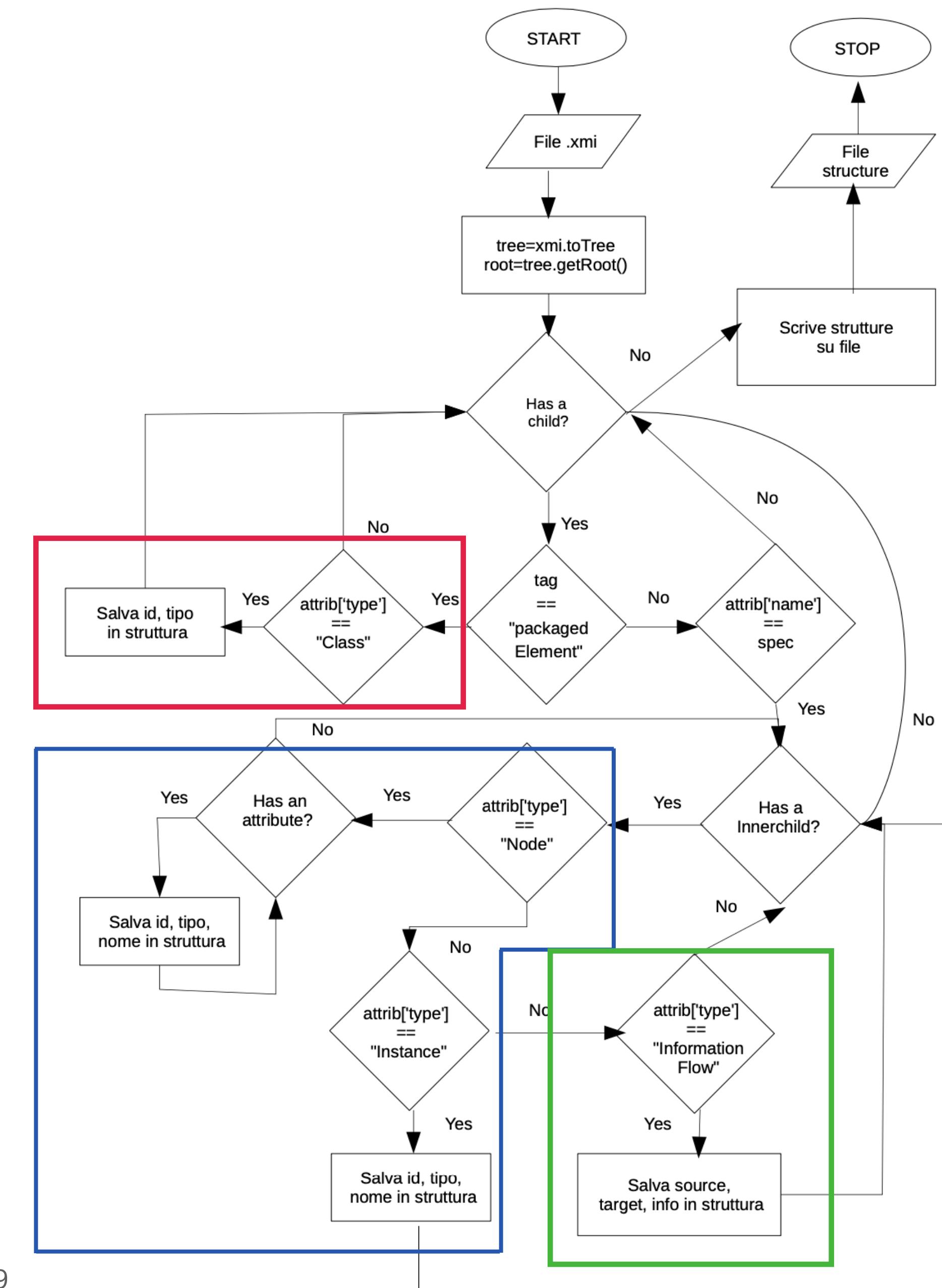
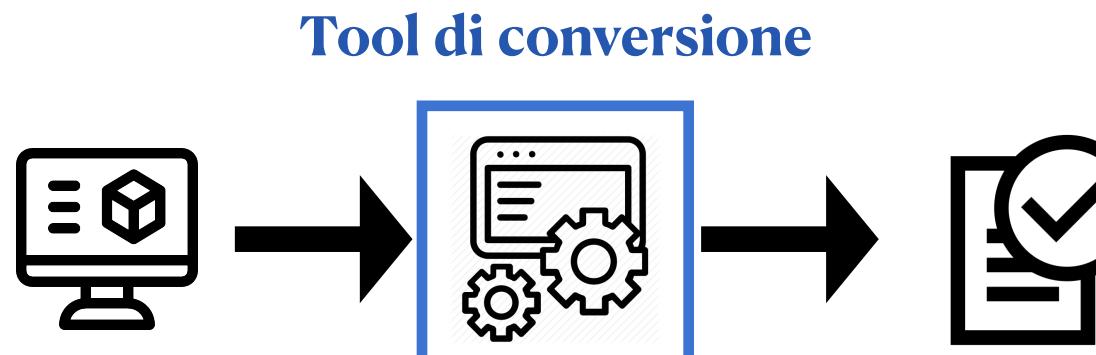
Esempio di output:

Blocks :

```
{
  '_6KYCwV_CEeuUS4KSsiMLRg': { 'name': 'generateTestMessage',
    'type': 'FunctionalBlock'},
  '_6KYCwl_CEeuUS4KSsiMLRg': { 'name': 'OutPort', 'type': 'OutputPort'}
}
```

Flows :

```
{1: { 'information': 'message',
  'source': '_6KYCwV_CEeuUS4KSsiMLRg',
  'target': '_6KYCwl_CEeuUS4KSsiMLRg'}}
```



Tool di conversione

Parte2

Esempio di output:

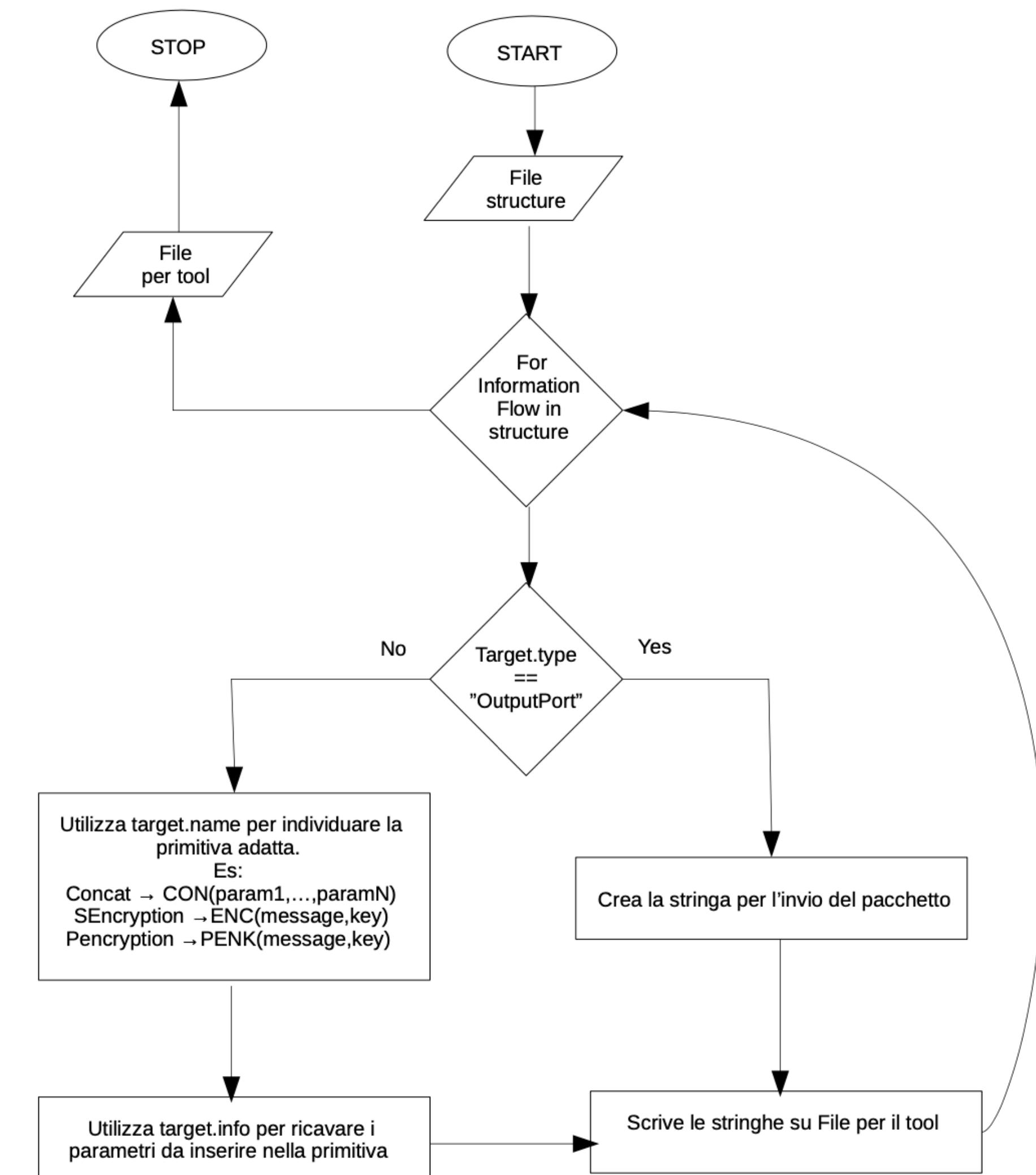
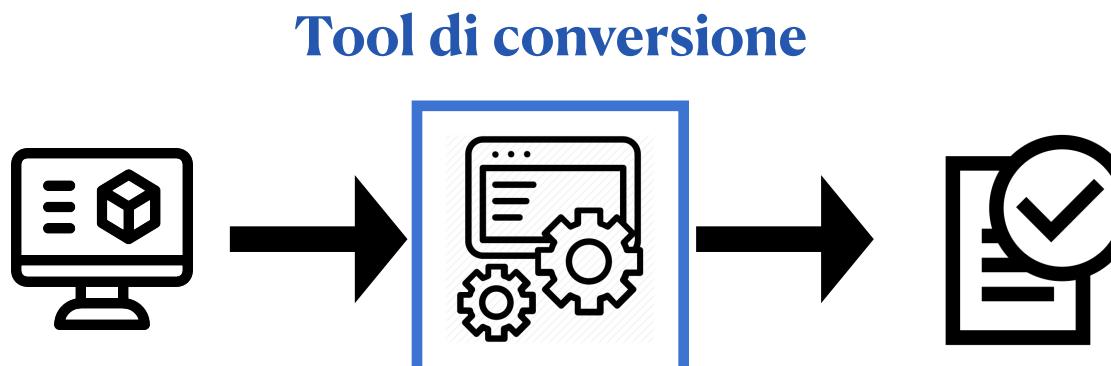
```
attacker [active]//or [passive]
```

```
principal A [  
    knows private message  
]
```

