Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

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

Candidato:
Alessandro Busatto
Matricola VR432075

Relatore:
Prof.ssa Mila Dalla Preda

Correlatore:
Marco Rocchetto

Anno Accademico 2019-2020

Prima di procedere con la trattazione, vorrei dedicare qualche riga a tutti coloro che mi sono stati vicini in questo percorso di crescita personale e professionale.
Un sentito grazie alla mia relatrice, Prof.ssa Dalla Preda, per la sua infinita disponibilità e tempestività ad ogni mia richiesta.
Un ringraziamento speciale va al mio correlatore, Marco Rocchetto, per essere stato negli ultimi mesi il mio punto di riferimento per la stesura di questa tesi, mettendo a mia disposizione la sua esperienza, la sua professionalità ed il suo tempo. Grazie anche per tutti i consigli e per aver suscitato in me l'interesse nel campo della ricerca.
Ringrazio i miei genitori per avermi sempre supportato, per i loro saggi consigli e la loro capacità di ascoltarmi, per essere sempre stati al mio fianco, soprattutto nei momenti di sconforto. Senza il loro supporto morale non sarei mai potuto arrivare fin qui.
Tutti i miei amici e colleghi hanno avuto un peso determinante nel conseguimento di questo risultato. Grazie per aver condiviso con me in questi anni un percorso così bello e importante, ma non privo di momenti di difficoltà.

Indice

Elenco delle figure

1 Introduzione

Negli ultimi anni abbiamo assistito all'evoluzione del mondo digitale, la nascita di Internet e lo sviluppo di nuove tecnologie ha cambiato radicalmente le nostre vite, ad esempio le e-mail ci consentono di spedire lettere senza dover andare fisicamente ad imbucarle nella cassetta della posta, è possibile gestire i conti bancari comodamente da casa accedendo all'area personale del sito della banca, controllare gli elettrodomestici da remoto quando non si è a casa, frequentare le lezioni o lavorare da casa quando impossibilitati ad essere presenti in aula o sul luogo di lavoro.
Naturalmente tutta quest'evoluzione ci ha portato numerosi vantaggi, ma allo stesso tempo sono emersi nuovi tipi di problematiche legati soprattutto alla gestione e alla protezione dei dati e delle informazioni che viaggiano attraverso la rete.
Molti ricercatori hanno cercato di trovare delle soluzioni a queste problematiche dando vita a protocolli di sicurezza che utilizzano la crittografia per mantenere al sicuro la comunicazione.
La storia ci insegna che anche protocolli di sicurezza ritenuti sicuri per anni, si sono dimostrati vulnerabili a qualche tipologia di attacco, come ad esempio il protocollo a chiave pubblica descritto da Needham e Schroeder in [NS78] che è risultato vulnerabile ad un attacco descritto da Lowe in [Low95].
Per questo motivo si cercano sempre dei nuovi protocolli di sicurezza, o varianti di quelli già esistenti, in grado di mantenere al sicuro i nostri dati.
Come si può affermare che un protocollo di sicurezza rispetti determinate specifiche di sicurezza?
L'obiettivo di questo documento è analizzare lo stato dell'arte sulle varie tecniche per la verifica dei protocolli e cercare di avvicinare la verifica di protocolli all'ingegneria di sistemi utilizzando la modellazione UML.
Verrà proposta una nuova tecnica di modellazione per i protocolli di sicurezza mediante l'utilizzo dei diagrammi UML, questo perché le tecniche attualmente esistenti richiedono la conoscenza di un linguaggio di dominio specifico per la modellazione di protocolli da utilizzare nei tool per la verifica automatica.
Vedremo che attualmente esistono due tecniche per la verifica dei protocolli, il modello computazionale e il modello simbolico, ma solo il modello simbolico viene considerato abbastanza "maturo" per essere utilizzato e si presta ad essere automatizzato, per questo motivo al giorno d'oggi viene utilizzato dai principali tool di verifica automatica.
L'utilizzo della modellazione tramite diagrammi UML consente ai progettisti di protocolli di utilizzare una rappresentazione semi-grafica[1], di ragionare sulla struttura del protocollo senza curarsi dell'effettiva implementazione e di utilizzare il tool da me sviluppato, presentato successivamente, per convertire il diagramma UML nel linguaggio adatto per essere utilizzato dal tool di verifica VerifPal[2].

[1] un modello UML è composto da un insieme di diagrammi correlati tra loro ed ogni diagramma è costituito da diversi elementi grafici oltre che da elementi di testo libero

[2] il codice presentato nella tesi è consultabile presso il sito: `edu.v-research.it`

2 Modellazione dei protocolli mediante UML

Per la modellazione di protocolli[3] si è deciso di utilizzare lo standard UML (Unified Modeling Language) nella versione 2.4[4], perché questo tipo di modellazione utilizza una notazione semi-grafica che consente al progettista di ragionare sulla logica del protocollo che è più semplice rispetto al codice, senza dover porre eccessiva attenzione a come effettivamente deve essere implementato e senza dover necessariamente scrivere codice per la sua rappresentazione.
Esistono 14 tipi di diagrammi nella modellazione UML, suddivisi in diagrammi strutturali e diagrammi comportamentali (o di interazione).
Tra i vari tipi di diagrammi troviamo il sequence diagram, un diagramma comportamentale, definito per la rappresentazione dettagliata delle interazioni tra gli agenti (componenti hardware o software) partecipanti al protocollo, per sua natura risulta essere il diagramma più adatto alla rappresentazione dei protocolli.
Una caratteristica di questo tipo di diagramma è che l'asse verticale rappresenta il tempo, quindi è possibile rappresentare l'ordine cronologico delle interazioni tra gli agenti partecipanti, dall'alto verso il basso.
In alternativa può essere utilizzato l'object diagram, un diagramma di tipo strutturale nel quale vengono rappresentate le relazioni tra i vari oggetti che compongono il sistema, nel nostro caso le relazioni e gli oggetti corrispondono rispettivamente a interazioni e agenti del sequence diagram.
Gli altri 12 tipi di diagramma sono stati esclusi dai possibili diagrammi utili per rappresentare protocolli perché descrivono aspetti ortogonali alle interazioni descritte nel sequence diagram.
Nella Sezione 2.1 verrà spiegato in dettaglio perché si può utilizzare indifferentemente uno tra il sequence diagram e l'object diagram.

2.1 Confronto tra sequence diagram e object diagram

Per la modellazione tramite lo standard UML è stato utilizzato il tool, open source, Modelio[5] nella versione 4.1.
Dato il protocollo Needham Schroeder Symmetric Key descritto in [NS78]:

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

Dove:

- $X \to Y : m$ indica che il principal `X` invia un messaggio al principal `Y`,
- N_x indica il nonce generato dal principal `X`,
- K_{xy} indica la chiave condivisa tra i principal `X` e `Y`,

[3] per protocolli si intendono protocolli di sicurezza come Needham Schroeder Symmetric Key, protocolli di comunicazione come ARP e protocolli di encryption come RSA
[4] `https://www.omg.org/spec/UML/2.4/About-UML/`
[5] `https://www.modelio.org/`

- $\{\ldots\}_{K_{xy}}$ indica che il pacchetto è cifrato con la chiave condivisa tra i principal `X` e `Y`.

Le figure 1-2, rappresentano il terzo messaggio del protocollo rispettivamente con un sequence ed un object diagram.
Più precisamente vengono rappresentate le operazioni fatte dall'agente `A` una volta ricevuta la chiave simmetrica da utilizzare con `B` dal server `S`.
L'agente `A` decifra con la chiave condivisa con il server `S` il pacchetto, ed inoltra all'agente `B` il pacchetto cifrato con la chiave condivisa tra `B` e `S`, contenente la chiave simmetrica tra `A` e `B`.

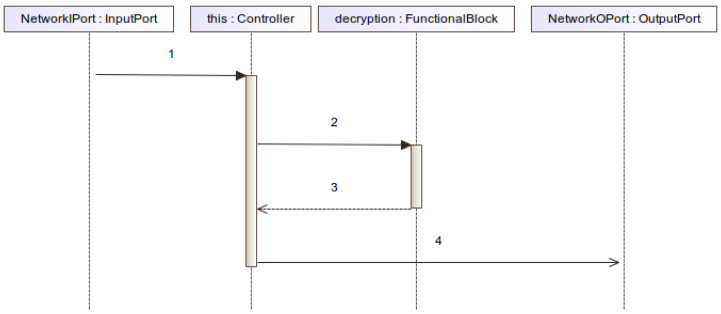


Figura 1: Sequence diagram: Inoltro pacchetto da `A` a `B`

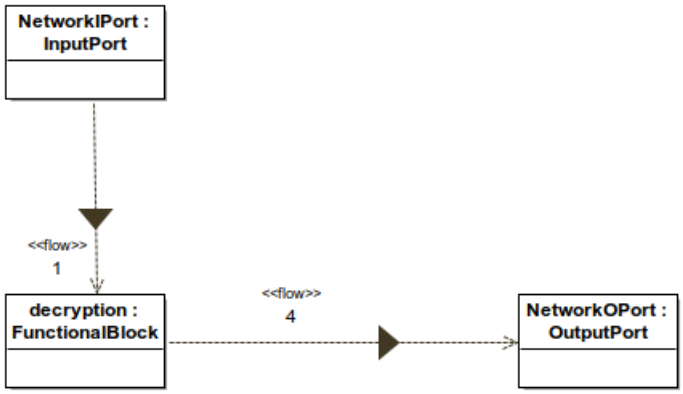


Figura 2: Object diagram: Inoltro pacchetto da `A` a `B`

1. $\{N_a, K_{ab}, B, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
2. $\{N_a, K_{ab}, B, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $\{N_a, K_{ab}, B, \{K_{ab}, A\}_{K_{bs}}\}$
4. $\{K_{ab}, A\}_{K_{bs}}$

Durante la fase di ingegnerizzazione di un sistema viene modellata la parte fisica, ovvero le componenti hardware che compongono il sistema, l'obiettivo è quello di unirla alla modellazione logica del funzionamento del protocollo attraverso la modellazione UML.

Per fare questo si è resa necessaria l'introduzione di nuovi tipi[6] di elementi, che non appartengono ai tipi degli elementi standard dei diagrammi UML.

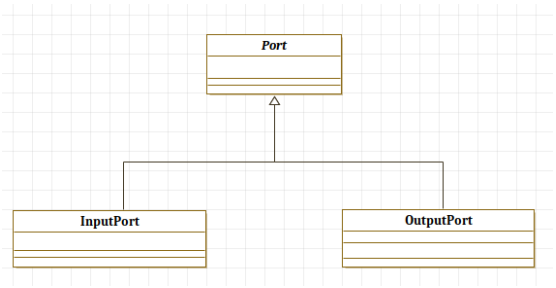


Figura 3: Class diagram Port

Come vediamo in Figura 3 sono stati definiti i tipi InputPort e OutPort, specializzazioni del tipo Port.
Questi tipi di oggetti vengono utilizzati nelle due tipologie di diagramma come unico punto di contatto tra i modelli del software e dell'hardware, infatti sono le componenti utilizzate dai protocolli per ricevere o inoltrare messaggi.
Inoltre, guardando le lifelines in Figura 1 si può notare anche il Controller, il quale è il dispositivo proprietario delle porte.
Anche per la modellazione logica del funzionamento del protocollo si è resa necessaria l'introduzione di nuovi tipi.

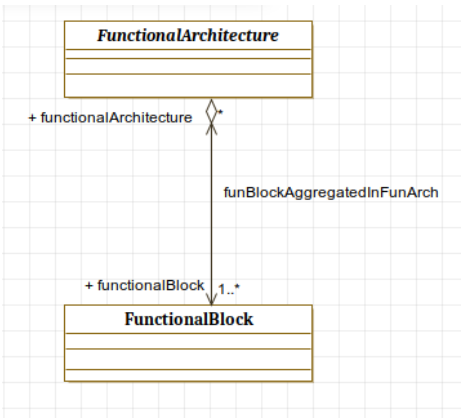


Figura 4: Class diagram Function

Gli elementi di tipo FunctionalBlock vengono utilizzati per la codifica di un'operazione che non può essere dettagliata maggiormente (oppure non è necessario farlo considerando l'obiettivo della verifica di questo modello) che, presi degli input, restituisce un output.
Nel caso in cui si voglia rappresentare un insieme di operazioni si utilizzano gli

[6]con abuso di notazione, quando si parla di tipi di elementi, si intende la tipologia degli oggetti della classe istanziata

elementi di tipo FunctionalArchitecture, questi vengono utilizzati come "segnaposto" per l'insieme di operazioni che verrà descritto in un altro diagramma dello stesso tipo di quello utilizzato, avente come nome lo stesso dato all'elemento di tipo FunctionalArchitecture.
Infatti come vediamo in Figura 4 una FunctionalArchitecture può essere composta da uno o più elementi di tipo FunctionalBlock.
In entrambi i diagrammi le InformationFlow rappresentano gli input e gli output dei vari elementi, nelle etichette vengono descritti i parametri passati, per convenzione se i parametri si trovano tra { } significa che l'input o l'output di un oggetto è un pacchetto.
Modelio consente di esportare i modelli UML in file .xmi (XML Metadata Interchange), uno standard basato sulla struttura XML che ne consente lo scambio tra applicazioni.
Nel file .xmi, strutturato come un xml, troviamo i tag per l'identificazione tramite codice univoco (id) degli elementi (lifelines, oggetti, InformationFlow) e i tag per descrivere le proprietà dei vari elementi (tipo, nome, descrizione). Tra le proprietà degli elementi di tipo InformationFlow sono presenti i tag per identificare tramite id l'elemento sorgente e l'elemento destinazione delle informazioni.
Analizzando il file .xmi estratto da un protocollo modellato tramite sequence diagram (Listing 1) o da un protocollo modellato tramite object diagram (Listing 2), possiamo vedere come l'estrazione delle lifelines del sequence diagram corrisponde all'estrazione degli oggetti dell'object diagram (righe 7-12) e l'estrazione delle information flow viene rappresentata nello stesso modo (righe 14-20).

Listing 1: Estratto di XMI da sequence diagram

```
1  <packagedElement
2   xmi:type="uml:Class"
3   xmi:id="
        _WmudZWlZEeunebkXFN4K9w"
4   name="FunctionalBlock"
5  />
6
7  <packagedElement xmi:type="uml
       :Lifeline"
8   xmi:id="
        _WmudemlZEeunebkXFN4K9w"
9   name="decryption"
10  type="_WmudZWlZEeunebkXFN4K9w
        "
11  aggregation="composite"
12 />
13
14 <packagedElement
15  xmi:type="uml:InformationFlow
        "
16  xmi:id="
        _WmudkWlZEeunebkXFN4K9w"
17  name="{Na, Kab, B, {Kab, A}Kbs}Kas
        "
18  informationSource="
        _WmudeWlZEeunebkXFN4K9w"
19  informationTarget="
        _WmudemlZEeunebkXFN4K9w"
20 />
```

Listing 2: Estratto di XMI da object diagram

```
<packagedElement
 xmi:type="uml:Class"
 xmi:id="
      _WmudZWlZEeunebkXFN4K9w"
 name="FunctionalBlock"
/>

<packagedElement xmi:type="uml
    :InstanceSpecification"
 xmi:id="
      _WmudemlZEeunebkXFN4K9w"
 name="decryption"
 type="_WmudZWlZEeunebkXFN4K9w
      "
 aggregation="composite"
/>

<packagedElement
 xmi:type="uml:InformationFlow
      "
 xmi:id="
      _WmudkWlZEeunebkXFN4K9w"
 name="{Na, Kab, B, {Kab, A}Kbs}Kas
      "
 informationSource="
      _WmudeWlZEeunebkXFN4K9w"
 informationTarget="
      _WmudemlZEeunebkXFN4K9w"
/>
```

Questo ci consente di affermare che utilizzando uno a scelta tra i due diagrammi è possibile modellare un protocollo senza la perdita di alcuna informazione.
Il file .xmi può essere utilizzato come input di un software in grado di navigarne la struttura e restituire in output dei file pronti per essere utilizzati dai tool di verifica formale e automatica dei protocolli, ad esempio nella Sezione 6 viene presentato un software, da me sviluppato, in grado di restituire come output un file scritto nel linguaggio specifico per essere utilizzato come input dal tool di verifica automatica VerifPal.

Figura 5: Esempio di modellazione con object diagram

```
<packagedElement
  xmi:type="uml:Package"
  xmi:id="_6KYCwF_CEeuUS4KSsiMLRg"
  name="ExampleThesis
>
    <packagedElement
      xmi:type="uml:InstanceSpecification"
      xmi:id="_6KYCwV_CEeuUS4KSsiMLRg"
      name="generateTestMessage"
      classifier="_6KWNiV_CEeuUS4KSsiMLRg"
    />
    <packagedElement
      xmi:type="uml:InstanceSpecification"
      xmi:id="_6KYCwl_CEeuUS4KSsiMLRg"
      name="OutPort"
      classifier="_6KWNqV_CEeuUS4KSsiMLRg"
    />
    <packagedElement
      xmi:type="uml:InformationFlow"
      xmi:id="_6KYCw1_CEeuUS4KSsiMLRg"
      name="message"
      informationSource="_6KYCwV_CEeuUS4KSsiMLRg"
      informationTarget="_6KYCwl_CEeuUS4KSsiMLRg"
    />
</packagedElement>
```

Figura 6: Esempio di estrazione di un file .xmi

3 Il modello di Attaccante Dolev-Yao

In [DY83], gli autori hanno proposto un modello, noto anche come modello di attaccante Dolev-Yao, per la formalizzazione degli attaccanti, questo modello viene utilizzato dai tool per la verifica formale dei protocolli basati sul modello simbolico, come vedremo con ProVerif e VerifPal.
Con questo tipo di modello si assume che l'attaccante sia 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 cifrati solo se a conoscenza della chiave di cifratura.

L'attaccante di Dolev-Yao è un attaccante attivo in grado di osservare, modificare ed eliminare i messaggi dalla rete, questo gli consente di utilizzare le regole descritte nella Figura 7 per effettuare degli attacchi.
Nel modello Dolev-Yao si assume che la crittografia sia perfetta, ovvero che i meccanismi crittografici non siano vulnerabili, ad esempio l'attaccante non può ottenere il testo in chiaro da un testo cifrato senza essere a conoscenza della chiave corretta per la decifratura, rendendo questo tipo di attaccante utile solo per attacchi alla logica dei protocolli.
Grazie alle sue abilità, ai fini della verifica formale dei protocolli è utile identificare questo tipo di attaccante come la rete stessa.

3.1 Primitive nel modello Dolev-Yao

Considerando un attaccante attivo nel modello standard di Dolev-Yao è possibile definire delle regole per il suo comportamento, come fatto in [RVV17].
Sia $\mathcal{S}$ un sistema e $\mathcal{S}_{\mathcal{DY}}$ un sistema in presenza di attaccante del modello Dolev-Yao, sia IK un insieme di messaggi e DY(IK) il più piccolo insieme chiuso rispetto le regole del sistema $\mathcal{S}_{\mathcal{DY}}$ di generazione (G) e di analisi (A).
I messaggi vengono indicati con il simbolo M e vengono utilizzati per definire le seguenti operazioni:

- $\{M_1\}_{M_2}$ rappresenta la cifratura asimmetrica di M_1 con la chiave pubblica M_2;
- $\{M_1\}_{inv(M_2)}$ rappresenta la cifratura asimmetrica di M_1 con la chiave privata $inv(M_2)$;
- $\{|M_1|\}_{M_2}$ rappresenta la cifratura simmetrica di M_1 con la chiave simmetrica M_2;
- $[M_1, M_2]$ rappresenta la concatenazione di M_1 e M_2.

La regola G consente all'attaccante di generare nuovi messaggi a partire dai messaggi conosciuti, attraverso la concatenazione e utilizzando la crittografia simmetrica e asimmetrica, la regola A descrive come l'attaccante può decomporre i messaggi.

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

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

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

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

Figura 7: Sistema delle regole $\mathcal{S}_{\mathcal{DY}}$ per l'attaccante Dolev-Yao

3.2 Le primitive nella modellazione UML

Le primitive descritte nella Figura 7 vengono rappresentate anche nella modellazione del protocollo attraverso l'utilizzo dei diagrammi UML.
La primitiva G_{concat} (concatenazione) appare nei diagrammi UML quando abbiamo più input in ingresso ad un oggetto con un singolo output, ad esempio viene utilizzata quando in un oggetto viene preparato un pacchetto, dati vari parametri in ingresso vengono concatenati prima di essere inoltrati.
Al contrario la primitiva A_{concat} (deconcatenazione) appare quando abbiamo un singolo input e in uscita più output, come quando abbiamo in ingresso un pacchetto e dobbiamo ricavare qualche elemento da cui è composto.

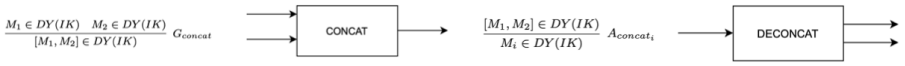


Figura 8: Concatenazione e deconcatenazione

Nella modellazione UML gli oggetti di encryption e decryption sono considerati delle black box, questo perché in caso di crittografia simmetrica corrispondono alle primitive G_{critS} e A_{critS} e in caso di crittografia asimmetrica corrispondono alle primitive G_{crittA} e A_{crittA} del modello Dolev-Yao.

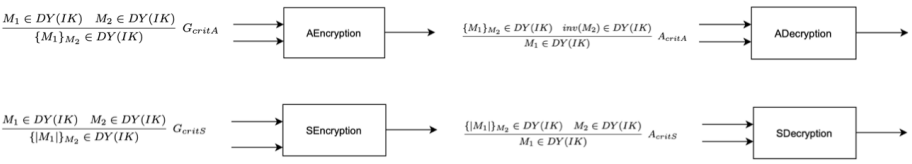


Figura 9: Encryption e decryption a chiave simmetrica e asimmetrica

Nel caso in cui nel diagramma sia rappresentato un oggetto per la firma di un messaggio o per la verifica della firma, ancora una volta troviamo nel modello Dolev-Yao le primitive necessarie, ovvero la primitiva G_{crittA}, la quale si occupa della firma di un messaggio prendendo come input la chiave privata e il

messaggio, e la primitiva A^{-1}_{crittA}, la quale verifica la firma utilizzando la chiave pubblica.

Figura 10: Verifica della firma

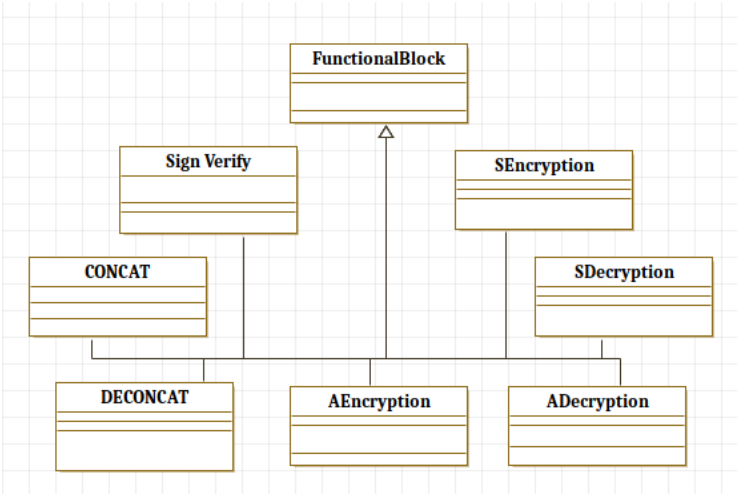


Figura 11: Diagramma delle classi delle primitive

4 Analisi dello stato dell'arte

La sicurezza delle informazioni viene definita utilizzando tre concetti di base: confidenzialità, integrità e disponibilità, conosciuti anche con l'acronimo CIA (Confidentiality, Integrity e Availability).
Per confidenzialità si intende la proprietà che impedisce a individui, entità o processi non autorizzati di accedere alle informazioni, per integrità si intende la proprietà dei dati che ne vieta l'alterazione accidentale o eseguita da una terza parte malevola, comprendendo il caso limite di generazione ex novo di dati, infine per disponibilità si intende la capacità di accedere in qualsiasi momento ai dati richiesti.
Il solo utilizzo di queste tre proprietà è stato spesso criticato perché ritenuto troppo generale per essere effettivamente applicato durante l'ingegnerizzazione di sistemi.
Per questo motivo molti ricercatori e organizzazioni hanno cercato di estendere la CIA con altre proprietà, ad esempio con l'autenticazione utilizzata proprio nelle proprietà di confidenzialità e integrità.

L'estensione della CIA negli anni è documentata nell'articolo [SC14] ed è riassunta nella Tabella 1.

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

Tabella 1: Evoluzione della CIA

Per raggiungere obiettivi di sicurezza, come confidenzialità o autenticazione dei dati nella rete, anche in presenza di attaccanti, si sono sviluppati i protocolli di sicurezza, questi ultimi descrivono come gli agenti si scambiano messaggi utilizzando le primitive crittografiche.
Effettuare dei test di sicurezza è complesso, soprattutto quando si devono testare dei protocolli, per questo motivo l'utilizzo di tool automatici per la verifica può essere utile per ottenere garanzie sulla correttezza dei protocolli di sicurezza.
Grazie agli sforzi della ricerca per trovare nuovi protocolli sempre più sicuri, al giorno d'oggi è possibile trovarli implementati nella maggior parte delle applicazioni commerciali.
Sin dal 1976 con la ricerca di Diffie-Hellman si è cercato un modo per dimostrare che un protocollo soddisfacesse determinate proprietà di sicurezza, per fare questo dalla ricerca di Dolev-Yao nel 1983 sono nate due tecniche per testare la correttezza dei protocolli.
La prima tecnica è conosciuta come modello computazionale (o crittografico), mentre la seconda tecnica è conosciuta come modello simbolico (o metodo formale).
Queste due tecniche si basano sulla matematica formale per l'esecuzione di protocolli in ambienti con la presenza di uno o più attaccanti, l'obiettivo è quello di definire formalmente le proprietà di sicurezza previste dal sistema crittografico e sviluppare metodi per dimostrare rigorosamente la soddisfazione delle stesse.
Le caratteristiche principali del modello computazionale sono i modelli per l'esecuzione del sistema dettagliati a livello di bit e l'utilizzo di un attaccante potente, la sicurezza viene valutata rispetto a macchine probabilistiche a tempo polinomiale e questo fa si che, in caso di esito positivo, le dimostrazioni di sicurezza forniscano delle garanzie.
In questo modello, la lunghezza delle chiavi è determinata da un valore chiamato security parameter, e la durata dell'esecuzione dell'avversario dovrebbe essere polinomiale rispetto al security parameter.
Tuttavia utilizzando questo modello anche su protocolli di piccole dimensioni le dimostrazioni di sicurezza sono molto lunghe, difficili e soggette ad errori.
Nel modello simbolico si utilizza una visione astratta dell'esecuzione del protocollo in quanto i messaggi scambiati dai partecipanti sono termini simbolici e il modello di attaccante Dolev-Yao.
Questo fa si che la modellazione di un protocollo con il modello simbolico sia più semplice rispetto a quella con il modello computazionale e che anche le di-

mostrazioni di sicurezza siano più semplici, anche se l'alto livello di astrazione rende poco chiare le garanzie di sicurezza offerte.
A causa del diverso insieme di strumenti e tecniche, i due modelli hanno convissuto e si sono sviluppati in modo indipendente per molti anni.
Negli anni si è cercato di ridurre il gap tra i due modelli, i primi sono stati Abadi e Rogway [AR00], i quali hanno svolto ricerche sulla correttezza del modello computazionale chiedendosi sotto quali condizioni i messaggi simbolicamente equivalenti, intercettati da un attaccante passivo, sono anche equivalenti computazionalmente.
La loro ricerca ha portato ad ulteriori ricerche nell'ambito della correttezza del modello computazionale, riassunte nell'articolo di Cortier, Kremer, Warinschi[CKW11], nel quale gli autori ripercorrono le varie tappe della ricerca da quella di Abadi e Rogway allo sviluppo del tool CryptoVerif di Blanchet.
Nella Figura 12 è possibile vedere come si è sviluppata la ricerca e quali sono le possibili strade future da intraprendere per nuovi studi.
In [Bla12] Bruno Blanchet afferma che, allo stato dell'arte, la verifica dei protocolli attraverso il modello simbolico abbia raggiunto un buon livello di maturità, anche se alcuni aspetti richiedono ulteriori ricerche.
Inoltre il modello simbolico si presta meglio ad essere automatizzato, a differenza del modello computazionale che fino a poco tempo fa richiedeva che le verifiche fossero fatte a mano (un esempio di tool automatico non ancora completo è il sopracitato CryptoVerif).
Per questo motivo da questo momento in poi verranno trattati solo tool per la verifica automatica di protocolli che utilizzano il modello simbolico.

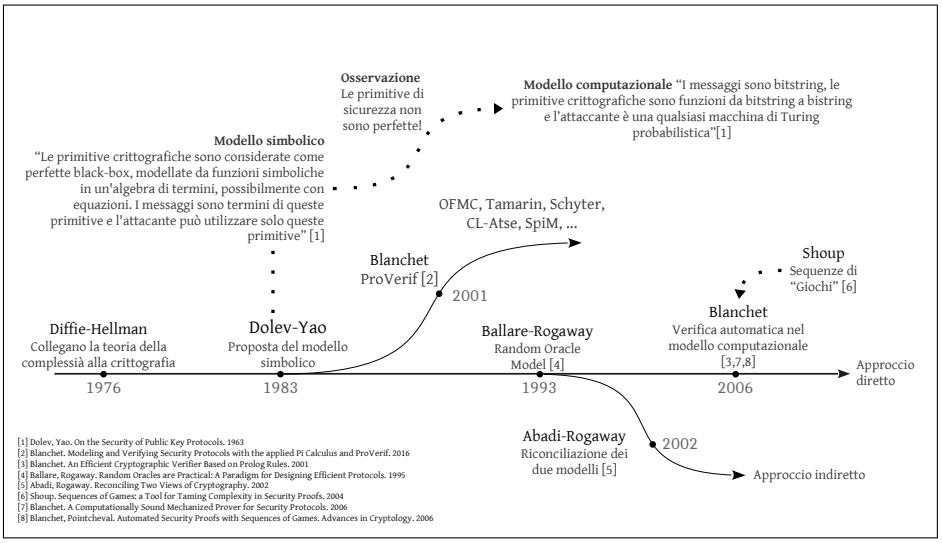


Figura 12: Evoluzione della ricerca negli anni

5 Tool per la verifica di protocolli basati sul modello simbolico

Sviluppare un tool per la verifica automatica di protocolli basato sul modello simbolico resta comunque una sfida, questo perché lo spazio di stati da esplorare è potenzialmente infinito per due motivi:

1. la dimensione dello spazio dei messaggi non è definita in presenza di un attaccante attivo,
2. il numero di sessioni del protocollo non è limitato.

Una semplice soluzione a questo problema è quella di esplorare solo una parte dello spazio degli stati, limitando arbitrariamente sia la dimensione dello spazio dei messaggi che il numero di sessioni del protocollo.
Solo se il numero di sessioni è limitato, la verifica dei protocolli rimane decidibile: l'insicurezza del protocollo (esistenza di un attacco) è NP-completa con ipotesi ragionevoli sulle primitive crittografiche.
Nonostante questa indecidibilità, molte tecniche sono state progettate per verificare protocolli con un numero illimitato di sessioni, limitandosi a sottoclassi di protocolli, richiedendo l'interazione dell'utente, tollerando la non terminazione, o con sistemi incompleti (che possono rispondere "Non lo so").
In seguito verrano analizzati i tool per la verifica automatica dei protocolli basati sul modello simbolico ProVerif e VerifPal.

5.1 ProVerif

Per la descrizione e l'analisi dei protocolli di sicurezza il tool ProVerif utilizza un linguaggio chiamato Applied Pi Calculus, un'estensione di Pi Calculus, il quale consente di dettagliare le azioni dei partecipanti al protocollo concentrando l'attenzione sulla loro comunicazione, più in generale consente di modellare processi concorrenti e la loro interazione.
A differenza del Pi Calculus vengono utilizzati termini al posto dei nomi per i messaggi.
Applied Pi Calculus [ABF16] è un linguaggio fortemente tipato che aggiunge a Pi Calculus l'algebra utile a modellare le operazioni crittografiche utilizzate dai protocolli di sicurezza mediante una teoria equazionale (ad esempio l'operazione di modulo utilizzata nella generazione delle chiavi).
Inoltre consente di definire manualmente le primitive di sicurezza a differenza di un'altra estensione chiamata SPi Calculus [AG97], dove le primitive crittografiche come encryption e decryption sono implementate internamente.