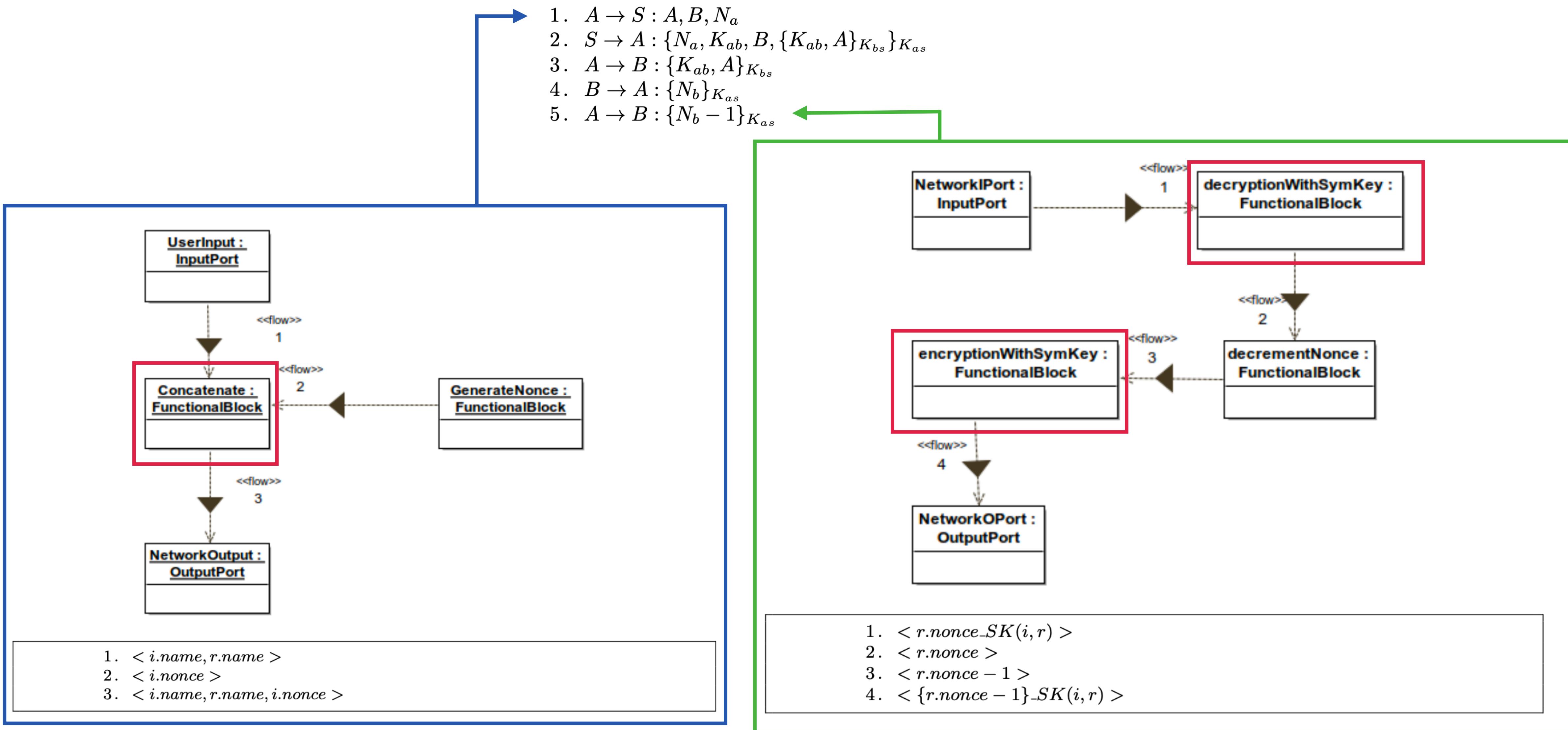
```
principal B []
```

```
A -> B : message
```

```
queries [  
    //Enter what you want to verify  
]
```



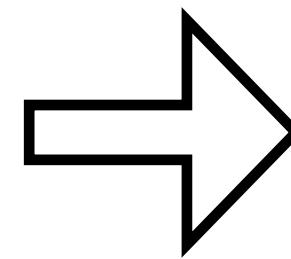
Needham-Schroeder Symmetric Key



Estratto .xmi

```
<ownedAttribute xmi:type="uml:Property" xmi:id="_
_WmudcmlZEeunebkXFN4K9w" name="GenerateSymKey(i,r)" type
=_WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
<ownedAttribute xmi:type="uml:Property" xmi:id="_
_Wmudc2lZEeunebkXFN4K9w" name="GetSharedKey" type=
_WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>

<packagedElement xmi:type="uml:InformationFlow" xmi:id="_
_WmudhmlZEeunebkXFN4K9w" name=<i.name,r.name>;
informationSource=_WmudcGlZEeunebkXFN4K9w"
informationTarget=_Wmudc2lZEeunebkXFN4K9w"/>
```



Dizionario degli oggetti

```
'_Wmudc2lZEeunebkXFN4K9w': { 'name': 'GetSharedKey', 'type': 'FunctionalBlock' },
```

Dizionario delle InformationFlow

```
{1: { 'information': '<i.name,r.name>', 'source': '_WmudcGlZEeunebkXFN4K9w', 'target': '_Wmudc2lZEeunebkXFN4K9w' },
```

Verifica con VerifPal

attacker[active] **Inizializzazione principale**

```
principal Server[
    knows private k_is
    knows private k_rs
]
```

```
principal Initiator[
    knows private i_name
    knows private r_name
    knows private k_is
    generates n_i
]
```

```
principal Recipient[
    knows private r_name
    knows private k_rs
    generates n_r
]
```

Initiator -> Server: i_name, r_name, n_i

Invio di messaggi

```
principal Server[
    generates k_ir
    e_recipient = AEAD_ENC(k_rs, CONCAT(k_ir,
        i_name), nil)
    e_initiator = AEAD_ENC(k_is, CONCAT(n_i, k_ir,
        r_name, e_recipient), nil)
]
```

Server -> Initiator: e_initiator

```
principal Initiator[
    e_initiator_dec = AEAD_DEC(k_is, e_initiator,
        nil)
    n_i_response, k_ir_initiator, r_name_initiator
        , e_recipient_initiator = SPLIT(
            e_initiator_dec)
    - = ASSERT(n_i, n_i_response)
    - = ASSERT(r_name, r_name_initiator)
]
```

Initiator -> Recipient: e_recipient_initiator

```
principal Recipient[
    e_recipient_dec = AEAD_DEC(k_rs,
        e_recipient_initiator, nil)
    k_ir_recipient, i_name_recipient = SPLIT(
        e_recipient_dec)
    e_n_r = AEAD_ENC(k_ir_recipient, n_r, nil)
]
```

Recipient -> Initiator: e_n_r

```
principal Initiator[
    n_r_initiator = AEAD_DEC(k_ir_initiator, e_n_r
        , nil)
    n_r_minus_one = HASH(n_r_initiator)
    e_n_r_minus_one = AEAD_ENC(k_ir_initiator,
        n_r_minus_one, nil)
]
```

Initiator -> Recipient: e_n_r_minus_one

```
principal Recipient[
    n_r_minus_one_recipient = AEAD_DEC(
        k_ir_recipient, e_n_r_minus_one, nil)
    - = ASSERT(n_r_minus_one_recipient, HASH(n_r))
]
```

```
principal Server[
    leaks k_ir
]
```

Leaks della chiave sul server

```
queries[
    confidentiality? k_ir
    confidentiality? n_r_minus_one
    authentication? Initiator -> Recipient:
        e_n_r_minus_one
]
```

Proprietà da verificare

Risultati:

confidentiality? k_ir → **violata dall'Attaccante**.

confidentiality? n_r_minus_one → **violata dall'Attaccante**.

authentication? -> : e_n_r_minus_one → **inviato dall'Attaccante e non da Initiator**.