Modellazione con ProVerif

La modellazione di un protocollo in un file da dare in input a ProVerif può essere suddivisa in tre parti:

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

I processi sono composti da un insieme finito di tipi, nomi liberi e costruttori (funzioni simboliche) che sono associati ad un insieme finito di decostruttori.
Il linguaggio è fortemente tipato e l'utente può definire dei nuovi tipi in questo modo:

```
type t .
```

Tutti i nomi liberi devono essere dichiarati usando la sintassi:

```
free n : t .
```

dove n è un nome e t il tipo.
La sintassi per dichiarare un canale è:

```
free c:channel.
```

Tutte i nomi dichiarati con free sono conosciuti dall'attaccante, per fare in modo che non siano conosciuti dall'attaccante vanno dichiarati come privati utilizzando questa sintassi:

```
free n : t [ private ].
```

I costruttori (funzioni simboliche) sono usati per costruire termini di modellazione di primitive, usati dalla crittografia dei protocolli, per esempio: funzioni di hash one-way, cifratura e firme digitali.
I costruttori si definiscono con:

fun $f(t_1, \ldots, t_n)$: t .

dove f è un costruttore di arietà n, t è il tipo dell'oggetto di output e $t_1, \ldots, t_n$ sono i tipi degli argomenti di input.
Anche i costruttori sono conosciuti dall'attaccante a meno della dichiarazione utilizzando [private].
I decostruttori vengono utilizzati per manipolare i termini formati dai costruttori e catturano le relazioni tra le primitive crittografiche, si modellano usando regole di riscrittura della forma:

reduc forall $x_{1,1} : t_{1,1}, \ldots, x_{1,n_1} : t_{1,n_1}; g(M_{1,1}, \ldots, M_{1,k}) = M_{1,0};$

dove g è un decostruttore di arietà k, i termini $M_{1,1}, \ldots, M_{1,k}, M_{1,0}$ sono costruiti dall'applicazione del costruttore alle variabili $x_{1,1}, \ldots, x_{1,n_1}$ di tipo $t_{1,1}, \ldots, t_{1,n_1}$ rispettivamente ed il tipo dell'output è $M_{1,0}$.
Analogamente ai costruttori, i decostruttori possono essere dichiarati privati con l'aggiunta di [private].
Vediamo come i costruttori e decostruttori vengono utilizzati per la definizione manuale delle primitive crittografiche:

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

Macro Processi

Per semplificare lo sviluppo i sotto-processi vengono dichiarati utilizzando macro della forma: let $R(x_1 : t_1, \ldots, x_n : t_n) = P.$, dove R è il nome della macro, P è il sotto-processo che si vuole definire e $x_1, \ldots, x_n$ sono le variabili libere di P di tipo $t_1, \ldots, t_n$

Processi

$M, N ::=$	*termini*
$a, b, c, \ldots, k, \ldots, m, n, \ldots, s$	*nomi*
x, y, z	*variabili*
$h(M_1, \ldots, M_k)$	*applicazione di costruttore/decostruttore*
$M = N$	*uguaglianza tra termini*
$M <> N$	*disuglianza tra termini*
$M \&\& N$	*congiunzione*
$M \|\| M$	*disgiunzione*
not$M$	*negazione*

Tabella 2: Grammatica di base

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

Tabella 3: Grammatica dei processi

Nella Tabella 2 è descritta la grammatica di base dove i termini M, N consistono in nomi a, b, c, k, m, n, s, variabili x, y, z, tuple $(M_1, \ldots, M_l)$ dove l è l'arietà della tupla, funzioni simboliche (costruttori/decostruttori) indicati con $h(M_1, \ldots, M_k)$ dove k è l'arietà di h e gli argomenti $M_1, ..., M_k$ sono del tipo richiesto.
Le altre funzioni utilizzano la notazione infissa e lavorano con l'algebra booleana.
Nella Tabella 3 è descritta la grammatica dei processi dove il processo vuoto 0 non fa nulla, $P|Q$ è la composizione parallela di P e Q, la replicazione $!P$ si comporta come un numero infinito di copie di P in esecuzione in parallelo.
new $n : t; P$ lega il nome n del tipo t a P.
Il costrutto condizionale $if\ M = N\ then\ P\ else\ Q$ è standard e $M = N$ rappresenta l'uguaglianza, non l'identità.
Il processo in$(M, x : t); P$ indica che il processo P è in attesa di un messaggio x di tipo t dal canale M e poi prosegue la sua esecuzione.

Il processo $\text{out}(M, N); P$ indica che il processo P è pronto per inviare N nel canale M e proseguire con la sua esecuzione.
Per evitare ambiguità durante l'esecuzione dei processi è consigliato utilizzare le parentesi per ordinarli.

Proprietà di sicurezza

Per verificare la segretezza di un termine M in un modello è sufficiente inserire la seguente query prima del processo principale:

```
query attacker(M) .
```

dove M è un nome di solito dichiarato privato, se così non fosse l'attaccante ne sarebbe banalmente a conoscenza.
Per verificare la proprietà di autenticazione è necessario annotare i processi con degli eventi in alcuni stati importanti, che non influiscono sul comportamento del protocollo.
Gli eventi verranno utilizzati per chiedere al tool se un determinato evento ha avuto luogo prima di un altro.
La sintassi per una richiesta di autenticazione è la seguente:

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

Ipotizzando che l'evento e_0 sia l'accettazione di un determinato client da parte di un server e l'evento e_1 sia l'invio di un messaggio dal client al server, con questo tipo di query ci chiediamo se il messaggio destinato al server è effettivamente stato inviato dal client corretto, nel caso in cui l'evento e_0 si sia effettivamente verificato prima dell'evento e_1, possiamo dire con certezza che il client è autenticato.
Ecco un esempio di come viene modellato in Applied Pi Calculus l'invio di un messaggio dall'agente A all'agente B e l'attaccante che prova ad intercettarlo:

```
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
```

5.2 VerifPal

Tra i vari tool basati sul modello simbolico in grado di effettuare l'analisi formale dei protocolli di sicurezza, attualmente, il più utilizzato è ProVerif, purtroppo questo tool richiede all'utilizzatore di conoscere la sintassi del linguaggio Applied Pi Calculus, la quale è poco intuitiva, di conoscere il funzionamento delle clausole Horn create implicitamente dal tool e delle clausole Horn che devono

essere esplicitate per modellare in maniera corretta lo scenario in cui verificare il protocollo.
Per questi motivi è stato sviluppato un nuovo tool chiamato VerifPal.
L'obiettivo di VerifPal è quello di descrivere protocolli con un linguaggio molto simile a come i protocolli potrebbero essere descritti in una conversazione informale.
Per fare ciò, pur basandosi internamente su costruzione e decostruzione di termini astratti simili a ProVerif, fa si che l'utente debba solo definire gli agenti partecipanti al protocollo che hanno stati indipendenti, conoscono determinati valori ed eseguono operazioni con le primitive crittografiche.
Le primitive crittografiche sono già definite e devono essere solo richiamate, in questo modo si evita che l'utente possa implementarle in maniera errata.
VerifPal è nato con lo scopo di creare un tool utilizzabile nell'ingegneria, per questo si presta ad essere utilizzato per il raggiungimento dell'obiettivo principale di questo documento, ovvero quello di utilizzare la modellazione UML per avvicinare la verifica di protocolli all'ingegneria di sistemi.

Modellazione con VerifPal

Quando si scrive il codice per la verifica formale di protocolli da analizzare con il tool VerifPal, la prima cosa da fare è scegliere se l'attaccante è di tipo attivo o di tipo passivo.
L'attaccante di tipo attivo viene comunemente chiamato attaccante di tipo Dolev-Yao (Sezione 3), mentre l'attaccante di tipo passivo è semplicemente un ascoltatore in grado di vedere i pacchetti che transitano sulla rete.
La seconda cosa da fare è inizializzare gli agenti che partecipano al protocollo, chiamati `principal`[7], all'interno della definizione dei principal possono essere definite delle costanti.
Queste costanti possono essere conoscenze pregresse del principal, per fare questo vengono inizializzate con la parola chiave `knows`, oppure possono essere delle costanti con dei valori generati al momento, in questo caso si utilizza la parola chiave `generates`.
Inoltre, le costanti possono essere definite pubbliche o private, nel caso in cui la costante sia definita pubblica anche l'attaccante ne è a conoscenza.
Per quanto detto sopra, per evitare errori da parte dell'utente, le variabili hanno uno scope globale, di conseguenza non possono esistere due costanti con lo stesso nome, a meno che non vengano definite come `private` e non possono essere riassegnate ad altri valori.
Una volta inizializzati i principal si passa alla modellazione del protocollo vero e proprio, con lo scambio dei messaggi e le varie operazioni fatte dai principal.
Per inviare un messaggio basta semplicemente indicare mittente, destinatario e contenuto del messaggio in questo modo:

A→B: m

Inoltre è possibile forzare il fatto che l'attaccante attivo non possa modificare il messaggio m semplicemente inserendolo tra [ ], l'utilizzo di questa tecnica di guardia è sconsigliato, in quanto potrebbe alterare il risultato della verifica, se non utilizzata correttamente.

[7] questo tipo di font verrà applicato a tutte le keyword utilizzate da VerifPal

Come detto sopra VerifPal ci viene incontro definendo le seguenti primitive (le quali corrispondono anche alle primitive del modello Dolev-Yao):

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

Oltre a queste primitive VerifPal fornisce altre primitive utili per effettuare l'hash dei messaggi.
Il modello del protocollo si conclude con un blocco chiamato `queries`, dove sono contenute le domande alle quali vorremmo che VerifPal ci desse le risposte come risultato dell'analisi del modello.
Nel blocco di queries possiamo fare delle domande su confidenzialità, autenticazione, freshness dei messaggi utilizzando rispettivamente i comandi `confidentiality?`, `authentication?` e `freshness?`.
Inoltre nel caso in cui si volesse modellare uno scenario in cui l'attaccante riesce ad ottenere una costante dichiarata come privata in qualche modo, basta utilizzare la parola chiave `leaks` seguita dal nome della variabile.
Qui è proposto lo stesso scenario modellato nella Sezione 5.1 con ProVerif:

```
attacker[active]

principal Initiator[
        knows private m
]

principal Recipient[]

Initiator -> Recipient : m

queries[
        confidentiality? m
]
```

6 Tool di conversione

In questa sezione viene presentato il tool in grado di convertire il file .xmi estratto con Modelio, in un file adatto all'utilizzo del tool di verifica automatica VerifPal.
La scelta di creare un file utilizzabile da VerifPal è motivata dal fatto che è più semplice e intuitivo riuscire a trovare delle relazioni tra il file .xmi e i blocchi che compongono il modello del protocollo nel linguaggio di questo tool rispetto al linguaggio Applied Pi Calculus utilizzato da ProVerif.
Il tool si può scomporre in due parti, nella prima si occupa di generare un file intermedio, a partire dal file .xmi, contenente delle strutture in grado di rappresentare le relazioni tra i blocchi e le InformationFlow della modellazione tramite UML.
Mentre nella seconda parte viene utilizzato il file intermedio generato nella prima parte per andare a generare un template, il quale consente al progettista di inserire il tipo dell'attaccante (attivo o passivo) e quali sono le verifiche che vuole effettuare sul protocollo nel blocco di queries, prima di utilizzare il file come input a VerifPal.
La Figura 13 rappresenta la modellazione della prima parte del tool attraverso un flowchart, possiamo vedere come dato il file .xmi come input, il tool trasformi l'input in un albero in modo da poterlo navigare ed estrarne le informazioni rilevanti.
Partendo dalla radice il tool estrae i vari tipi degli oggetti istanziati nel modello UML, tra i vari tipi troviamo InputPort, OutPort, FunctionalBlock, FunctionalArchitecture, InformationFlow.
Dopo aver estratto i tipi il tool si occupa di verificare se sono presenti degli oggetti di tipo Node, di solito utilizzati per la rappresentazione fisica dei dispositivi.
Nel caso in cui sia presente un oggetto di tipo Node naviga i suoi attributi per estrarre le informazioni sugli oggetti che lo compongono, nello specifico estrae id, nome e tipo dell'oggetto che compone l'oggetto di tipo Node e salva queste informazioni in un dizionario.
Il passo successivo del tool è quello di vedere se vi sono oggetti di tipo Instance, in questo caso, come per gli oggetti che compongono il Node, estrae id, nome e tipo dell'oggetto e li salva nello stesso dizionario.
Infine il tool cerca le InformationFlow, per ognuna di queste salva in un dizionario id dell'oggetto sorgente, id dell'oggetto destinazione e le informazioni sul messaggio.
La particolarità del dizionario per le InformationFlow è che oltre alle informazioni estratte viene mantenuto anche un contatore, il contatore sarà utile nella seconda parte del tool per mantenere l'ordine dell'interazione tra gli agenti partecipanti al protocollo.
Una volta terminata la costruzione dei dizionari il tool si occupa di andarli a scrivere sul file che verrà utilizzato nella seconda parte.

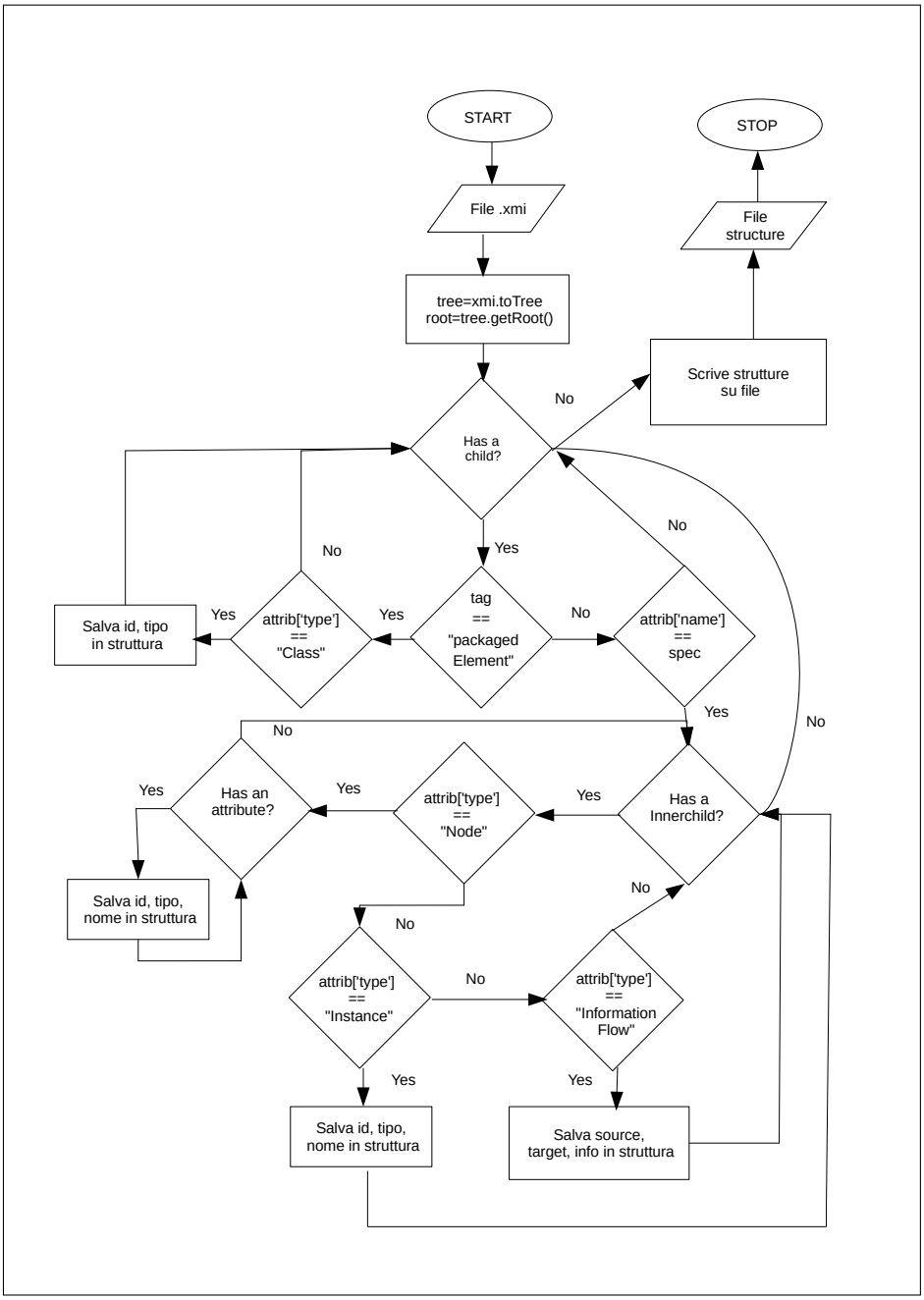


Figura 13: Flowchart modellazione prima parte del tool

Nella Figura 14 è rappresentata l'estrazione dei dizionari dell'esempio di modellazione UML in Figura 5.

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

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

Figura 14: Esempio di estrazione delle strutture da parte del tool

La figura 15 rappresenta la modellazione tramite flowchart della seconda parte del tool, dove dato in input il file contenente le strutture viene restituito il template per VerifPal.
Il funzionamento della seconda parte del protocollo consiste nello scorrere il dizionario delle InformationFlow, per ogni elemento nel dizionario si utilizza l'id della destinazione per verificare nel dizionario degli oggetti se la destinazione è di tipo OutputPort, in questo caso il tool si trova di fronte ad un invio di messaggio ad un secondo principal, questo viene convertito nella stringa adatta `A -> B : messaggio`.
Nel caso in cui l'oggetto di destinazione non sia una porta di output significa che le informazioni dell'InformationFlow sono un input per un FunctionalBlock, si utilizza il nome dell'oggetto per mapparlo in una primitiva di VerifPal e passargli le varie informazioni come argomenti.
Per mappare i nomi degli oggetti con le primitive all'interno del tool è presente un dizionario che mappa le primitive di VerifPal con i nomi degli oggetti.
Questo tipo di mappatura tra nomi e primitive richiede una standardizzazione dei nomi degli oggetti che deve essere fatta in fase di modellazione dei diagrammi UML.
La standardizzazione consiste nel definire dei blocchi di oggetti standard, che rappresentano le primitive, i quali devono semplicemente essere utilizzati dal progettista, in modo che non sia lui stesso a crearli con dei nomi che non trovano riscontro all'interno del dizionario di mappatura utilizzato dal tool.
A livello di creazione del template si può dire che tutti gli oggetti di tipo InformationFlow, compresi tra due InformationFlow con destinazione oggetti di tipo OutputPort, corrispondono alle operazioni fatte da un principal.
Quando il tool si trova di fronte ad oggetti che hanno solo informazioni in uscita, si trova di fronte a blocchi in cui le informazioni possono essere conoscenze pregresse del pricipal oppure informazioni generate al momento, in questo caso è in grado di distinguere se deve utilizzare la parola chiave `knows` oppure `generates`.

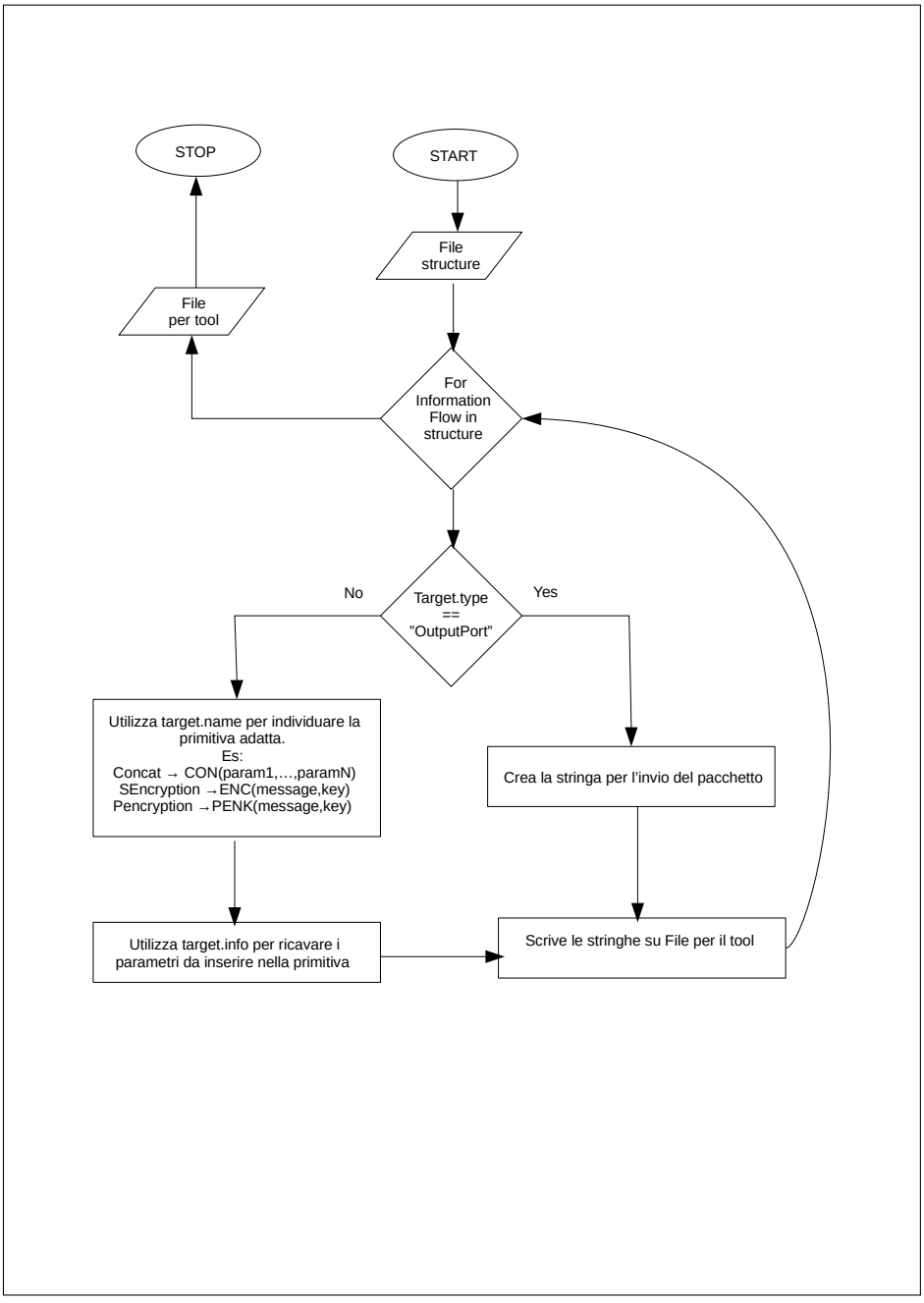


Figura 15: Flowchart modellazione seconda parte del tool

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

principal A[
        knows private message
]

principal B[]

A -> B : message

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

Figura 16: Esempio di template per VerifPal

In Figura 16 vediamo il template ottenuto nella seconda parte del tool partendo dal file delle strutture descritto in Figura 14.

7 Casi d'uso

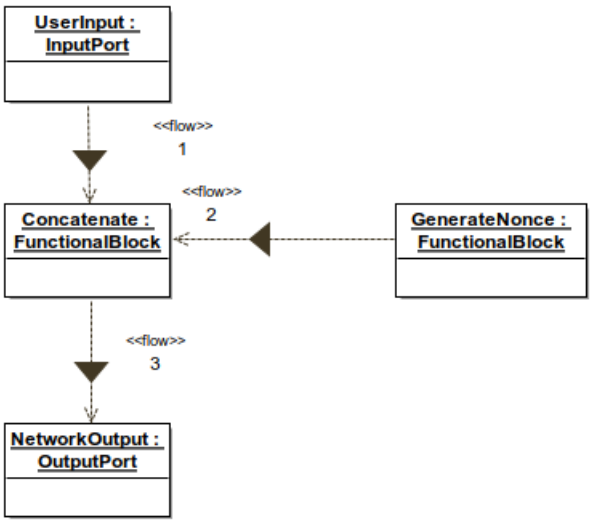
In questa sezione vedremo come vengono modellati i protocolli tramite diagrammi UML, come il tool di conversione utilizza i file .xmi per arrivare a generare dei file da dare in input a VerifPal e quali sono le risposte dei risultati dell'analisi.
I protocolli utilizzati come esempio sono il protocollo di Needham e Schroeder a chiave simmetrica, il protocollo ARP e il crittosistema RSA.

7.1 Needham Schroeder Symmetric Key

In questa sezione vedremo come è possibile modellare attraverso i diagrammi object diagram dello standard UML il protocollo di sicurezza a chiave simmetrica proposto da Needham e Schroeder:

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. $A \rightarrow B : \{N_b - 1\}_{K_{as}}$

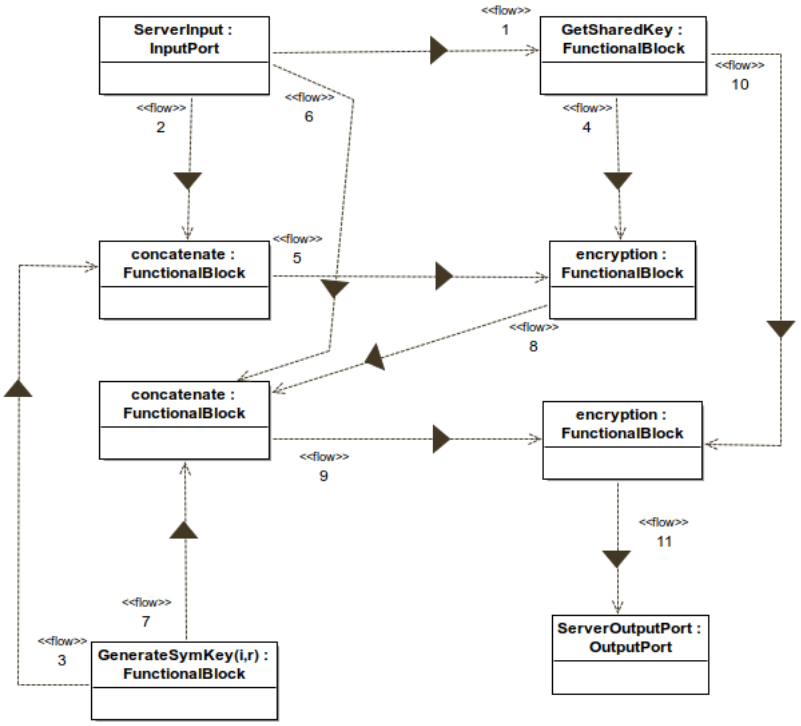
Nelle Figure seguenti avremo i seguenti partecipanti al protocollo: l'agente `Initiator` che vuole iniziare la comunicazione con l'agente `Recipient` e richiede la password per la comunicazione al server `S`.
Inoltre la chiave simmetrica viene rappresentata in questo modo `SK(`a`,`b`)`, dove a e b indicano l'identità degli agenti proprietari della chiave.



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

Figura 17: $A \rightarrow B : A, B, N_a$

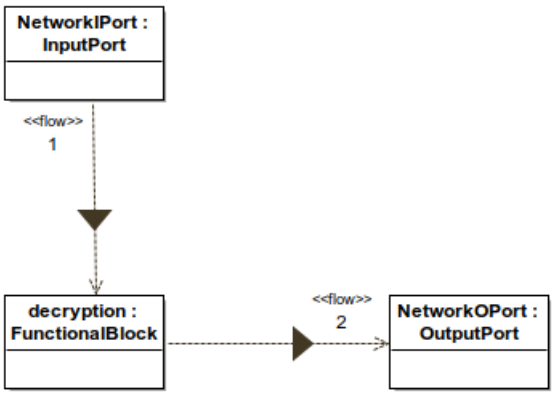
Nell'oggetto UserInput il sistema che andrà ad implementare il protocollo riceve il nome ($r.name$) dell'agente `Recipient`, l'oggetto GenerateNonce genera un nuovo Nonce e l'oggetto Concatenate prepara il pacchetto da mandare al server `S` attraverso l'oggetto NetworkOPort, composto da $i.name, r.name, i.nonce$, ovvero dai nomi dei partecipanti al protocollo e il nonce per assicurarsi che la comunicazione sia fresh.



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. $< \{i.nonce, r.name, SK(i,r), \{i.name, SK(i,r)\}\_SK(rs)\}\}\_SK(i,s) >$

Figura 18: $S \rightarrow A : \{N_a, K_{ab}, B, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

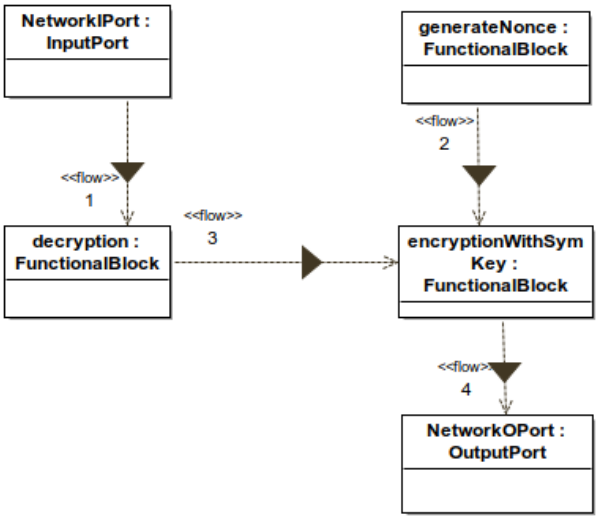
Una volta ricevuto il pacchetto, il server `S` provvede alla generazione della chiave simmetrica (`SK(`i`,`r`)`) per la comunicazione tra `Initiator` e `Recipient` utilizzando l'oggetto GenerateSymKey(i,r), passando a quest'ultimo i nomi $i.name, r.name$ dei partecipanti.
A questo punto, fornendo sempre come input i nomi dei partecipanti all'oggetto GetSharedKey, ottiene le chiavi simmetriche precedentemente condivise tra lui e ogni agente partecipante (`SK(`i`,`s`)` e `SK(`r`,`s`)`).
La chiave `SK(`r`,`s`)` verrà utilizzata dall'oggetto encryption dopo aver preparato con l'oggetto Concatenate il pacchetto per l'agente `Recipient`, questo pacchetto a sua volta verrà inserito da un altro oggetto Concatenate nel pacchetto per l'agente `Initiator` e il tutto verrà cifrato da un altro oggetto encryption con la chiave `SK(`i`,`s`)`. Il pacchetto risultante da queste operazioni verrà spedito all'agente `Initiator` attraverso l'oggetto ServerOutputPort.



1. $< \{i.nonce, r.name, SK(i,r), \{i.name, SK(i,r)\}\_SK(r,s)\}\_SK(i,s) >$
2. $< \{i.name, SK(i,r)\}\_SK(r,s) >$

Figura 19: $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$

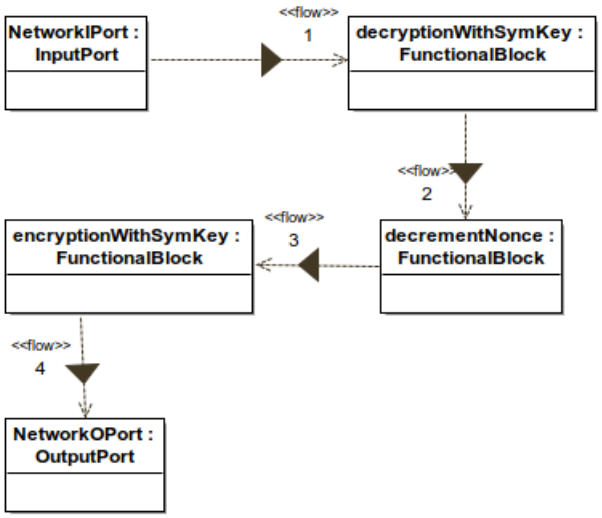
L'agente `Initiator` riceve il pacchetto attraverso l'oggetto NetworkIPort e utilizza l'oggetto di decryption con la chiave simmetrica `SK(`i`,`s`)` per estrarre il pacchetto da inoltrare all'agente `Recipient` attraverso l'oggetto NetworkOPort.



1. $< \{i.name, SK(i,r)\}\_SK(r,s) >$
2. $< r.nonce >$
3. $< SK(i,s) >$
4. $< \{r.nonce\}\_SK(i,r) >$

Figura 20: $B \rightarrow A : \{N_b\}_{K_{as}}$

L'agente `Recipient` riceve il pacchetto attraverso l'oggetto NetworkIPort, lo decifra con l'oggetto decryption utilizzando la chiave `SK(`r`,`s`)` ed estrae la chiave `SK(`i`,`r`)`.
`SK(`i`,`r`)` verrà utilizzata dall'oggetto encryptionWithSymKey per cifrare un nuovo pacchetto contenente il Nonce generato dall'oggetto generateNonce.
Infine l'agente `Recipient` spedisce il pacchetto all'agente `Initiator` utilizzando l'oggetto NetworkOPort.



1. $< r.nonce\_SK(i,r) >$
2. $< r.nonce >$
3. $< r.nonce - 1 >$
4. $< \{r.nonce - 1\}\_SK(i,r) >$

Figura 21: $A \rightarrow B : \{N_b - 1\}_{K_{as}}$

Nell'ultima fase del protocollo, l'agente `Initiator` riceve dall'oggetto NetworkIPort il pacchetto contenente il Nonce, lo decifra con l'oggetto decryptionWithSymKey utilizzando la chiave `SK(`i`,`r`)`, utilizza l'oggetto decrementNonce per sottrarre 1 al Nonce inviato dall'agente `Recipient` e cifra il risultato con l'oggetto encryptionWithSymKey utilizzando la chiave `SK(`i`,`r`)`.
Il pacchetto risultante viene spedito all'agente `Recipient` attraverso l'oggetto NetworkOPort.

Tool di verifica

Dopo aver modellato il protocollo ed estratto il file .xmi visibile nel Listing 7[8], si utilizza il tool di conversione per ottenere il file delle strutture Listing 8[8]. Sempre con il tool di conversione ricaviamo il seguente file da utilizzare come input a VerifPal:

Listing 3: File nssk.vp

```
attacker[active]

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

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)
]
```

[8]Listing in Appendice A

```
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
]

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

In questo esempio vengono verificate la confidenzialità della chiave **k_ir**, la confidenzialità del Nonce **n_r_minus_one** e l'autenticazione dell'ultimo pacchetto del protocollo.
Nel Listing 9[9] troviamo il risultato dell'analisi.
Come dimostrato da Denning-Sacco in [DS81] il protocollo è soggetto ad attacchi di tipo reply attack, se l'attaccante viene a conoscenza della chiave condivisa tra i due principal.
In questo modello l'attaccante in qualche modo riesce ad ottenere la chiave e di conseguenza i risultati della verifica diranno che la chiave non è confidenziale,

[9]Listing in Appendice A

come non è confidenziale il Nonce **n_r_minus_one**, oltre al fatto che può essere lo stesso attaccante ad inviare il pacchetto **e_n_r_minus_one** al principal `Recipient` fingendosi il principal Initiator, violando l'autenticazione.
Per completezza in Appendice A vengono riportate la modellazione del protocollo in Applied Pi Calculus nel Listing 10 e i risultati ottenuti con ProVerif nel Listing 11, i quali rispecchiano i risultati ottenuti con VerifPal.

Osservazioni

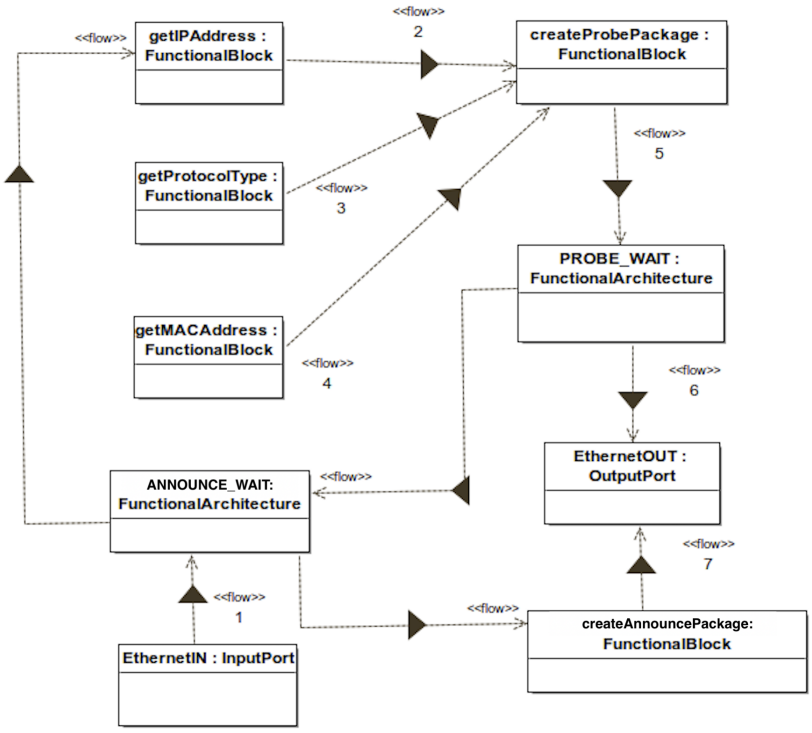
L'esecuzione del modello di questo protocollo consiste in una sola sessione dove l'attaccante è a conoscenza della chiave simmetrica tra i due principal, questo definisce uno scenario ben specifico infatti non è sempre vero che l'attaccante riesca ad ottenere la chiave simmetrica.
Con questa premessa si capisce come l'utilizzatore del tool per la verifica dovrebbe avere ben chiaro lo scenario in cui modellare il protocollo e allo stesso tempo dovrebbe avere almeno l'intuizione per capire sotto quali condizioni il protocollo può essere violato, per poterlo testare in diversi scenari.
Facendo vari tipi di verifiche sul protocollo è infatti emerso un limite dei verificatori automatici, ovvero il fatto di dover modellare il protocollo in uno specifico scenario per verificare la presenza di un determinato tipo di attacco, eventualmente già noto.
Per diversificare gli scenari, l'utilizzatore del tool può aumentare il numero di sessioni del protocollo, oppure fornire ad un certo punto dell'esecuzione del protocollo delle informazioni all'attaccante, come abbiamo visto in questo caso.
Queste tecniche consentono al tool di verifica di fornire delle risposte valide ai quesiti, evitando la non terminazione o risposte del tipo "Non lo so".

7.2 Address Resolution Protocol

Lo scopo del protocollo ARP descritto in [Plu82] e in [Che08] è quello di eseguire una mappatura tra indirizzo IP e indirizzo MAC di una macchina all'interno di una rete locale Ethernet.
La notazione seguente utilizzata nei pacchetti è ripresa da [Plu82]:

```
ar$hrd: Hardware address space
ar$pro: Protocol address space
ar$hln: byte length of each hardware address
ar$pln: byte length of each protocol address
ar$op:  opcode (request | reply)
ar$sha: Hardware address of sender
ar$spa: Protocol address of sender
ar$tha: Hardware address of target
ar$tpa: Protocol address of target
```

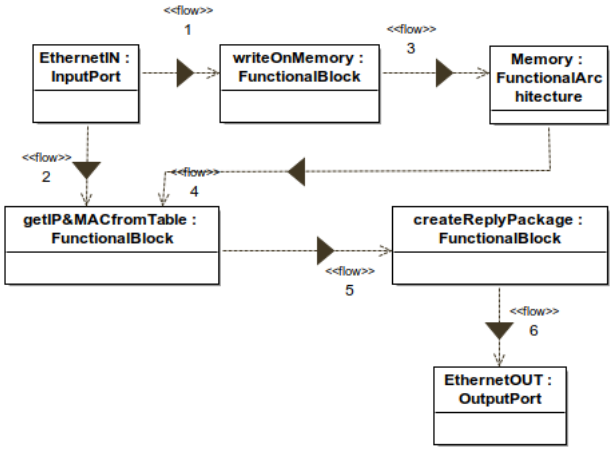
In Figura 22 vediamo come si modella il protocollo ARP.
Una macchina, appena connessa alla rete o accesa, si mette subito in ascolto con l'oggetto ANNOUNCE_WAIT e, allo stesso tempo, utilizza gli oggetti getIPAddress per generare un indirizzo IP sul quale essere contattata, getProtocolType e getMACAddress per ricavare informazioni sul tipo di protocollo ethernet da utilizzare e l'indirizzo MAC della sua scheda di rete.
A questo punto utilizza queste informazioni per creare attraverso l'oggetto createProbePackage un pacchetto da inviare in broadcast a tutte le macchine della rete.
Successivamente attende un tempo predefinito attraverso l'oggetto PROBE_WAIT, prima di spedire il pacchetto attraverso l'oggetto EthernetOUT e ritornare nello stato di ANNOUNCE_WAIT.
Se nell'arco di un tempo predefinito non arriva nessun pacchetto dall'oggetto EthernetIN, procede con la conferma dell'indirizzo IP attraverso la creazione di un nuovo pacchetto con l'oggetto createAnnouncePackage, il quale verrà sempre spedito in broadcast attraverso l'oggetto EthernetOUT.
Se invece riceve un pacchetto dall'oggetto EthernetIN, ricomincia generando un nuovo indirizzo IP.



1. $< \{as\$hrd, ar\$pro, ar\$hln, ar\$pln, reply, ar\$sha, ar\$spa, ar\$tha, ar\$tpa\} >$
2. $< ip >$
3. $< ar\$spa >$
4. $< ar\$sha >$
5. $< \{as\$hrd, ar\$pro, ar\$hln, ar\$pln, request, ar\$sha, 0.0.0.0, 00:00:00:00:00:00, ip\} >$
6. $< \{as\$hrd, ar\$pro, ar\$hln, ar\$pln, request, ar\$sha, 0.0.0.0, 00:00:00:00:00:00, ip\} >$
7. $< \{as\$hrd, ar\$pro, ar\$hln, ar\$pln, request, ar\$sha, ip, 00:00:00:00:00:00, ip\} >$

Figura 22: Modellazione del protocollo ARP

Nel caso in cui una macchina della rete riceva un pacchetto dall'oggetto EthernetIN (Figura 23), la prima cosa che fa è verificare se si tratta di un pacchetto di probe oppure di un pacchetto di announce.
Nel primo caso va a verificare con l'oggetto Memory se nell'ARP cache è già presente quell'indirizzo IP, assegnato ad un indirizzo MAC, che non corrisponde a quello del sender del pacchetto.
Se vi è una corrispondenza tra l'indirizzo ip del sender del pacchetto ed uno già presente nell'ARP cache, allora l'oggetto getIP&MACFromTable riceve in input le informazioni dalla ARP cache per generare un nuovo pacchetto attraverso l'oggetto createReplyPackage, con il quale rispondere in unicast al sender attraverso l'oggetto EthernetOut, altrimenti non fa nulla.
Nel caso in cui si tratti di un pacchetto di announce allora l'oggetto writeOnMemory si occuperà di salvare nella ARP cache la mappatura tra l'indirizzo IP e MAC del sender.



1. $< \{as\$hrd, ar\$pro, ar\$hln, ar\$pln, request, ar\$sha, ip, 00:00:00:00:00:00, ip\} >$
2. $< \{as\$hrd, ar\$pro, ar\$hln, ar\$pln, request, ar\$sha, 0.0.0.0, 00:00:00:00:00:00, ip\} >$
3. $< ar\$sha, ip >$
4. $< ar\$tha, ar\$tpa >$
5. $< ar\$tha, ar\$tpa >$
6. $< \{as\$hrd, ar\$pro, ar\$hln, ar\$pln, reply, ar\$sha, ar\$spa, ar\$tha, ar\$tpa\} >$

Figura 23: Modellazione del blocco di ARP Reply

Tool di verifica

Dopo aver modellato il protocollo ed estratto il file .xmi visibile nel Listing 12[10], si utilizza il tool di conversione per ottenere il file delle strutture Listing 13[10]. Per quanto già detto precedentemente, essendo il protocollo ARP suddivisibile in più scenari come l'invio del pacchetto di probe seguito dall'invio del pacchetto di announce oppure l'invio del pacchetto di probe seguito dall'invio del pacchetto di reply, una volta generato il template completo deve essere l'utilizzatore a suddividere i vari scenari per poi sottoporli alla verifica con Verifpal.
Per dividere i due scenari, l'utilizzatore deve generare due file da utilizzare come input in VerifPal, per fare questo è sufficiente copiare il template completo ed andare a cancellare le righe di generazione e invio del pacchetto di announce quando si vuole rappresentare lo scenario dell'invio del pacchetto di probe seguito dall'invio del pacchetto di reply, allo stesso modo per rappresentare lo scenario di invio di un pacchetto di probe seguito da un pacchetto di announce, l'utilizzatore non deve fare altro che cancellare le righe contenenti la creazione e l'invio del pacchetto di reply.

[10]Listing in Appendice B

Listing 4: File arp_announce.vp

```
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
]
```

Listing 5: File arp_reply.vp

```
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
]
```

Come ben noto, il protocollo ARP è un protocollo di rete che non si occupa dell'integrità dei pacchetti ed è vulnerabile ad un attacco chiamato ARP Poisoning, questo attacco consiste in un attacco del tipo Man In The Middle, dove l'attaccante modifica i pacchetti che attraversano la rete per andare a modificare con valori inesatti le ARP cache delle macchine connesse.

L'obiettivo della modifica della mappatura tra indirizzo MAC e indirizzo IP può essere attuare un Denial Of Service oppure fingersi una determinata macchina. Dato l'utilizzo dell'attaccante attivo da parte del tool VerifPal, come possiamo vedere dai Listing 14-15[11], l'attaccante riesce a violare sia la confidenzialità che l'integrità dei pacchetti, di conseguenza riesce ad attuare l'attacco di tipo ARP Poisoning.
Per confermare quanto appena detto in Appendice B vengono riportate la modellazione dello scenario di invio di un pacchetto di probe descritto in Applied Pi Calculus nel Listing 16 e il risultato della verifica svolta con ProVerif nel Listing 17.

7.3 Crittosistema RSA

Nel 1978 in [RSA78] Rivest, Shamir e Adleman hanno proposto un crittosistema per la cifratura e la firma dei messaggi basato sulla crittografia asimmetrica.
Come vedremo nelle Figure successive il crittosistema si basa su tre funzioni principali: funzione di generazione delle chiavi, funzione di encryption e funzione di decryption.
Nella modellazione UML ogni oggetto rappresenta le funzioni matematiche utilizzate dalle tre funzioni.
L'oggetto Memory viene utilizzato dalla funzione di generazione delle chiavi per salvare le informazioni riguardo la chiave pubblica e la chiave privata necessarie alle funzioni di encryption e decryption.

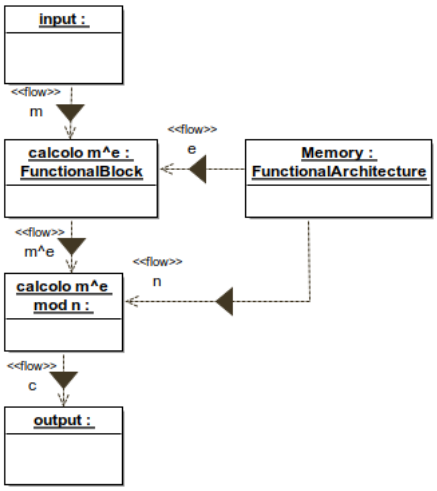


Figura 24: Funzione di encryption

[11]Listing in Appendice B

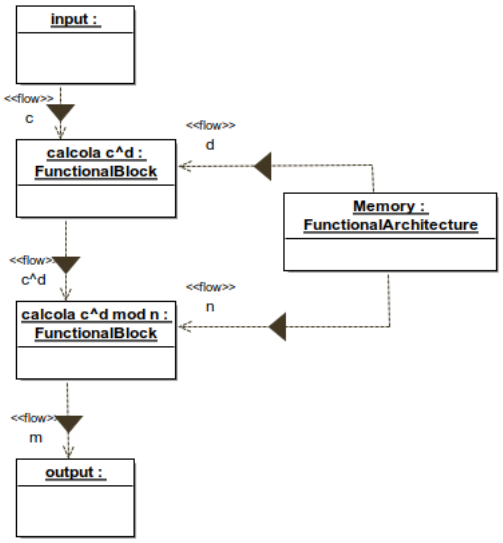


Figura 25: Funzione di decryption

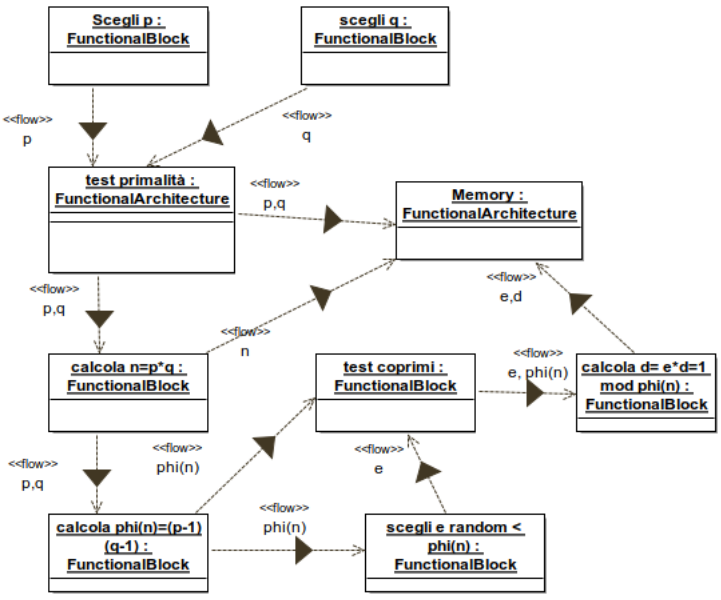


Figura 26: Generazione delle chiavi

Tool di verifica

A seguito della modellazione UML del crittosistema RSA è possibile utilizzare il file .xmi estratto, visibile nel Listing 18[12] per generare il file delle strutture Listing 19[12].
In questo caso ci troviamo però in un caso particolare per il tool di conversione perché il tool di analisi VerifPal utilizza un attaccante del modello Dolev-Yao, il quale non è in grado di rompere le primitive crittografiche, ed inoltre lo stesso VerifPal ci fornisce le primitive di encryption e decryption senza dover implementare nulla.
Per questo motivo deve essere l'utilizzatore a modellare il file da dare in input al tool di verifica a mano, andando ad ipotizzare uno scenario in cui viene utilizzato RSA:

Listing 6: File nssk.vp

```
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)
]
```

[12]Listing presentati in Appendice C

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

Come vediamo nel Listing 20[13] il risultato della verifica sulla confidenzialità del messaggio **m_b** viene preservata, questo perché è il risultato della decryption con la chiave privata del principal e non è mai transitato in chiaro nella rete, ma viene trovato un problema di autenticazione per il messaggio **e_2**, questo perché per come è stato modellato il protocollo, l'attaccante può fingersi il secondo partecipante al protocollo andando a creare una coppia chiave pubblica-privata e inviare la sua chiave pubblica al posto di quella del reale partecipante.
La stessa situazione si verifica con la modellazione in Applied Pi Calculus (Listing 21[13]) come si può vedere dai risultati ottenuti con ProVerif (Listing 22[13]).
Ovviamente cambiando il tipo di scenario, ad esempio facendo in modo che le chiavi pubbliche siano certificate da una Public Key Infrastructure, i due tool di verifica automatica non rileveranno nessun tipo di problema riguardo autenticazione e confidenzialità.
Un metodo semplice di modellare questo nuovo scenario è quello di utilizzare una guardia sulle chiavi pubbliche scambiate dai principal, in modo che l'attaccante non possa né leggerle, né modificarle:

```
Alice -> Bob : [ga]
Bob -> Alice : [gb]
```

[13]Listing presentati in Appendice C

8 Conclusioni

Abbiamo visto come la verifica formale dei protocolli di sicurezza sia un'operazione importante, da effettuare prima di utilizzare un protocollo all'interno di un'applicazione o di un software, per garantire la sicurezza dei dati e delle informazioni.
Inoltre abbiamo visto come allo stato dell'arte esistano il modello computazionale e il modello simbolico per la verifica formale automatica dei protocolli, ma solo il modello simbolico, il quale utilizza l'attaccante del modello Dolev-Yao, è abbastanza "maturo" per essere effettivamente utilizzato.
In questo documento è stato presentato un nuovo modo, intuitivo e agile, per modellare i protocolli mediante diagrammi UML, che può essere utilizzato dai progettisti ed è stata fatta un'analisi su limiti e capacità delle tecniche di modellazione attuali.
È stato presentato il software sviluppato appositamente per trasformare i protocolli modellati mediante diagrammi UML in un file utilizzabile come input per il tool di verifica automatica VerifPal.
I protocolli analizzati con VerifPal ci hanno consentito di analizzare limiti e capacità di questo tool di verifica automatica e di verificarne la bontà dei risultati di analisi.
Un obiettivo futuro può essere quello di 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.

A Needham-Schroeder a chiave simmetrica

Listing 7: XMI