Conclusioni

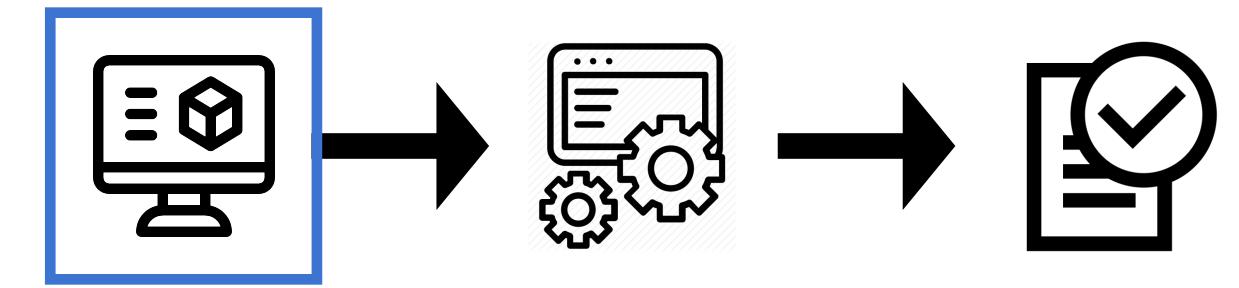
- È importante avere garanzie di sicurezza sui protocolli prima di utilizzarli in applicazioni commerciali
- I verificatori automatici basati sul modello simbolico sono attualmente la scelta migliore per verificare la correttezza dei protocolli
- Utilizzando la Toolchain proposta si riesce ad avvicinare l'ingegneria dei sistemi alla verifica dei protocolli

Obiettivi futuri

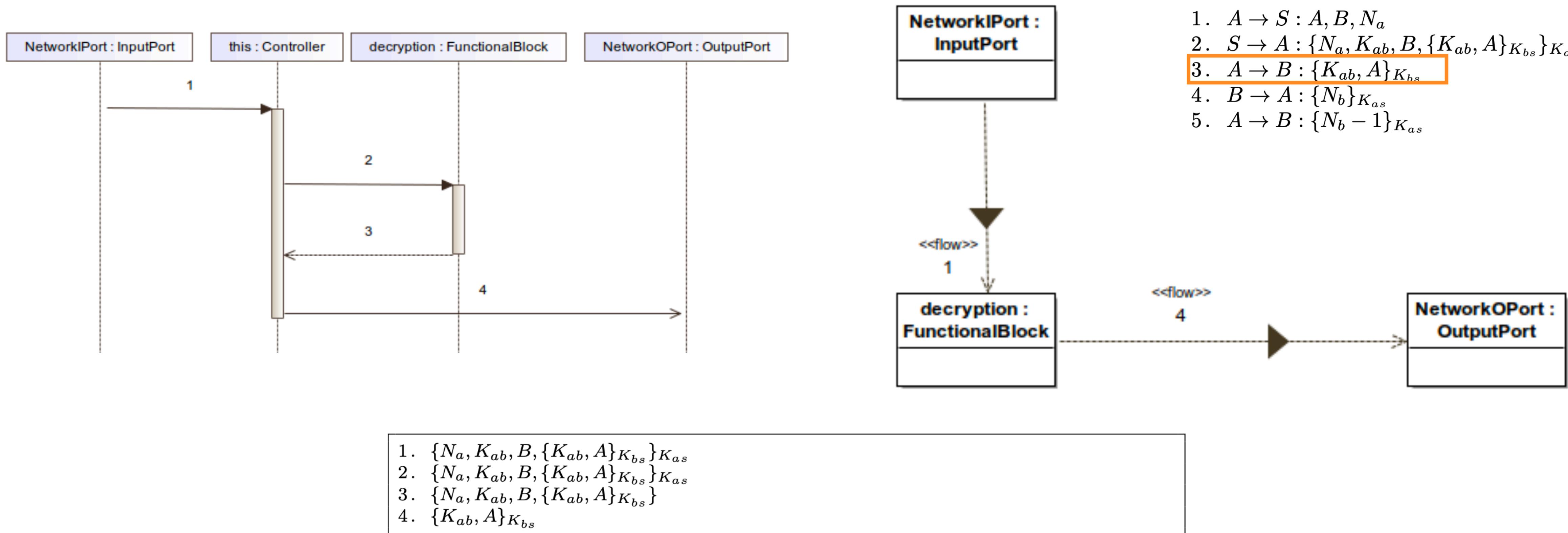
Utilizzare i software open source Modelio e VerifPal insieme al tool di conversione presentato, per creare un unico software open source in grado di consentire ai progettisti di modellare i protocolli tramite diagrammi UML ed avere immediatamente garanzie sulla sicurezza dei protocolli.

Modellazione UML

Modellazione



- Esistono 14 tipi di diagrammi nella modellazione UML, suddivisi in diagrammi strutturali e comportamentali
- Il più adatto alla rappresentazione dei protocolli è il sequence diagram
- Una valida alternativa può essere l'object diagram



ProVerif

Applied Pi Calculus: estende Pi Calculus e consente di modellare processi concorrenti e la loro interazione. Utilizza termini al posto dei nomi dei messaggi ed aggiunge l'algebra utile a modellare le operazioni crittografiche mediante la teoria equazionale.

Modellazione:

1. dichiarazione formale del comportamento delle primitive crittografiche
2. definizione di macroprocessi che consentono l'utilizzo di sotto-processi per semplificare lo sviluppo
3. codifica del protocollo stesso come processo principale utilizzando le macro

Dichiarazioni:

- **type** t .
- **free** n : t [private] .
- **fun** f(t_1, \dots, t_n) : t
- **reduc forall** $x_{1,1} : t_{1,1}, \dots, x_{1,n_1} : t_{1,n_1};$
 $g(M_{1,1}, \dots, M_{1,k}) = M_{1,0}$

Grammatica di base:

$M, N ::=$	<i>termini</i>
$a, b, c, \dots, k, \dots, m, n, \dots, s$	<i>nomi</i>
x, y, z	<i>variabili</i>
$h(M_1, \dots, M_k)$	<i>applicazione di costruttore/decostruttore</i>
$M = N$	<i>uguaglianza tra termini</i>
$M <> N$	<i>disugualanza tra termini</i>
$M \&\& N$	<i>congiunzione</i>
$M M$	<i>disgiunzione</i>
$\text{not } M$	<i>negazione</i>

Grammatica dei processi:

$P, Q ::=$	<i>processi</i>
0	<i>processo vuoto</i>
$P Q$	<i>composizione parallela</i>
$!P$	<i>replicazione</i>
$\text{new } n : t; P$	<i>limitazione del nome</i>
$\text{if } M = N \text{ then } P \text{ else } Q$	<i>condizione</i>
$\text{in}(M, x : t); P$	<i>messaggio in input</i>
$\text{out}(M, N); P$	<i>messaggio in output</i>
$\text{let } x = M \text{ in } P \text{ else } Q$	<i>valutazione del termine</i>
$R(M_1, \dots, M_k)$	<i>utilizzo delle macro</i>

Proprietà di sicurezza:

- **query** attacker (M)
- **query** $x_1 : t_1, \dots, x_n : t_n;$
- **event** ($e_1(M_1, \dots, M_j)$) ==> **event** ($e_0(N_1, \dots, N_k)$).

VerifPal

Linguaggio: richiede solo la definizione dei partecipanti che hanno stati indipendenti, conoscono determinati valori ed eseguono operazioni con le primitive crittografiche.
Internamente basato su costruzione e decostruzione di termini simbolici permette di modellare un protocollo come se fosse descritto in una conversazione informale.

Keyword:

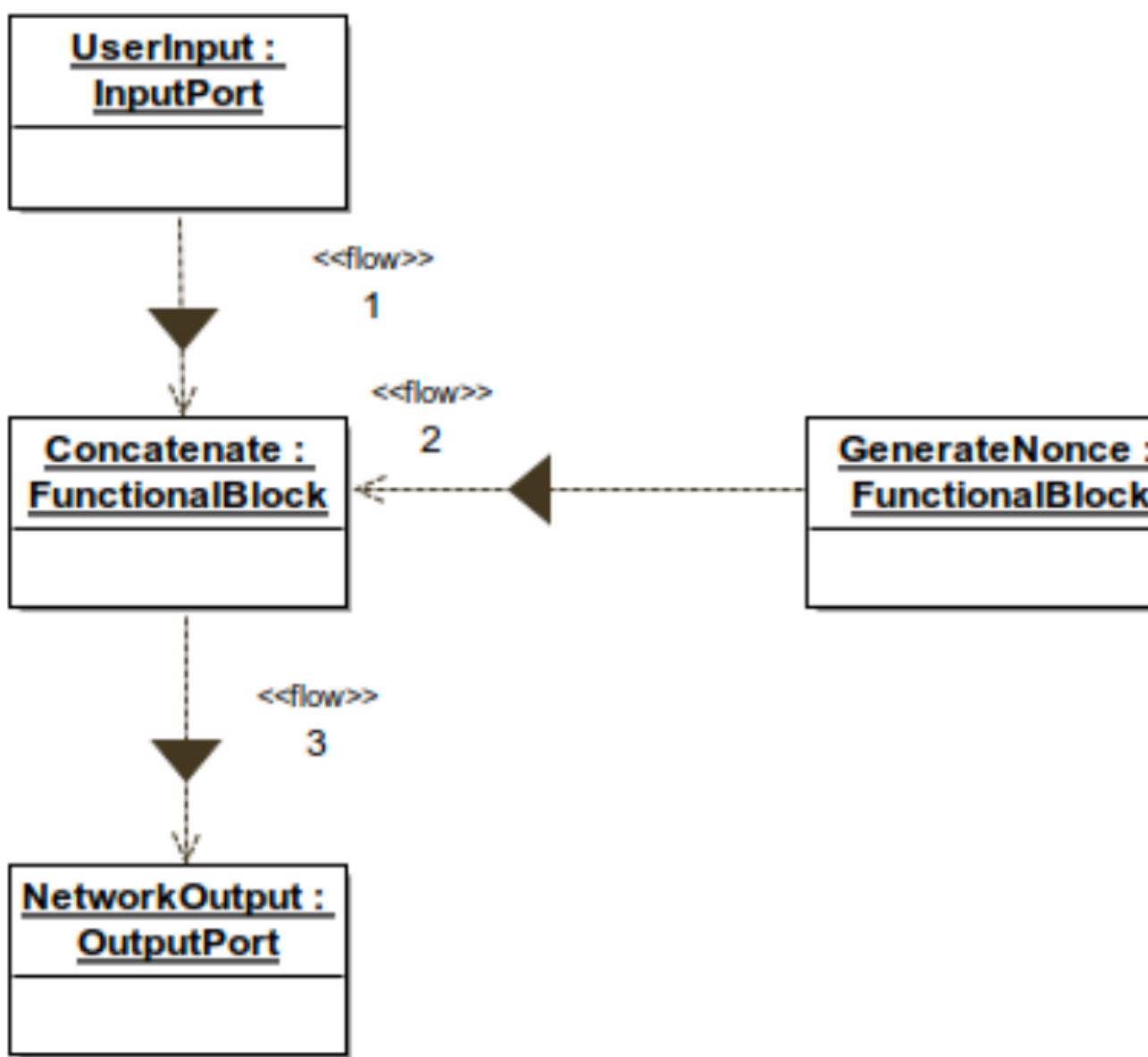
- **attacker[]**: serve a definire il tipo di attaccante, attivo o passivo
- **principal**: serve ad inizializzare i partecipanti al protocollo
- **knows**: inizializza una costante con conoscenze pregresse del principal
- **generates**: inizializza una costante con un valore generato al momento
- **private**: opzione nell'inizializzazione delle costanti, significa che la costante è conosciuta solo dal principal
- **A -> B : m** : rappresenta l'invio di un messaggio dal principal A al principal B
- **queries**: indica il blocco del modello dove va inserito cosa si vuole verificare
- **confidentiality?**: verifica della confidenzialità
- **authentication?**: verifica dell'autenticazione
- **freshness?**: verifica che il messaggio sia nuovo e non un replay attack
- **leaks**: consente di forzare la conoscenza di una costante privata all'attaccante

Primitive fornite (le stesse del modello Dolev Yao):

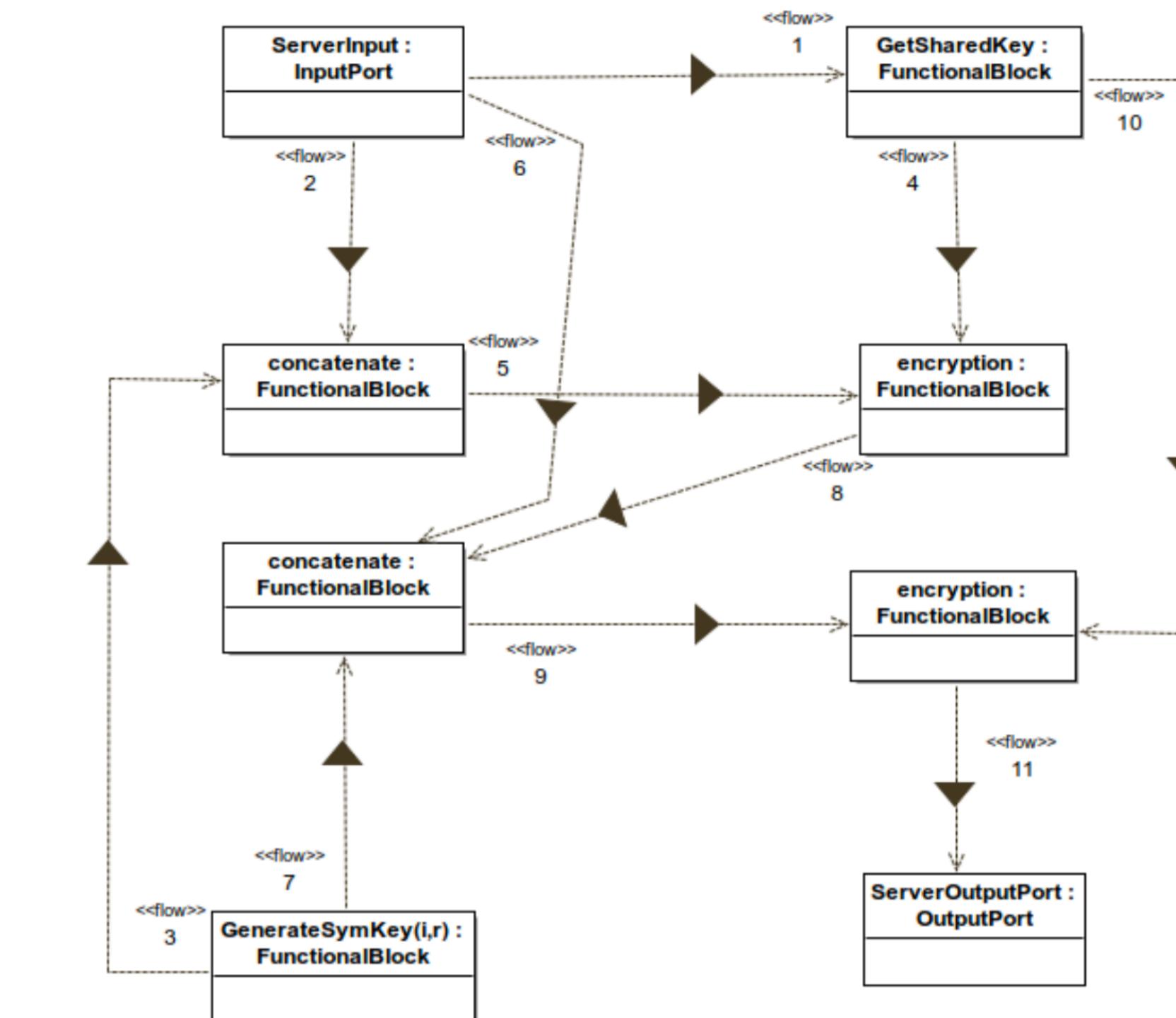
- **CONCAT(a,b...)** : $c \rightarrow$ concatena due o più valori in uno
- **SPLIT(CONCAT(a,b...))** : $a,b. \rightarrow$ separa una concatenazione negli elementi che la compongono
- **ENC(key,paintext)** : ciphertext \rightarrow codifica nella crittografia a chiave simmetrica
- **DEC(key, ENC(key,paintext))**: plaintext \rightarrow decodifica nella crittografia a chiave simmetrica
- **PKE_ENC(G^{key} , plaintext)** : ciphertext \rightarrow codifica nella crittografia a chiave asimmetrica
- **PKE_DEC(key, PKE_ENC(G^{key} , plaintext))** : plaintext \rightarrow decodifica nella crittografia a chiave asimmetrica
- **SIGN(key, message)** : signature \rightarrow firma un messaggio
- **SIGNVERIF(G^{key} , message, SIGN(key, message))** : message \rightarrow verifica della firma

Needham-Schroeder Symmetric Key

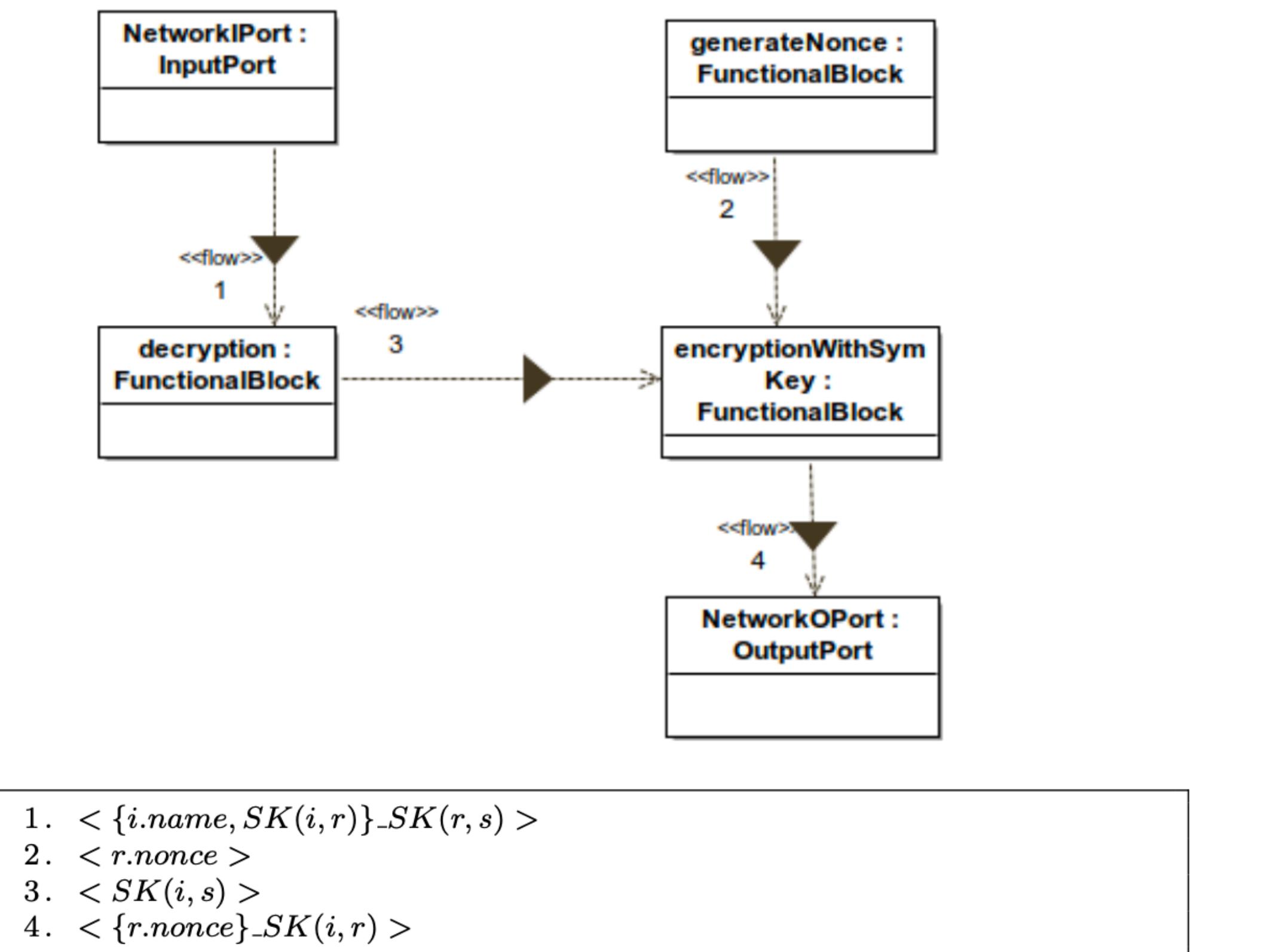
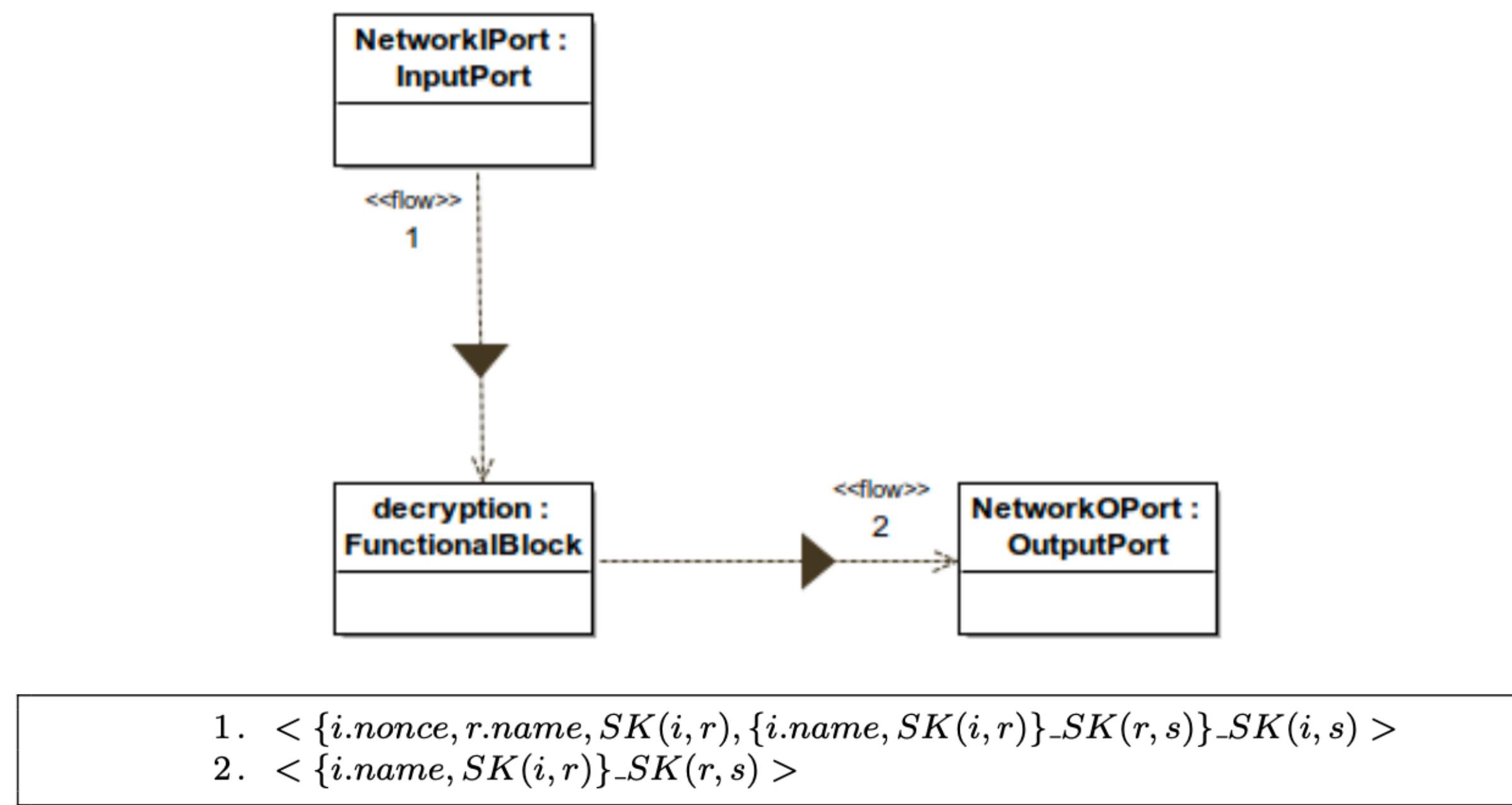
1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, K_{ab}, B, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b\}_{K_{as}}$
5. $A \rightarrow B : \{N_b - 1\}_{K_{as}}$



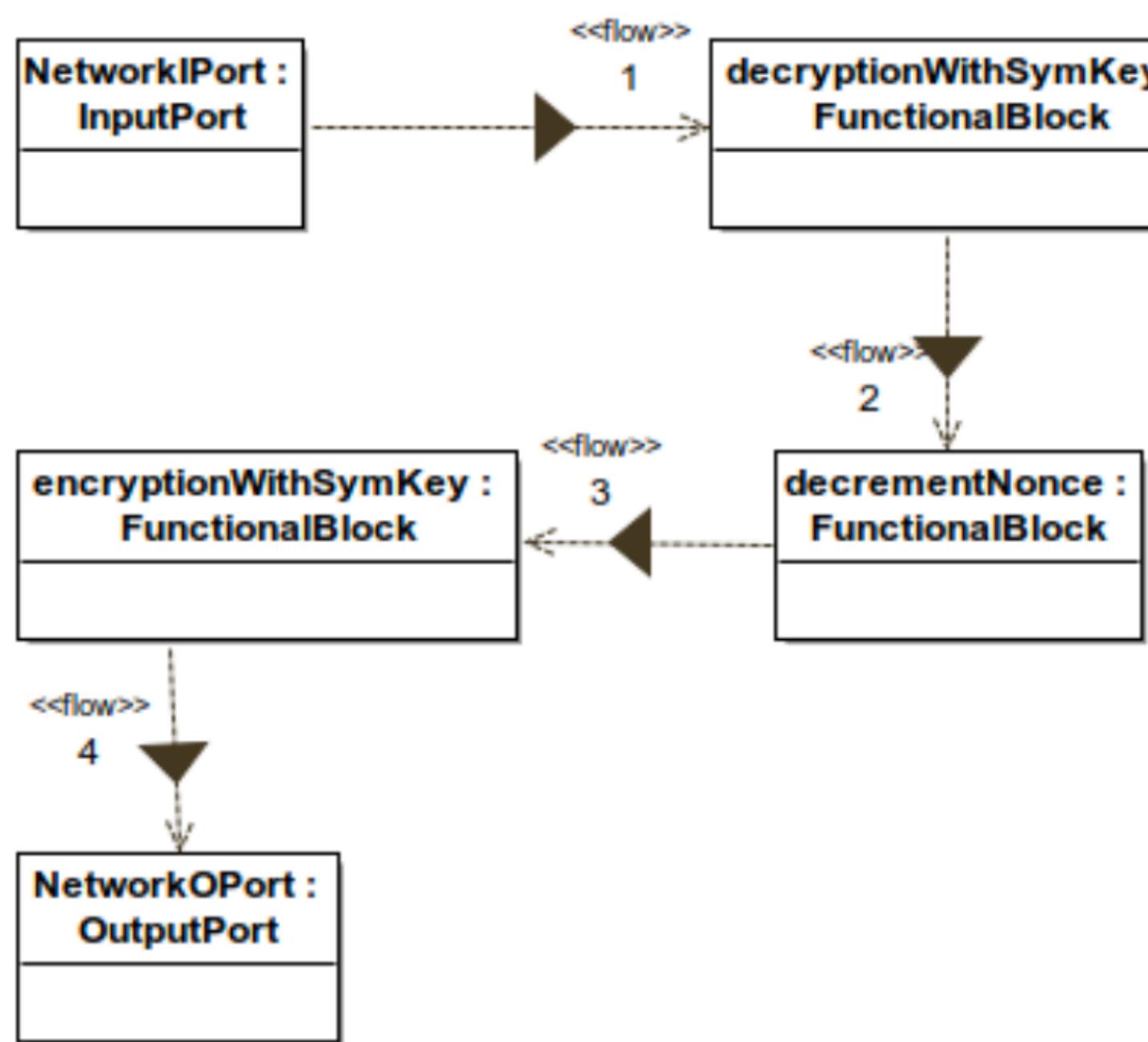
- 1. $< i.name, r.name >$
- 2. $< i.nonce >$
- 3. $< i.name, r.name, i.nonce >$