```
<?xml version="1.0" encoding="UTF-8"?><uml:Model xmlns:uml="http://
    www.omg.org/spec/UML/20110701" xmlns:xmi="http://schema.omg.org
    /spec/XMI/2.1" xmi:version="2.1" xmi:id="
    _WmudYGlZEeunebkXFN4K9w" name="ns">
  <eAnnotations xmi:id="_WmudYWlZEeunebkXFN4K9w" source="Objing">
    <contents xmi:type="uml:Property" xmi:id="
        _WmudYmlZEeunebkXFN4K9w" name="exporterVersion">
      <defaultValue xmi:type="uml:LiteralString" xmi:id="
          _WmudY2lZEeunebkXFN4K9w" value="3.0.0"/>
    </contents>
  </eAnnotations>
  <packagedElement xmi:type="uml:Class" xmi:id="
      _WmudZGlZEeunebkXFN4K9w" name="Agent"/>
  <packagedElement xmi:type="uml:Class" xmi:id="
      _WmudZWlZEeunebkXFN4K9w" name="FunctionalBlock"/>
  <packagedElement xmi:type="uml:Class" xmi:id="
      _WmudZmlZEeunebkXFN4K9w" name="Port"/>
  <packagedElement xmi:type="uml:Class" xmi:id="
      _WmudZ2lZEeunebkXFN4K9w" name="InputPort"/>
  <packagedElement xmi:type="uml:Class" xmi:id="
      _WmudaGlZEeunebkXFN4K9w" name="OutputPort"/>
  <packagedElement xmi:type="uml:Class" xmi:id="
      _WmudaWlZEeunebkXFN4K9w" name="Aggregate"/>
  <packagedElement xmi:type="uml:Package" xmi:id="
      _WmudamlZEeunebkXFN4K9w" name="TEST_NS">
    <packagedElement xmi:type="uml:Node" xmi:id="
        _Wmuda2lZEeunebkXFN4K9w" name="Controller">
      <ownedAttribute xmi:type="uml:Port" xmi:id="
          _WmudbGlZEeunebkXFN4K9w" name="NetworkOPort" type="
          _WmudaGlZEeunebkXFN4K9w"/>
      <ownedAttribute xmi:type="uml:Port" xmi:id="
          _WmudbWlZEeunebkXFN4K9w" name="NetworkIPort" type="
          _WmudZ2lZEeunebkXFN4K9w"/>
      <ownedAttribute xmi:type="uml:Port" xmi:id="
          _WmudbmlZEeunebkXFN4K9w" name="UserOPort" type="
          _WmudaGlZEeunebkXFN4K9w"/>
      <ownedAttribute xmi:type="uml:Port" xmi:id="
          _Wmudb2lZEeunebkXFN4K9w" name="UserIPort" type="
          _WmudZ2lZEeunebkXFN4K9w"/>
      <ownedAttribute xmi:type="uml:Property" xmi:id="
          _WmudcGlZEeunebkXFN4K9w" name="ServerInput" type="
          _WmudZ2lZEeunebkXFN4K9w" aggregation="composite">
        <ownedComment xmi:type="uml:Comment" xmi:id="
            _WmudcWlZEeunebkXFN4K9w">
          <body>&lt;Enter note text here&gt;</body>
        </ownedComment>
      </ownedAttribute>
      <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"/>
      <ownedAttribute xmi:type="uml:Property" xmi:id="
          _WmuddGlZEeunebkXFN4K9w" name="concatenate" type="
          _WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
      <ownedAttribute xmi:type="uml:Property" xmi:id="
          _WmuddWlZEeunebkXFN4K9w" name="encryption" type="
```

```
        _WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmuddmlZEeunebkXFN4K9w" name="concatenate" type="
        _WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _Wmudd2lZEeunebkXFN4K9w" name="encryption" type="
        _WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudeGlZEeunebkXFN4K9w" name="ServerOutputPort" type="
        _WmudaGlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudeWlZEeunebkXFN4K9w" name="NetworkIPort" type="
        _WmudZ2lZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudemlZEeunebkXFN4K9w" name="decryption" type="
        _WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _Wmude2lZEeunebkXFN4K9w" name="NetworkOPort" type="
        _WmudaGlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudfGlZEeunebkXFN4K9w" name="NetworkIPort" type="
        _WmudZ2lZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudfWlZEeunebkXFN4K9w" name="decryption" type="
        _WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudfmlZEeunebkXFN4K9w" name="generateNonce" type="
        _WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _Wmudf2lZEeunebkXFN4K9w" name="encryptionWithSymKey" type
        ="_WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudgGlZEeunebkXFN4K9w" name="NetworkOPort" type="
        _WmudaGlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudgWlZEeunebkXFN4K9w" name="NetworkIPort" type="
        _WmudZ2lZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudgmlZEeunebkXFN4K9w" name="decryptionWithSymKey" type
        ="_WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _Wmudg2lZEeunebkXFN4K9w" name="decrementNonce" type="
        _WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudhGlZEeunebkXFN4K9w" name="encryptionWithSymKey" type
        ="_WmudZWlZEeunebkXFN4K9w" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _WmudhWlZEeunebkXFN4K9w" name="NetworkOPort" type="
        _WmudaGlZEeunebkXFN4K9w" aggregation="composite"/>
</packagedElement>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudhmlZEeunebkXFN4K9w" name="&lt;i.name,r.name&gt;"
    informationSource="_WmudcGlZEeunebkXFN4K9w"
    informationTarget="_Wmudc2lZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _Wmudh2lZEeunebkXFN4K9w" name="&lt;i.name&gt;"
    informationSource="_WmudcGlZEeunebkXFN4K9w"
    informationTarget="_WmuddGlZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudiGlZEeunebkXFN4K9w" name="SymKey(i,r)"
    informationSource="_WmudcmlZEeunebkXFN4K9w"
    informationTarget="_WmuddGlZEeunebkXFN4K9w"/>
```

```
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudiWlZEeunebkXFN4K9w" name="SharedKey(s,r)"
    informationSource="_Wmudc2lZEeunebkXFN4K9w"
    informationTarget="_WmuddWlZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudimlZEeunebkXFN4K9w" name="&lt;i.name; SymKey(i,r)&gt;"
     informationSource="_WmuddGlZEeunebkXFN4K9w"
    informationTarget="_WmuddWlZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _Wmudi2lZEeunebkXFN4K9w" name="SymKey(i,r)"
    informationSource="_WmudcmlZEeunebkXFN4K9w"
    informationTarget="_WmuddmlZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudjGlZEeunebkXFN4K9w" name="{i.name; SymKey(i,r)}
    _SharedKey(r,s)" informationSource="_WmuddWlZEeunebkXFN4K9w
    " informationTarget="_WmuddmlZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudjWlZEeunebkXFN4K9w" name="&lt;r.name, i.nonce&gt;"
    informationSource="_WmudcGlZEeunebkXFN4K9w"
    informationTarget="_WmuddmlZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudjmlZEeunebkXFN4K9w" name="&lt;i.nonce, r,name, SymKey(
    i,r), {i.name,SymKey(i,r)}_SharedKey(rs)}&gt;"
    informationSource="_WmuddmlZEeunebkXFN4K9w"
    informationTarget="_Wmudd2lZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _Wmudj2lZEeunebkXFN4K9w" name="SharedKey(i,s)"
    informationSource="_Wmudc2lZEeunebkXFN4K9w"
    informationTarget="_Wmudd2lZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudkGlZEeunebkXFN4K9w" name="{i.nonce, r,name, SymKey(i,r
    ), {i.name,SymKey(i,r)}_SharedKey(rs)}}_ShareKey(i,s)"
    informationSource="_Wmudd2lZEeunebkXFN4K9w"
    informationTarget="_WmudeGlZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudkWlZEeunebkXFN4K9w" name="&lt;{i.nonce, r,name, SymKey
    (i,r), {i.name,SymKey(i,r)}_SharedKey(rs)}_SharedKey(i,s)&
    gt;" informationSource="_WmudeWlZEeunebkXFN4K9w"
    informationTarget="_WmudemlZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudkmlZEeunebkXFN4K9w" name="&lt;{i.name,SymKey(i,r)}
    _SharedKey(rs)&gt;" informationSource="
    _WmudemlZEeunebkXFN4K9w" informationTarget="
    _Wmude2lZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _Wmudk2lZEeunebkXFN4K9w" name="&lt;{i.name,SymKey(i,r)}
    _SharedKey(rs)&gt;" informationSource="
    _WmudfGlZEeunebkXFN4K9w" informationTarget="
    _WmudfWlZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudlGlZEeunebkXFN4K9w" name="&lt;r.nonce&gt;"
    informationSource="_WmudfmlZEeunebkXFN4K9w"
    informationTarget="_Wmudf2lZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudlWlZEeunebkXFN4K9w" name="&lt;ShareKey(i,s)&gt;"
    informationSource="_WmudfWlZEeunebkXFN4K9w"
    informationTarget="_Wmudf2lZEeunebkXFN4K9w"/>
<packagedElement xmi:type="uml:InformationFlow" xmi:id="
    _WmudlmlZEeunebkXFN4K9w" name="&lt;{r.nonce}_SymKey(i,r)&gt
    ;" informationSource="_Wmudf2lZEeunebkXFN4K9w"
    informationTarget="_WmudgGlZEeunebkXFN4K9w"/>
```

```
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _Wmudl2lZEeunebkXFN4K9w" name="&lt;{r.nonce}_SymKey(i,r)&gt
        ;" informationSource="_WmudgWlZEeunebkXFN4K9w"
        informationTarget="_WmudgmlZEeunebkXFN4K9w"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _WmudmGlZEeunebkXFN4K9w" name="&lt;r.nonce&gt;"
        informationSource="_WmudgmlZEeunebkXFN4K9w"
        informationTarget="_Wmudg2lZEeunebkXFN4K9w"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _WmudmWlZEeunebkXFN4K9w" name="&lt;r.nonce-1&gt;"
        informationSource="_Wmudg2lZEeunebkXFN4K9w"
        informationTarget="_WmudhGlZEeunebkXFN4K9w"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _WmudmmlZEeunebkXFN4K9w" name="&lt;{r.nonce-1}_SymKey(i,r)
        }" informationSource="_WmudhGlZEeunebkXFN4K9w"
        informationTarget="_WmudhWlZEeunebkXFN4K9w"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _Wmudm2lZEeunebkXFN4K9w" name="UserInput" classifier="
        _WmudZ2lZEeunebkXFN4K9w"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _WmudnGlZEeunebkXFN4K9w" name="GenerateNonce" classifier="
        _WmudZWlZEeunebkXFN4K9w"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _WmudnWlZEeunebkXFN4K9w" name="Concatenate" classifier="
        _WmudZWlZEeunebkXFN4K9w"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _WmudnmlZEeunebkXFN4K9w" name="NetworkOutput" classifier="
        _WmudaGlZEeunebkXFN4K9w"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _Wmudn2lZEeunebkXFN4K9w" name="i.name, r.name"
        informationSource="_Wmudm2lZEeunebkXFN4K9w"
        informationTarget="_WmudnWlZEeunebkXFN4K9w"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _WmudoGlZEeunebkXFN4K9w" name="I.nonce" informationSource="
        _WmudnGlZEeunebkXFN4K9w" informationTarget="
        _WmudnWlZEeunebkXFN4K9w"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _WmudoWlZEeunebkXFN4K9w" name="i.name, r.name, i.nonce"
        informationSource="_WmudnWlZEeunebkXFN4K9w"
        informationTarget="_WmudnmlZEeunebkXFN4K9w"/>
  </packagedElement>
</uml:Model>
```

Listing 8: File delle strutture estratto dal tool

```
Blocks:
{'_Wmudb2lZEeunebkXFN4K9w': {'name': 'UserIPort', 'type': '
    InputPort'},
'_WmudbGlZEeunebkXFN4K9w': {'name': 'NetworkOPort', 'type': '
    OutputPort'},
'_WmudbWlZEeunebkXFN4K9w': {'name': 'NetworkIPort', 'type': '
    InputPort'},
'_WmudbmlZEeunebkXFN4K9w': {'name': 'UserOPort', 'type': '
    OutputPort'},
'_Wmudc2lZEeunebkXFN4K9w': {'name': 'GetSharedKey', 'type': '
    FunctionalBlock'},
'_WmudcGlZEeunebkXFN4K9w': {'name': 'ServerInput', 'type': '
    InputPort'},
'_WmudcmlZEeunebkXFN4K9w': {'name': 'GenerateSymKey(i,r)',
                            'type': 'FunctionalBlock'},
'_Wmudd2lZEeunebkXFN4K9w': {'name': 'encryption', 'type': '
    FunctionalBlock'},
```

```
'_WmuddGlZEeunebkXFN4K9w': {'name': 'concatenate', 'type': '
    FunctionalBlock'},
'_WmuddWlZEeunebkXFN4K9w': {'name': 'encryption', 'type': '
    FunctionalBlock'},
'_WmuddmlZEeunebkXFN4K9w': {'name': 'concatenate', 'type': '
    FunctionalBlock'},
'_Wmude2lZEeunebkXFN4K9w': {'name': 'NetworkOPort', 'type': '
    OutputPort'},
'_WmudeGlZEeunebkXFN4K9w': {'name': 'ServerOutputPort', 'type': '
    OutputPort'},
'_WmudeWlZEeunebkXFN4K9w': {'name': 'NetworkIPort', 'type': '
    InputPort'},
'_WmudemlZEeunebkXFN4K9w': {'name': 'decryption', 'type': '
    FunctionalBlock'},
'_Wmudf2lZEeunebkXFN4K9w': {'name': 'encryptionWithSymKey',
                            'type': 'FunctionalBlock'},
'_WmudfGlZEeunebkXFN4K9w': {'name': 'NetworkIPort', 'type': '
    InputPort'},
'_WmudfWlZEeunebkXFN4K9w': {'name': 'decryption', 'type': '
    FunctionalBlock'},
'_WmudfmlZEeunebkXFN4K9w': {'name': 'generateNonce', 'type': '
    FunctionalBlock'},
'_Wmudg2lZEeunebkXFN4K9w': {'name': 'decrementNonce',
                            'type': 'FunctionalBlock'},
'_WmudgGlZEeunebkXFN4K9w': {'name': 'NetworkOPort', 'type': '
    OutputPort'},
'_WmudgWlZEeunebkXFN4K9w': {'name': 'NetworkIPort', 'type': '
    InputPort'},
'_WmudgmlZEeunebkXFN4K9w': {'name': 'decryptionWithSymKey',
                            'type': 'FunctionalBlock'},
'_WmudhGlZEeunebkXFN4K9w': {'name': 'encryptionWithSymKey',
                            'type': 'FunctionalBlock'},
'_WmudhWlZEeunebkXFN4K9w': {'name': 'NetworkOPort', 'type': '
    OutputPort'},
'_Wmudm2lZEeunebkXFN4K9w': {'name': 'UserInput', 'type': 'InputPort
    '},
'_WmudnGlZEeunebkXFN4K9w': {'name': 'GenerateNonce', 'type': '
    FunctionalBlock'},
'_WmudnWlZEeunebkXFN4K9w': {'name': 'Concatenate', 'type': '
    FunctionalBlock'},
'_WmudnmlZEeunebkXFN4K9w': {'name': 'NetworkOutput', 'type': '
    OutputPort'}}

Flows:
{1: {'information': '<i.name,r.name>',
   'source': '_WmudcGlZEeunebkXFN4K9w',
   'target': '_Wmudc2lZEeunebkXFN4K9w'},
2: {'information': '<i.name>',
   'source': '_WmudcGlZEeunebkXFN4K9w',
   'target': '_WmuddGlZEeunebkXFN4K9w'},
3: {'information': 'SymKey(i,r)',
   'source': '_WmudcmlZEeunebkXFN4K9w',
   'target': '_WmuddGlZEeunebkXFN4K9w'},
4: {'information': 'SharedKey(s,r)',
   'source': '_Wmudc2lZEeunebkXFN4K9w',
   'target': '_WmuddWlZEeunebkXFN4K9w'},
5: {'information': '<i.name; SymKey(i,r)>',
   'source': '_WmuddGlZEeunebkXFN4K9w',
   'target': '_WmuddWlZEeunebkXFN4K9w'},
6: {'information': 'SymKey(i,r)',
   'source': '_WmudcmlZEeunebkXFN4K9w',
   'target': '_WmuddmlZEeunebkXFN4K9w'},
```

```
7: {'information': '{i.name; SymKey(i,r)}_SharedKey(r,s)',
    'source': '_WmuddWlZEeunebkXFN4K9w',
    'target': '_WmuddmlZEeunebkXFN4K9w'},
8: {'information': '<r.name, i.nonce>',
    'source': '_WmudcGlZEeunebkXFN4K9w',
    'target': '_WmuddmlZEeunebkXFN4K9w'},
9: {'information': '<i.nonce, r,name, SymKey(i,r), '
                   '{i.name,SymKey(i,r)}_SharedKey(rs)}>',
    'source': '_WmuddmlZEeunebkXFN4K9w',
    'target': '_Wmudd2lZEeunebkXFN4K9w'},
10: {'information': 'SharedKey(i,s)',
     'source': '_Wmudc2lZEeunebkXFN4K9w',
     'target': '_Wmudd2lZEeunebkXFN4K9w'},
11: {'information': '{i.nonce, r,name, SymKey(i,r), '
                    '{i.name,SymKey(i,r)}_SharedKey(rs)}}_ShareKey(i
                        ,s)',
     'source': '_Wmudd2lZEeunebkXFN4K9w',
     'target': '_WmudeGlZEeunebkXFN4K9w'},
12: {'information': '<{i.nonce, r,name, SymKey(i,r), '
                    '{i.name,SymKey(i,r)}_SharedKey(rs)}_SharedKey(i
                        ,s)>',
     'source': '_WmudeWlZEeunebkXFN4K9w',
     'target': '_WmudemlZEeunebkXFN4K9w'},
13: {'information': '<{i.name,SymKey(i,r)}_SharedKey(rs)>',
     'source': '_WmudemlZEeunebkXFN4K9w',
     'target': '_Wmude2lZEeunebkXFN4K9w'},
14: {'information': '<{i.name,SymKey(i,r)}_SharedKey(rs)>',
     'source': '_WmudfGlZEeunebkXFN4K9w',
     'target': '_WmudfWlZEeunebkXFN4K9w'},
15: {'information': '<r.nonce>',
     'source': '_WmudfmlZEeunebkXFN4K9w',
     'target': '_Wmudf2lZEeunebkXFN4K9w'},
16: {'information': '<ShareKey(i,s)>',
     'source': '_WmudfWlZEeunebkXFN4K9w',
     'target': '_Wmudf2lZEeunebkXFN4K9w'},
17: {'information': '<{r.nonce}_SymKey(i,r)>',
     'source': '_Wmudf2lZEeunebkXFN4K9w',
     'target': '_WmudgGlZEeunebkXFN4K9w'},
18: {'information': '<{r.nonce}_SymKey(i,r)>',
     'source': '_WmudgWlZEeunebkXFN4K9w',
     'target': '_WmudgmlZEeunebkXFN4K9w'},
19: {'information': '<r.nonce>',
     'source': '_WmudgmlZEeunebkXFN4K9w',
     'target': '_Wmudg2lZEeunebkXFN4K9w'},
20: {'information': '<r.nonce-1>',
     'source': '_Wmudg2lZEeunebkXFN4K9w',
     'target': '_WmudhGlZEeunebkXFN4K9w'},
21: {'information': '<{r.nonce-1}_SymKey(i,r)}',
     'source': '_WmudhGlZEeunebkXFN4K9w',
     'target': '_WmudhWlZEeunebkXFN4K9w'},
22: {'information': 'i.name, r.name',
     'source': '_Wmudm2lZEeunebkXFN4K9w',
     'target': '_WmudnWlZEeunebkXFN4K9w'},
23: {'information': 'I.nonce',
     'source': '_WmudnGlZEeunebkXFN4K9w',
     'target': '_WmudnWlZEeunebkXFN4K9w'},
24: {'information': 'i.name, r.name, i.nonce',
     'source': '_WmudnWlZEeunebkXFN4K9w',
     'target': '_WmudnmlZEeunebkXFN4K9w'}}
```

Listing 9: Output VerifPal