1. $< i.name, r.name >$
2. $< i.name >$
3. $< SK(i, r) >$
4. $< SK(s, r) >$
5. $< i.name, SK(i, r) >$
6. $< r.name, i.nonce >$
7. $< SK(i, r) >$
8. $< \{i.name; SK(i, r)\} - SK(r, s) >$
9. $< i.nonce, r.name, SK(i, r), \{i.name, SymKey(i, r)\} - SK(r, s) >$
10. $< SK(s, r) >$
11. $< \{i.nonce, r.name, SK(i, r), \{i.name, SK(i, r)\} - SK(rs)\} - SK(i, s) >$



Estratto .xmi

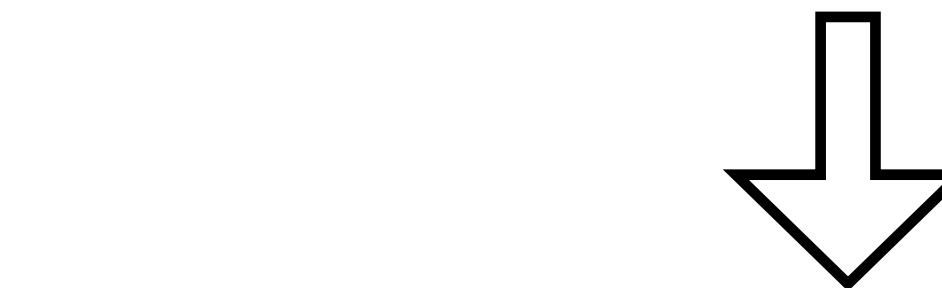


- 1. $\langle r.\text{nonce}-SK(i, r) \rangle$
- 2. $\langle r.\text{nonce} \rangle$
- 3. $\langle r.\text{nonce} - 1 \rangle$
- 4. $\langle \{r.\text{nonce} - 1\}\text{-}SK(i, r) \rangle$

```

<ownedAttribute xmi:type="uml:Property" xmi:id="_
_WmudcmlZEEunebkXFN4K9w" name="GenerateSymKey(i, r)" type="_
_WmudZWlZEEunebkXFN4K9w" aggregation="composite"/>
<ownedAttribute xmi:type="uml:Property" xmi:id="_
_Wmudc2lZEEunebkXFN4K9w" name="GetSharedKey" type="_
_WmudZWlZEEunebkXFN4K9w" aggregation="composite"/>

<packagedElement xmi:type="uml:InformationFlow" xmi:id="_
_WmudhmlZEEunebkXFN4K9w" name="<i.name, r.name>" informationSource="_
_WmudcGlZEEunebkXFN4K9w" informationTarget="_
_Wmudc2lZEEunebkXFN4K9w"/>
  
```



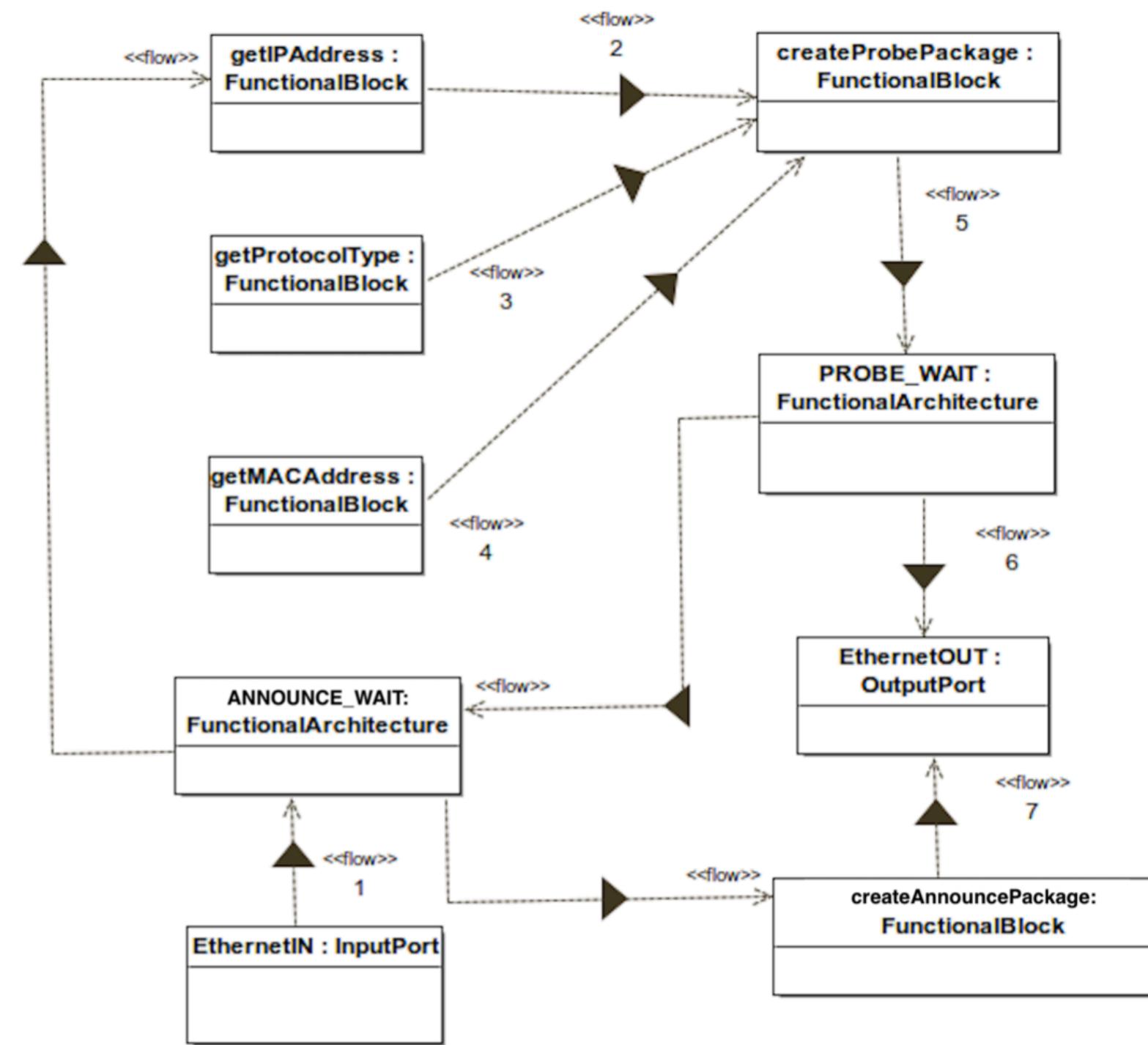
Dizionario degli oggetti

```
'_Wmudc2lZEEunebkXFN4K9w': { 'name': 'GetSharedKey', 'type': 'FunctionalBlock' },
```

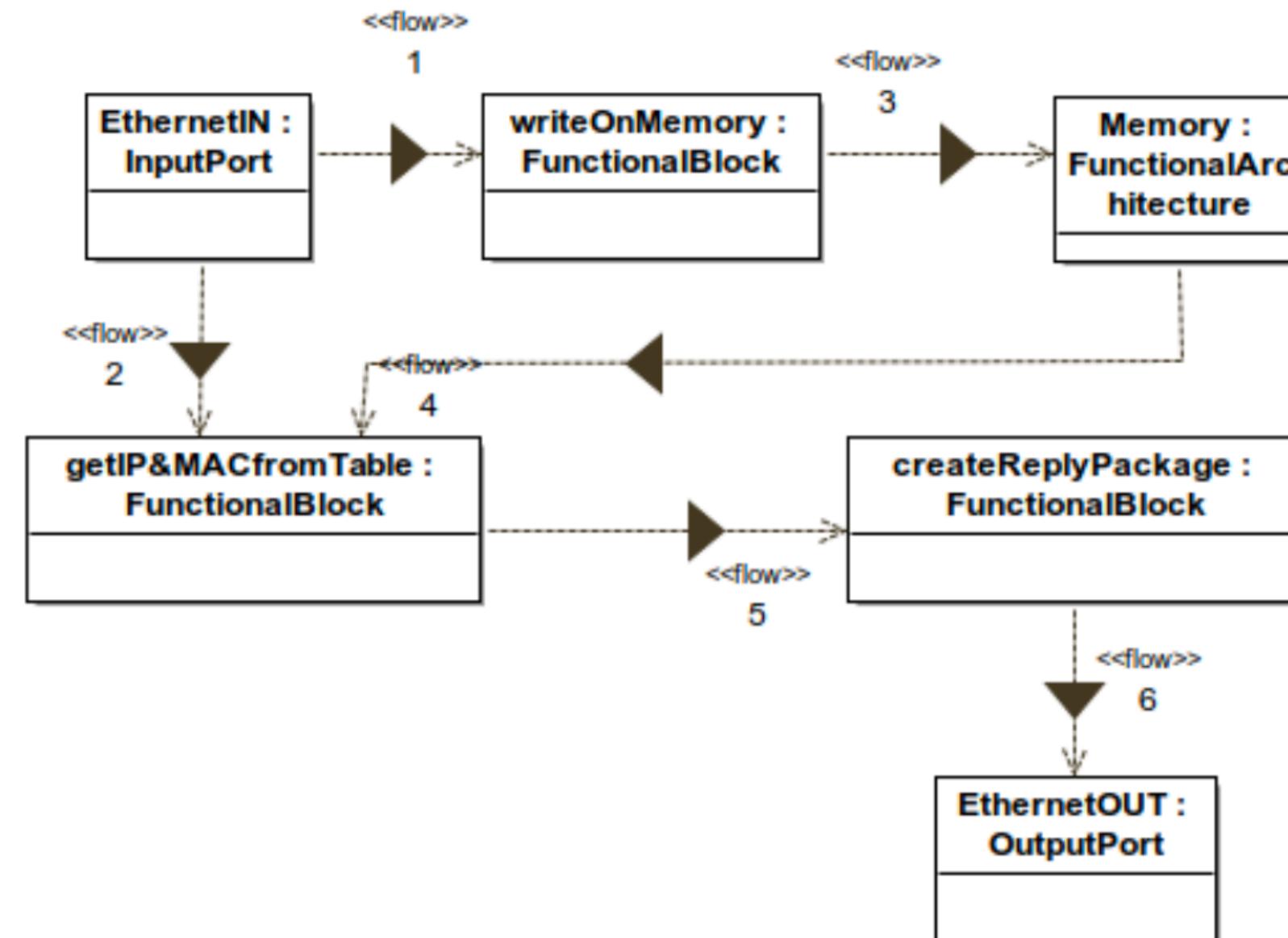
Dizionario delle InformationFlow

```
{1: { 'information': '<i.name, r.name>', 'source': '_WmudcGlZEEunebkXFN4K9w', 'target': '_Wmudc2lZEEunebkXFN4K9w' }},
```

Address Resolution Protocol



1. $\langle \{as$hrd, arpro, arhln, ar$pln, reply, ar$sha, arspa, artha, ar$tpa\} \rangle$
2. $\langle ip \rangle$
3. $\langle ar$spa \rangle$
4. $\langle ar$sha \rangle$
5. $\langle \{as$hrd, arpro, arhln, ar$pln, request, ar$sha, 0.0.0.0, 00:00:00:00:00:00, ip\} \rangle$
6. $\langle \{as$hrd, arpro, arhln, ar$pln, request, ar$sha, 0.0.0.0, 00:00:00:00:00:00, ip\} \rangle$
7. $\langle \{as$hrd, arpro, arhln, ar$pln, request, ar$sha, ip, 00:00:00:00:00:00, ip\} \rangle$



1. $\langle \{as$hrd, arpro, arhln, ar$pln, request, ar$sha, ip, 00:00:00:00:00:00, ip\} \rangle$
2. $\langle \{as$hrd, arpro, arhln, ar$pln, request, ar$sha, 0.0.0.0, 00:00:00:00:00:00, ip\} \rangle$
3. $\langle ar$sha, ip \rangle$
4. $\langle ar$tha, ar$tpa \rangle$
5. $\langle ar$tha, ar$tpa \rangle$
6. $\langle \{as$hrd, arpro, arhln, ar$pln, reply, ar$sha, arspa, artha, ar$tpa\} \rangle$

Address Resolution Protocol

```

attacker[active]

principal A[
    knows private ar_hrd, ar_pro, ar_hln, type_req,
    ar_sha
    generates ip
    info_protocol_a = CONCAT(ar_hrd, ar_pro, ar_hln,
        type_req)
    info_address_a = CONCAT(ar_sha, nil, nil, ip)
    probe = CONCAT(info_protocol_a,info_address_a)
]

A -> B : probe

principal B[
    info_pro_probe, info_add_probe=SPLIT(probe)
    ar_sha_probe,ar_spa_probe,ar_tha_probe,
    ar_tpa_probe=SPLIT(info_add_probe)
]

principal A[
    info_address_a2 = CONCAT(ar_sha, ip, nil, ip)
    announce = CONCAT(info_protocol_a, info_address_a2
        )
]

A -> B : announce

principal B[
    _, info_announce=SPLIT(announce)
    ar_sha_announce,ar_spa_announce,_,_=SPLIT(
        info_announce)
]

queries[
    confidentiality? probe
    confidentiality? announce
    authentication? A -> B: probe
    authentication? A -> B: announce
]

```

Risultati:

confidentiality? probe → **violata dall'Attaccante**.

confidentiality? announce → **violata dall'Attaccante**.

authentication? -> : probe → **invia dall'Attaccante e non da A**.

authentication? -> : announce → **invia dall'Attaccante e non da A**.

```

attacker[active]

principal A[
    knows private ar_hrd, ar_pro, ar_hln, type_req,
    ar_sha
    generates ip
    info_protocol_a = CONCAT(ar_hrd, ar_pro, ar_hln,
        type_req)
    info_address_a = CONCAT(ar_sha, nil, nil, ip)
    probe = CONCAT(info_protocol_a,info_address_a)
]

A -> B : probe

principal B[
    knows public ip_used, sha_used
    knows private type_reply
    in_prot_a, in_add_a=SPLIT(probe)
    ar_hrd_a, ar_pro_a, ar_hln_a, type_req_a = SPLIT(
        in_prot_a)
    ar_sha_a, _, _, ar_tpa_a = SPLIT(in_add_a)
    info_protocol_b=CONCAT(ar_hrd_a, ar_pro_a,
        ar_hln_a, type_reply)
    info_address_b = CONCAT(ar_sha_a, ip_used,
        sha_used, ip_used)
    reply = CONCAT(info_protocol_b,info_address_b)
]

B -> A : reply

principal A[
    _, info_reply=SPLIT(reply)
    ___,ar_tha_reply,_=SPLIT(info_reply)
    _=ASSERT(ar_sha,ar_tha_reply)
]

queries[
    confidentiality? probe
    confidentiality? reply
    authentication? A -> B: probe
    authentication? B -> A: reply
]

```

Risultati:

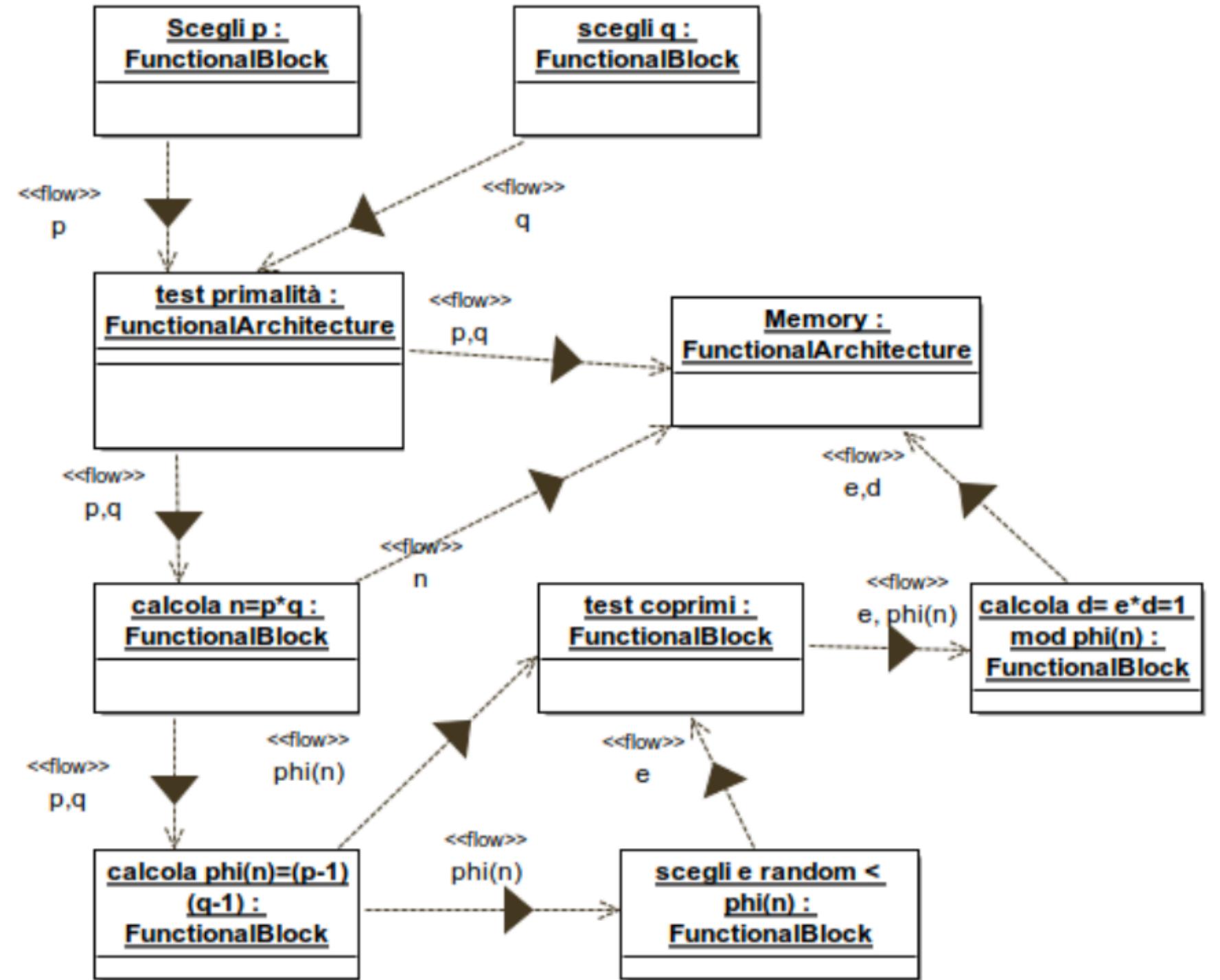
confidentiality? probe → **violata dall'Attaccante**.

confidentiality? reply → **violata dall'Attaccante**.