```
confidentiality? k_ir
  k_ir (k_ir) is obtained by Attacker.

confidentiality? n_r_minus_one
  n_r_minus_one (HASH(AEAD_DEC(SPLIT(AEAD_DEC(k_is, AEAD_ENC(k_is,
      CONCAT(n_i, k_ir, r_name, AEAD_ENC(k_rs, CONCAT(k_ir, i_name)
      , nil)), nil), nil)), AEAD_ENC(SPLIT(AEAD_DEC(k_rs, SPLIT(
      AEAD_DEC(k_is, AEAD_ENC(k_is, CONCAT(n_i, k_ir, r_name,
      AEAD_ENC(k_rs, CONCAT(k_ir, i_name), nil)), nil), nil)), nil)
      ), n_r, nil), nil))) is obtained by Attacker.

authentication? -> : e_n_r_minus_one
 When:
     i_name → nil ← mutated by Attacker (originally i_name)
     r_name → nil ← mutated by Attacker (originally r_name)
     n_i → nil ← mutated by Attacker (originally n_i)
     k_ir → nil ← mutated by Attacker (originally k_ir)
     e_recipient → AEAD_ENC(k_rs, CONCAT(k_ir, nil), nil)
     e_initiator → i_name ← mutated by Attacker (originally
         AEAD_ENC(k_is, CONCAT(n_i, k_ir, r_name, e_recipient), nil
         ))
     e_initiator_dec → AEAD_DEC(k_is, i_name, nil)
     n_i_response → SPLIT(AEAD_DEC(k_is, i_name, nil))
     k_ir_initiator → SPLIT(AEAD_DEC(k_is, i_name, nil))
     r_name_initiator → SPLIT(AEAD_DEC(k_is, i_name, nil))
     e_recipient_initiator → n_i ← mutated by Attacker (originally
          SPLIT(e_initiator_dec))
     unnamed_0 → ASSERT(n_i, SPLIT(AEAD_DEC(k_is, i_name, nil)))
     unnamed_1 → ASSERT(r_name, SPLIT(AEAD_DEC(k_is, i_name, nil))
         )
     e_recipient_dec → AEAD_DEC(k_rs, n_i, nil)
     k_ir_recipient → SPLIT(AEAD_DEC(k_rs, n_i, nil))
     i_name_recipient → SPLIT(AEAD_DEC(k_rs, n_i, nil))
     e_n_r → AEAD_ENC(SPLIT(AEAD_DEC(k_rs, n_i, nil)), n_r, nil)
     n_r_initiator → AEAD_DEC(SPLIT(AEAD_DEC(k_is, i_name, nil)),
         AEAD_ENC(SPLIT(AEAD_DEC(k_rs, n_i, nil)), n_r, nil), nil)
     n_r_minus_one → HASH(AEAD_DEC(SPLIT(AEAD_DEC(k_is, i_name,
         nil)), AEAD_ENC(SPLIT(AEAD_DEC(k_rs, n_i, nil)), n_r, nil)
         , nil))
     e_n_r_minus_one → AEAD_ENC(SPLIT(AEAD_DEC(k_rs, n_i, nil)),
         n_r, nil) ← mutated by Attacker (originally AEAD_ENC(
         k_ir_initiator, n_r_minus_one, nil))
     n_r_minus_one_recipient → n_r ← obtained by Attacker
     unnamed_2 → ASSERT(n_r, HASH(n_r))
     e_n_r_minus_one (AEAD_ENC(SPLIT(AEAD_DEC(k_rs, n_i, nil)), n_r
         , nil)), sent by Attacker and not by Initiator, is
         successfully used in AEAD_DEC(SPLIT(AEAD_DEC(k_rs, n_i,
         nil)), AEAD_ENC(SPLIT(AEAD_DEC(k_rs, n_i, nil)), n_r, nil)
         , nil) within Recipient's state.
```

Listing 10: Modellazione del protocollo in Applied Pi Calcolus

```
free c: channel.

type key.
type host.
type nonce.
type tag.

fun encrypt(bitstring,key): bitstring.
reduc forall x: bitstring, y: key; decrypt(encrypt(x,y),y) = x.

not attacker(new Kas).
not attacker(new Kbs).

free A, B: host.

table keys(host, key).

event endAparam(host,host).
event endBparam(host,host).
event beginAparam(host,host).
event beginBparam(host,host).
event endAkey(host, host, key).
event beginAkey(host, host, key).
event endBkey(host, host, key).
event beginBkey(host, host, key).

query x: host, y: host; inj-event(endAparam(x,y)) ==> inj-event(
    beginAparam(x,y)).
query x: host, y: host, z: key; inj-event(endAkey(x,y,z)) ==> inj-
    event(beginAkey(x,y,z)).
query x: host, y: host; inj-event(endBparam(x,y)) ==> inj-event(
    beginBparam(x,y)).
query x: host, y: host, z: key; inj-event(endBkey(x,y,z)) ==> inj-
    event(beginBkey(x,y,z)).


const c0: tag [data].
const c1: tag [data].


let processInitiator =

        new secretA: bitstring;
    in(c, (xA: host, h: host));
        if xA = A || xA = B then
        get keys(=xA, kas) in
        event beginBparam(xA,h);
        new Na: nonce;
        out(c,(xA, h, Na));
        in(c,m2: bitstring);
        let (=Na, =h, k: key, m: bitstring) = decrypt(m2, kas) in
        event beginBkey(xA, h, k);
        out(c,m);
        in(c, m3: bitstring);
        let (=c0, n: nonce) = decrypt(m3, k) in
        out(c, encrypt((c1, n), k));
        if h = B || h = A then
        event endAparam(xA, h);
        event endAkey(xA, h, k);
        out(c, encrypt(secretA, k)).
```

```
let processResponder =
        new secretB: bitstring;
        in(c, xB: host);
        if xB = A || xB = B then
        get keys(=xB, kbs) in

        in(c,m4: bitstring);
        let (k: key,h: host) = decrypt(m4,kbs) in
        event beginAparam(h, xB);
        event beginAkey(h, xB, k);
        new Nb: nonce;
        out(c, encrypt((c0, Nb), k));
        in(c,m5: bitstring);
        let (=c1, =Nb) = decrypt(m5,k) in
        if h = A || h = B then
        event endBparam(h, xB);
        event endBkey(h, xB, k);
        out(c, encrypt(secretB, k)).

let processS =
        in(c, (h1: host,h2: host,n: nonce));
        get keys(=h1, ks1) in
        get keys(=h2, ks2) in
        new k: key;
        out(c, encrypt((n, h2, k, encrypt((k,h1),ks2)), ks1)).

let processK =
        in(c, (h: host, k: key));
        if h <> A && h <> B then insert keys(h,k).

process
        new Kas: key; new Kbs: key;
        insert keys(A, Kas);
        insert keys(B, Kbs);
        ((!processInitiator) | (!processResponder) |(!processS) |
            (!processK))
```

Listing 11: Output ProVerif

```
Verification summary:

Query inj-event(endAparam(x,y)) ==> inj-event(beginAparam(x,y)) is
    true.

Query inj-event(endAkey(x,y,z)) ==> inj-event(beginAkey(x,y,z)) is
    true.

Query inj-event(endBparam(x,y)) ==> inj-event(beginBparam(x,y)) is
    true.

Query inj-event(endBkey(x,y,z)) ==> inj-event(beginBkey(x,y,z)) is
    true.
```

B Address Resolution Protocol

Listing 12: XMI

```
<?xml version="1.0" encoding="UTF-8"?><uml:Model xmlns:uml="http://
    www.omg.org/spec/UML/20110701" xmlns:xmi="http://schema.omg.org
    /spec/XMI/2.1" xmi:version="2.1" xmi:id="
    _b9ghgGVLEeurzoz2zKnPPg" name="Engineering">
  <eAnnotations xmi:id="_b9hIkGVLEeurzoz2zKnPPg" source="Objing">
    <contents xmi:type="uml:Property" xmi:id="
        _b9hIkWVLEeurzoz2zKnPPg" name="exporterVersion">
      <defaultValue xmi:type="uml:LiteralString" xmi:id="
          _b9hIkmVLEeurzoz2zKnPPg" value="3.0.0"/>
    </contents>
  </eAnnotations>
  <packagedElement xmi:type="uml:Class" xmi:id="
      _b9hIqGVLEeurzoz2zKnPPg" name="FunctionalBlock">
  </packagedElement>
  <packagedElement xmi:type="uml:Association" xmi:id="
      _b9hIrmVLEeurzoz2zKnPPg" name="funBlockAggregatedInFunArch"
      memberEnd="_b9hIqWVLEeurzoz2zKnPPg _b9hIsWVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:Class" xmi:id="
      _b9hIr2VLEeurzoz2zKnPPg" name="FunctionalArchitecture"
      isAbstract="true">
  </packagedElement>
  <packagedElement xmi:type="uml:Class" xmi:id="
      _b9hIwWVLEeurzoz2zKnPPg" name="InputPort">
  </packagedElement>
  <packagedElement xmi:type="uml:Class" xmi:id="
      _b9hIyGVLEeurzoz2zKnPPg" name="OutputPort">
  </packagedElement>
  <packagedElement xmi:type="uml:Package" xmi:id="
      _b9hwXGVLEeurzoz2zKnPPg" name="ARP">
    <packagedElement xmi:type="uml:Node" xmi:id="
        _b9hwXWVLEeurzoz2zKnPPg" name="Device">
      <ownedAttribute xmi:type="uml:Port" xmi:id="
          _b9hwXmVLEeurzoz2zKnPPg" name="EthernetIN" type="
          _b9hIwWVLEeurzoz2zKnPPg"/>
      <ownedAttribute xmi:type="uml:Port" xmi:id="
          _b9hwX2VLEeurzoz2zKnPPg" name="EthernetOUT" type="
          _b9hIyGVLEeurzoz2zKnPPg"/>
      <ownedAttribute xmi:type="uml:Property" xmi:id="
          _b9hwYGVLEeurzoz2zKnPPg" name="EthernetOUT" type="
          _b9hIyGVLEeurzoz2zKnPPg" aggregation="composite"/>
      <ownedAttribute xmi:type="uml:Property" xmi:id="
          _b9hwYWVLEeurzoz2zKnPPg" name="EthernetIN" type="
          _b9hIwWVLEeurzoz2zKnPPg" aggregation="composite"/>
      <ownedAttribute xmi:type="uml:Property" xmi:id="
          _b9hwYmVLEeurzoz2zKnPPg" name="getProtocolType" type="
          _b9hIqGVLEeurzoz2zKnPPg" aggregation="composite"/>
      <ownedAttribute xmi:type="uml:Property" xmi:id="
          _b9hwY2VLEeurzoz2zKnPPg" name="createAnnouncePackage"
          type="_b9hIqGVLEeurzoz2zKnPPg" aggregation="composite"/>
      <ownedAttribute xmi:type="uml:Property" xmi:id="
          _b9hwZGVLEeurzoz2zKnPPg" name="createProbePackage" type="
          _b9hIqGVLEeurzoz2zKnPPg" aggregation="composite"/>
      <ownedAttribute xmi:type="uml:Property" xmi:id="
          _b9hwZWVLEeurzoz2zKnPPg" name="getIPAddress" type="
          _b9hIqGVLEeurzoz2zKnPPg" aggregation="composite"/>
      <ownedAttribute xmi:type="uml:Property" xmi:id="
          _b9hwZmVLEeurzoz2zKnPPg" name="getIP&amp;MACfromTable"
          type="_b9hIqGVLEeurzoz2zKnPPg" aggregation="composite"/>
```

```
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _b9hwZ2VLEeurzoz2zKnPPg" name="createReplyPackage" type="
        _b9hIqGVLEeurzoz2zKnPPg" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _b9hwaGVLEeurzoz2zKnPPg" name="getMACAddress" type="
        _b9hIqGVLEeurzoz2zKnPPg" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _b9hwaWVLEeurzoz2zKnPPg" name="writeOnMemory" type="
        _b9hIqGVLEeurzoz2zKnPPg" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _b9hwamVLEeurzoz2zKnPPg" name="Memory" type="
        _b9hIr2VLEeurzoz2zKnPPg" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _b9hwa2VLEeurzoz2zKnPPg" name="PROBE_WAIT" type="
        _b9hIr2VLEeurzoz2zKnPPg" aggregation="composite"/>
    <ownedAttribute xmi:type="uml:Property" xmi:id="
        _b9hwbGVLEeurzoz2zKnPPg" name="ANNOUNCE_WAIT" type="
        _b9hIr2VLEeurzoz2zKnPPg" aggregation="composite"/>
  </packagedElement>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwbWVLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
      ar$pln, request, ar$sha, ip_gen, 00:00:00:00:00:00, ip_gen
      }" informationSource="_b9hwYWVLEeurzoz2zKnPPg"
      informationTarget="_b9hwaWVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwbmVLEeurzoz2zKnPPg" name="ar$sha, ip_gen"
      informationSource="_b9hwaWVLEeurzoz2zKnPPg"
      informationTarget="_b9hwamVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwb2VLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
      ar$pln, request, ar$sha, 0.0.0.0, 00:00:00:00:00:00, ip_gen
      }" informationSource="_b9hwYWVLEeurzoz2zKnPPg"
      informationTarget="_b9hwZmVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwcGVLEeurzoz2zKnPPg" name="ar$tha, ar$tpa"
      informationSource="_b9hwamVLEeurzoz2zKnPPg"
      informationTarget="_b9hwZmVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwcWVLEeurzoz2zKnPPg" name="ar$tha, ar$tpa"
      informationSource="_b9hwZmVLEeurzoz2zKnPPg"
      informationTarget="_b9hwZ2VLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwcmVLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
      ar$pln, reply, ar$sha, ar$spa, ar$tha, ar$tpa}"
      informationSource="_b9hwZ2VLEeurzoz2zKnPPg"
      informationTarget="_b9hwYGVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwc2VLEeurzoz2zKnPPg" name="ip_gen" informationSource="
      _b9hwZWVLEeurzoz2zKnPPg" informationTarget="
      _b9hwZGVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwdGVLEeurzoz2zKnPPg" name="ar$spa" informationSource="
      _b9hwYmVLEeurzoz2zKnPPg" informationTarget="
      _b9hwZGVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwdWVLEeurzoz2zKnPPg" name="ar$sha" informationSource="
      _b9hwaGVLEeurzoz2zKnPPg" informationTarget="
      _b9hwZGVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwdmVLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
      ar$pln, request, ar$sha, 0.0.0.0, 00:00:00:00:00:00, ip_gen
      }" informationSource="_b9hwZGVLEeurzoz2zKnPPg"
```

```
        informationTarget="_b9hwa2VLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwd2VLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
        ar$pln, request, ar$sha, 0.0.0.0, 00:00:00:00:00:00, ip_gen
        }" informationSource="_b9hwa2VLEeurzoz2zKnPPg"
        informationTarget="_b9hwYGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hweGVLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
        ar$pln, reply, ar$sha, ar$spa, ar$tha, ar$tpa}"
        informationSource="_b9hwYWVLEeurzoz2zKnPPg"
        informationTarget="_b9hwbGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hweWVLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
        ar$pln, request, ar$sha, ip_gen, 00:00:00:00:00:00, ip_gen
        }" informationSource="_b9hwY2VLEeurzoz2zKnPPg"
        informationTarget="_b9hwYGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwemVLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
        ar$pln, reply, ar$sha, ar$spa, RANDOM(ar$tha), ar$tpa}"
        informationSource="_b9hwZ2VLEeurzoz2zKnPPg"
        informationTarget="_b9hwYGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwe2VLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
        ar$pln, request, ar$sha, ip_gen, 00:00:00:00:00:00, ip_gen
        }" informationSource="_b9hwYWVLEeurzoz2zKnPPg"
        informationTarget="_b9hwaWVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwfGVLEeurzoz2zKnPPg" name="ar$sha, ip_gen"
        informationSource="_b9hwaWVLEeurzoz2zKnPPg"
        informationTarget="_b9hwamVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwfWVLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
        ar$pln, request, ar$sha, 0.0.0.0, 00:00:00:00:00:00, ip_gen
        }" informationSource="_b9hwYWVLEeurzoz2zKnPPg"
        informationTarget="_b9hwZmVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwfmVLEeurzoz2zKnPPg" name="ar$tha, ar$tpa"
        informationSource="_b9hwamVLEeurzoz2zKnPPg"
        informationTarget="_b9hwZmVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwf2VLEeurzoz2zKnPPg" name="ar$tha, ar$tpa"
        informationSource="_b9hwZmVLEeurzoz2zKnPPg"
        informationTarget="_b9hwZ2VLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwgGVLEeurzoz2zKnPPg" name="{as$hrd, ar$pro, ar$hln,
        ar$pln, reply, ar$sha, ar$spa, RANDOM(ar$tha), ar$tpa}"
        informationSource="_b9hwZ2VLEeurzoz2zKnPPg"
        informationTarget="_b9hwYGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwgWVLEeurzoz2zKnPPg" informationSource="
        _b9hwa2VLEeurzoz2zKnPPg" informationTarget="
        _b9hwbGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwgmVLEeurzoz2zKnPPg" informationSource="
        _b9hwbGVLEeurzoz2zKnPPg" informationTarget="
        _b9hwY2VLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwg2VLEeurzoz2zKnPPg" informationSource="
        _b9hwbGVLEeurzoz2zKnPPg" informationTarget="
        _b9hwZWVLEeurzoz2zKnPPg"/>
</packagedElement>
```

Listing 13: File delle strutture estratto dal tool