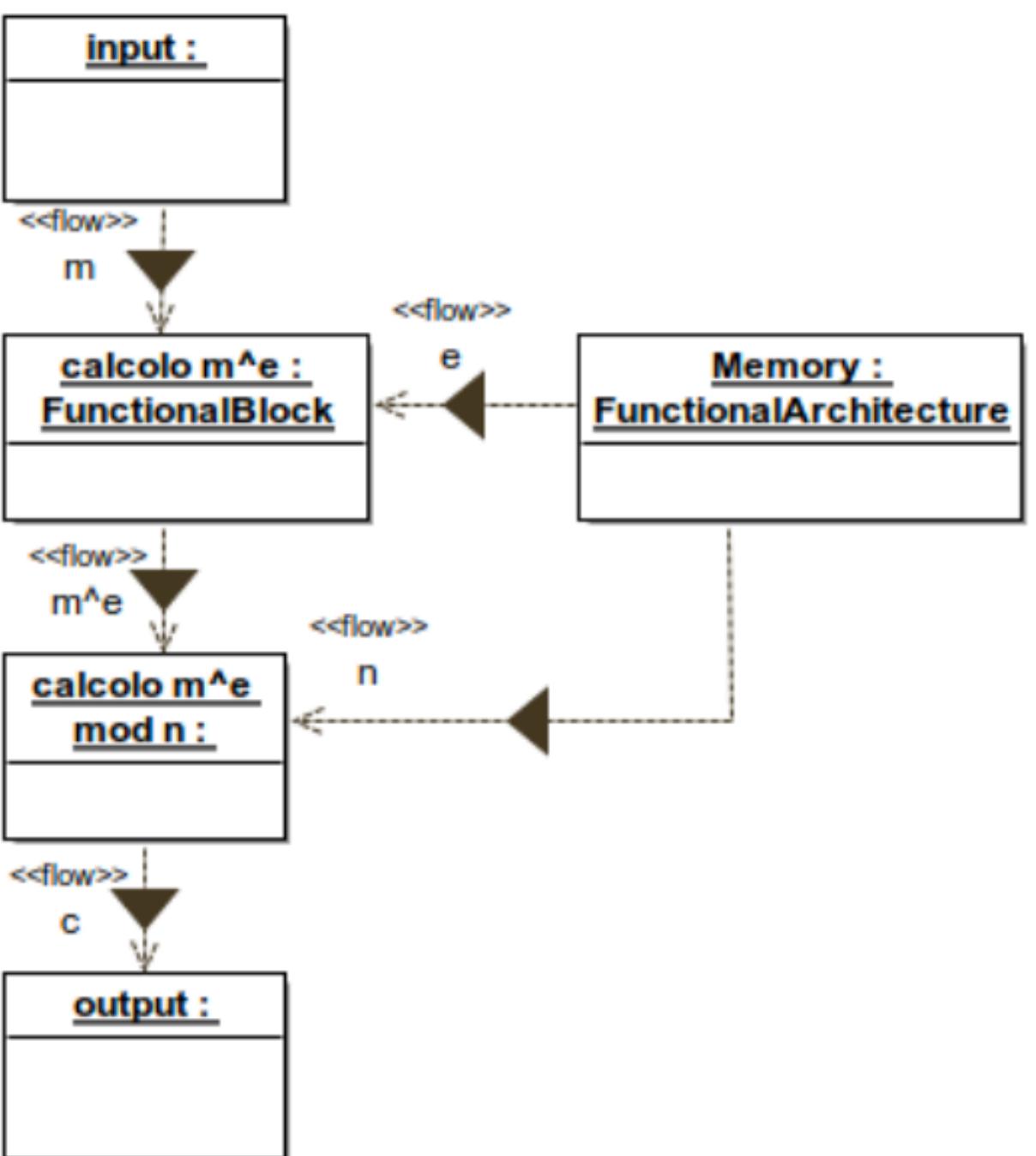
authentication? -> : probe → **invia dall'Attaccante e non da A**.

authentication? -> : reply → **invia dall'Attaccante e non da B**.

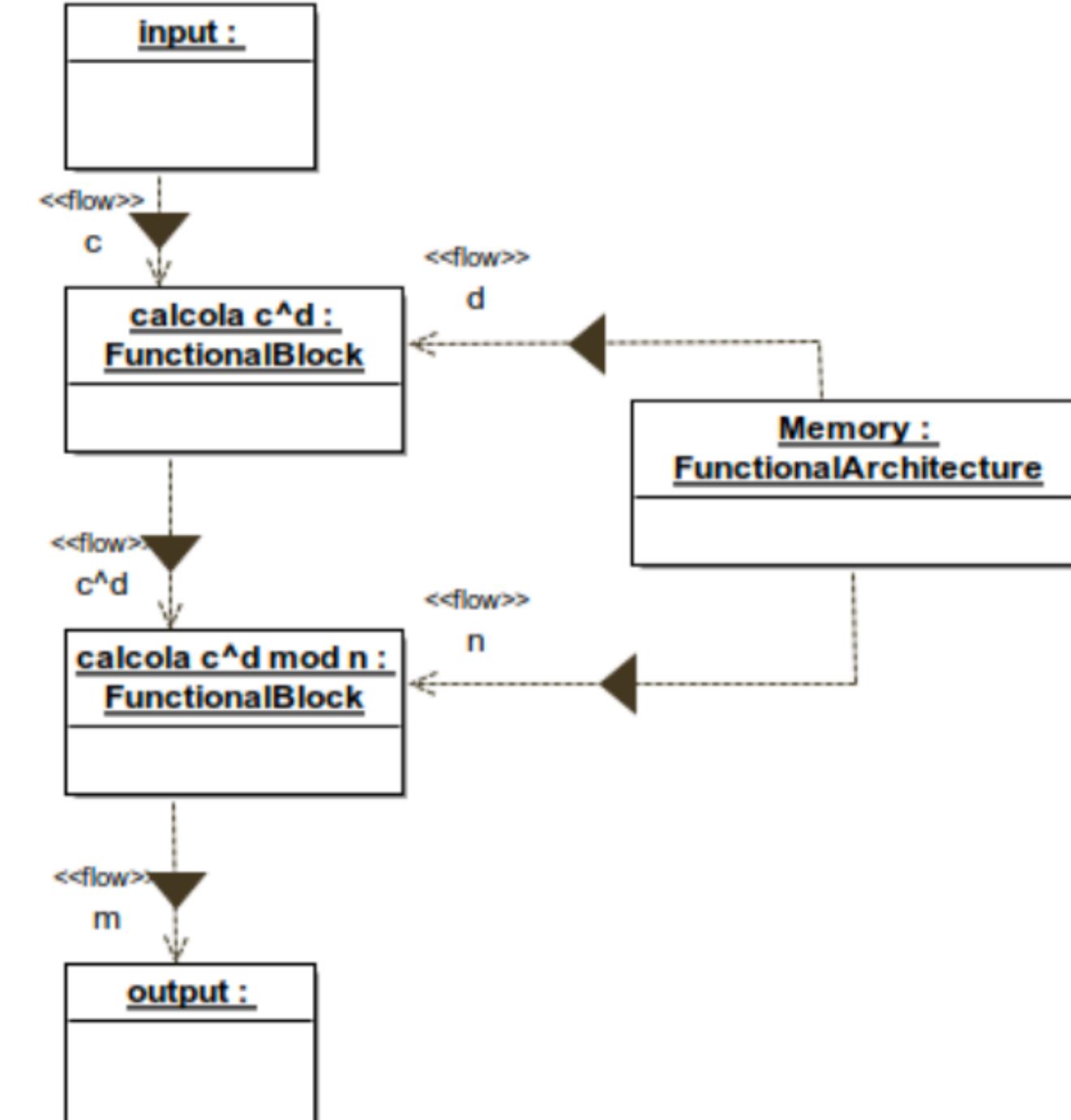
RSA



Generazione delle chiavi



Funzione di encryption



Funzione di decryption

RSA

Risultati:

confidentiality? $m_b \rightarrow \checkmark$

authentication? $\rightarrow : e_2 \rightarrow$ inviato dall'Attaccante e non da Bob..

Possibili fix:

Alice \rightarrow Bob : [ga]

Bob \rightarrow Alice : [gb]

```
attacker[active]

principal Alice[
    knows private a
    knows private m_a
    ga = G^a
]

principal Bob[
    knows private b
    knows private m2_b
    gb = G^b
]

Alice -> Bob : ga
Bob -> Alice : gb

principal Alice[
    e1 = PKE_ENC(gb, m_a)
]

Alice -> Bob : e1

principal Bob[
    m_b = PKE_DEC(b, e1)
    e2=PKE_ENC(ga, m2_b)
]

Bob -> Alice : e2

principal Alice[
    m2_a=PKE_DEC(a, e2)
]

queries [
    authentication? Bob -> Alice: e2
    confidentiality? m_b
]
```