```
Blocks:
{'_b9hwX2VLEeurzoz2zKnPPg': {'name': 'EthernetOUT', 'type': '
    OutputPort'},
'_b9hwXmVLEeurzoz2zKnPPg': {'name': 'EthernetIN', 'type': '
    InputPort'},
'_b9hwY2VLEeurzoz2zKnPPg': {'name': 'createAnnouncePackage',
                            'type': 'FunctionalBlock'},
'_b9hwYGVLEeurzoz2zKnPPg': {'name': 'EthernetOUT', 'type': '
    OutputPort'},
'_b9hwYWVLEeurzoz2zKnPPg': {'name': 'EthernetIN', 'type': '
    InputPort'},
'_b9hwYmVLEeurzoz2zKnPPg': {'name': 'getProtocolType',
                            'type': 'FunctionalBlock'},
'_b9hwZ2VLEeurzoz2zKnPPg': {'name': 'createReplyPackage',
                            'type': 'FunctionalBlock'},
'_b9hwZGVLEeurzoz2zKnPPg': {'name': 'createProbePackage',
                            'type': 'FunctionalBlock'},
'_b9hwZWVLEeurzoz2zKnPPg': {'name': 'getIPAddress', 'type': '
    FunctionalBlock'},
'_b9hwZmVLEeurzoz2zKnPPg': {'name': 'getIP&MACfromTable',
                            'type': 'FunctionalBlock'},
'_b9hwa2VLEeurzoz2zKnPPg': {'name': 'PROBE_WAIT',
                            'type': 'FunctionalArchitecture'},
'_b9hwaGVLEeurzoz2zKnPPg': {'name': 'getMACAddress', 'type': '
    FunctionalBlock'},
'_b9hwaWVLEeurzoz2zKnPPg': {'name': 'writeOnMemory', 'type': '
    FunctionalBlock'},
'_b9hwamVLEeurzoz2zKnPPg': {'name': 'Memory', 'type': '
    FunctionalArchitecture'},
'_b9hwbGVLEeurzoz2zKnPPg': {'name': 'ANNOUNCE_WAIT',
                            'type': 'FunctionalArchitecture'}}

Flows:
{1: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, request,
    ar$sha, ip_gen, '
                    '00:00:00:00:00:00, ip_gen}',
   'source': '_b9hwYWVLEeurzoz2zKnPPg',
   'target': '_b9hwaWVLEeurzoz2zKnPPg'},
2: {'information': 'ar$sha, ip_gen',
   'source': '_b9hwaWVLEeurzoz2zKnPPg',
   'target': '_b9hwamVLEeurzoz2zKnPPg'},
3: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, request,
    ar$sha, 0.0.0.0, '
                    '00:00:00:00:00:00, ip_gen}',
   'source': '_b9hwYWVLEeurzoz2zKnPPg',
   'target': '_b9hwZmVLEeurzoz2zKnPPg'},
4: {'information': 'ar$tha, ar$tpa',
   'source': '_b9hwamVLEeurzoz2zKnPPg',
   'target': '_b9hwZmVLEeurzoz2zKnPPg'},
5: {'information': 'ar$tha, ar$tpa',
   'source': '_b9hwZmVLEeurzoz2zKnPPg',
   'target': '_b9hwZ2VLEeurzoz2zKnPPg'},
6: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, reply, ar$sha,
     ar$spa, '
                    'ar$tha, ar$tpa}',
   'source': '_b9hwZ2VLEeurzoz2zKnPPg',
   'target': '_b9hwYGVLEeurzoz2zKnPPg'},
7: {'information': 'ip_gen',
   'source': '_b9hwZWVLEeurzoz2zKnPPg',
   'target': '_b9hwZGVLEeurzoz2zKnPPg'},
8: {'information': 'ar$spa',
```

```
    'source': '_b9hwYmVLEeurzoz2zKnPPg',
    'target': '_b9hwZGVLEeurzoz2zKnPPg'},
9:  {'information': 'ar$sha',
    'source': '_b9hwaGVLEeurzoz2zKnPPg',
    'target': '_b9hwZGVLEeurzoz2zKnPPg'},
10: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, request,
    ar$sha, 0.0.0.0, '
                        '00:00:00:00:00:00, ip_gen}',
    'source': '_b9hwZGVLEeurzoz2zKnPPg',
    'target': '_b9hwa2VLEeurzoz2zKnPPg'},
11: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, request,
    ar$sha, 0.0.0.0, '
                        '00:00:00:00:00:00, ip_gen}',
    'source': '_b9hwa2VLEeurzoz2zKnPPg',
    'target': '_b9hwYGVLEeurzoz2zKnPPg'},
12: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, reply, ar$sha
    , ar$spa, '
                        'ar$tha, ar$tpa}',
    'source': '_b9hwYWVLEeurzoz2zKnPPg',
    'target': '_b9hwbGVLEeurzoz2zKnPPg'},
13: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, request,
    ar$sha, ip_gen, '
                        '00:00:00:00:00:00, ip_gen}',
    'source': '_b9hwY2VLEeurzoz2zKnPPg',
    'target': '_b9hwYGVLEeurzoz2zKnPPg'},
14: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, reply, ar$sha
    , ar$spa, '
                        'RANDOM(ar$tha), ar$tpa}',
    'source': '_b9hwZ2VLEeurzoz2zKnPPg',
    'target': '_b9hwYGVLEeurzoz2zKnPPg'},
15: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, request,
    ar$sha, ip_gen, '
                        '00:00:00:00:00:00, ip_gen}',
    'source': '_b9hwYWVLEeurzoz2zKnPPg',
    'target': '_b9hwaWVLEeurzoz2zKnPPg'},
16: {'information': 'ar$sha, ip_gen',
    'source': '_b9hwaWVLEeurzoz2zKnPPg',
    'target': '_b9hwamVLEeurzoz2zKnPPg'},
17: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, request,
    ar$sha, 0.0.0.0, '
                        '00:00:00:00:00:00, ip_gen}',
    'source': '_b9hwYWVLEeurzoz2zKnPPg',
    'target': '_b9hwZmVLEeurzoz2zKnPPg'},
18: {'information': 'ar$tha, ar$tpa',
    'source': '_b9hwamVLEeurzoz2zKnPPg',
    'target': '_b9hwZmVLEeurzoz2zKnPPg'},
19: {'information': 'ar$tha, ar$tpa',
    'source': '_b9hwZmVLEeurzoz2zKnPPg',
    'target': '_b9hwZ2VLEeurzoz2zKnPPg'},
20: {'information': '{as$hrd, ar$pro, ar$hln, ar$pln, reply, ar$sha
    , ar$spa, '
                        'RANDOM(ar$tha), ar$tpa}',
    'source': '_b9hwZ2VLEeurzoz2zKnPPg',
    'target': '_b9hwYGVLEeurzoz2zKnPPg'},
21: {'information': None,
    'source': '_b9hwa2VLEeurzoz2zKnPPg',
    'target': '_b9hwbGVLEeurzoz2zKnPPg'},
22: {'information': None,
    'source': '_b9hwbGVLEeurzoz2zKnPPg',
    'target': '_b9hwY2VLEeurzoz2zKnPPg'},
23: {'information': None,
    'source': '_b9hwbGVLEeurzoz2zKnPPg',
```

```
    'target': '_b9hwZWVLEeurzoz2zKnPPg'}}
```

Listing 14: Output VerifPal scenario probe-announce

```
confidentiality? probe
 probe (CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req), CONCAT(
     ar_sha, nil, nil, ip))) is obtained by Attacker.

confidentiality? announce
 announce (CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req), CONCAT(
     ar_sha, ip, nil, ip))) is obtained by Attacker.

authentication? -> : probe
 When:
    probe → CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req),
        CONCAT(ar_sha, ip, nil, ip)) ← mutated by Attacker (
        originally CONCAT(info_protocol_a, info_address_a))
    info_pro_probe → CONCAT(ar_hrd, ar_pro, ar_hln, type_req) ←
        obtained by Attacker
    info_add_probe → CONCAT(ar_sha, ip, nil, ip)
    ar_sha_probe → ar_sha
    ar_spa_probe → ip
    ar_tha_probe → nil
    ar_tpa_probe → ip
    announce → ar_hln ← mutated by Attacker (originally CONCAT(
        info_protocol_a, info_address_a2))
    unnamed_0 → SPLIT(ar_hln)
    info_announce → SPLIT(ar_hln)
    ar_sha_announce → SPLIT(SPLIT(ar_hln))
    ar_spa_announce → SPLIT(SPLIT(ar_hln))
    unnamed_1 → SPLIT(SPLIT(ar_hln))
    unnamed_2 → SPLIT(SPLIT(ar_hln))
    probe (CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req), CONCAT(
        ar_sha, ip, nil, ip))), sent by Attacker and not by A, is
        successfully used in SPLIT(CONCAT(CONCAT(ar_hrd, ar_pro,
        ar_hln, type_req), CONCAT(ar_sha, ip, nil, ip))) within B's
         state.

authentication? -> : announce
 When:
    probe → CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req),
        CONCAT(ar_sha, nil, nil, ip))
    info_pro_probe → CONCAT(ar_hrd, ar_pro, ar_hln, type_req) ←
        obtained by Attacker
    info_add_probe → CONCAT(ar_sha, nil, nil, ip)
    ar_sha_probe → ar_sha
    ar_spa_probe → nil
    ar_tha_probe → nil
    ar_tpa_probe → ip
    announce → CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req),
        CONCAT(ar_sha, nil, nil, ip)) ← mutated by Attacker (
        originally CONCAT(info_protocol_a, info_address_a2))
    unnamed_0 → CONCAT(ar_hrd, ar_pro, ar_hln, type_req) ←
        obtained by Attacker
    info_announce → CONCAT(ar_sha, nil, nil, ip)
    ar_sha_announce → ar_sha
    ar_spa_announce → nil
    unnamed_1 → nil
    unnamed_2 → ip
    announce (CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req),
        CONCAT(ar_sha, nil, nil, ip))), sent by Attacker and not by
         A, is successfully used in SPLIT(CONCAT(CONCAT(ar_hrd,
```

```
        ar_pro, ar_hln, type_req), CONCAT(ar_sha, nil, nil, ip)))
        within B's state.
```

Listing 15: Output VerifPal scenario probe-reply

```
confidentiality? probe
 probe (CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req), CONCAT(
     ar_sha, nil, nil, ip))) is obtained by Attacker.

confidentiality? reply
 reply (CONCAT(CONCAT(SPLIT(SPLIT(CONCAT(CONCAT(ar_hrd, ar_pro,
     ar_hln, type_req), CONCAT(ar_sha, nil, nil, ip)))), SPLIT(
     SPLIT(CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req), CONCAT(
     ar_sha, nil, nil, ip)))), SPLIT(SPLIT(CONCAT(CONCAT(ar_hrd,
     ar_pro, ar_hln, type_req), CONCAT(ar_sha, nil, nil, ip)))),
     type_reply), CONCAT(SPLIT(SPLIT(CONCAT(CONCAT(ar_hrd, ar_pro,
     ar_hln, type_req), CONCAT(ar_sha, nil, nil, ip)))), ip_used,
     sha_used, ip_used))) is obtained by Attacker.

authentication? -> : probe
 When:
    probe → ip_used ← mutated by Attacker (originally CONCAT(
        info_protocol_a, info_address_a))
    in_prot_a → SPLIT(ip_used)
    in_add_a → SPLIT(ip_used)
    ar_hrd_a → SPLIT(SPLIT(ip_used))
    ar_pro_a → SPLIT(SPLIT(ip_used))
    ar_hln_a → SPLIT(SPLIT(ip_used))
    type_req_a → SPLIT(SPLIT(ip_used))
    ar_sha_a → SPLIT(SPLIT(ip_used))
    unnamed_0 → SPLIT(SPLIT(ip_used))
    unnamed_1 → SPLIT(SPLIT(ip_used))
    ar_tpa_a → SPLIT(SPLIT(ip_used))
    info_protocol_b → CONCAT(SPLIT(SPLIT(ip_used)), SPLIT(SPLIT(
        ip_used)), SPLIT(SPLIT(ip_used)), type_reply)
    info_address_b → CONCAT(SPLIT(SPLIT(ip_used)), ip_used,
        sha_used, ip_used)
    reply → CONCAT(CONCAT(SPLIT(SPLIT(ip_used)), SPLIT(SPLIT(
        ip_used)), SPLIT(SPLIT(ip_used)), type_reply), CONCAT(SPLIT
        (SPLIT(ip_used)), ip_used, sha_used, ip_used))
    unnamed_2 → CONCAT(SPLIT(SPLIT(ip_used)), SPLIT(SPLIT(ip_used)
        ), SPLIT(SPLIT(ip_used)), type_reply)
    info_reply → CONCAT(SPLIT(SPLIT(ip_used)), ip_used, sha_used,
        ip_used)
    unnamed_3 → SPLIT(SPLIT(ip_used))
    unnamed_4 → ip_used
    ar_tha_reply → sha_used
    unnamed_5 → ip_used
    unnamed_6 → ASSERT(ar_sha, sha_used)
    probe (ip_used), sent by Attacker and not by A, is successfully
         used in CONCAT(SPLIT(SPLIT(ip_used)), SPLIT(SPLIT(ip_used)
        ), SPLIT(SPLIT(ip_used)), type_reply) within B's state.

authentication? -> : reply
 When:
    probe → CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req),
        CONCAT(ar_sha, nil, nil, ip))
    in_prot_a → CONCAT(ar_hrd, ar_pro, ar_hln, type_req) ←
        obtained by Attacker
    in_add_a → CONCAT(ar_sha, nil, nil, ip)
    ar_hrd_a → ar_hrd
    ar_pro_a → ar_pro
```

```
    ar_hln_a → ar_hln
    type_req_a → type_req
    ar_sha_a → ar_sha
    unnamed_0 → nil
    unnamed_1 → nil
    ar_tpa_a → ip
    info_protocol_b → CONCAT(ar_hrd, ar_pro, ar_hln, type_reply)
    info_address_b → CONCAT(ar_sha, ip_used, sha_used, ip_used)
    reply → CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req),
        CONCAT(ar_sha, nil, nil, ip)) ← mutated by Attacker (
        originally CONCAT(info_protocol_b, info_address_b))
    unnamed_2 → CONCAT(ar_hrd, ar_pro, ar_hln, type_req) ←
        obtained by Attacker
    info_reply → CONCAT(ar_sha, nil, nil, ip)
    unnamed_3 → ar_sha
    unnamed_4 → nil
    ar_tha_reply → nil
    unnamed_5 → ip
    unnamed_6 → ASSERT(ar_sha, nil)
    reply (CONCAT(CONCAT(ar_hrd, ar_pro, ar_hln, type_req), CONCAT(
        ar_sha, nil, nil, ip))), sent by Attacker and not by B, is
        successfully used in SPLIT(CONCAT(CONCAT(ar_hrd, ar_pro,
        ar_hln, type_req), CONCAT(ar_sha, nil, nil, ip))) within A'
        s state.
```

Listing 16: Modellazione del protocollo in Applied Pi Calcolus nello scenario di probe

```
type ipv4.
type MAC.

free c : channel.

fun concat(bitstring, bitstring, bitstring, bitstring, MAC, ipv4,
    MAC, ipv4):bitstring.

reduc forall x0: bitstring, x1: bitstring,x2: bitstring,x3:
    bitstring,x4:MAC, x5:ipv4, x6:MAC, x7:ipv4;
deconcat(concat(x0, x1,x2,x3,x4, x5, x6, x7))=(x0, x1,x2,x3,x4, x5,
     x6, x7).

free as_tpa:ipv4 [private].

query attacker(as_tpa).

let A =
  new as_hrd:bitstring;
  new as_pro:bitstring;
  new as_pln:bitstring;
  new request:bitstring;
  new as_sha:MAC;
  new as_spa:ipv4;
  new as_tha:MAC;
  out(c,concat(as_hrd,as_pro, as_pln, request, as_sha, as_spa,
      as_tha, as_tpa)).

let B = in (c, m:bitstring).

process
  !A|!B
```

Listing 17: Output ProVerif

```
Verification summary:

Query not attacker(as_tpa[]) is false.
```

C Crittosistema RSA

Listing 18: XMI

```
<?xml version="1.0" encoding="UTF-8"?><uml:Model xmlns:uml="http://
    www.omg.org/spec/UML/20110701" xmlns:xmi="http://schema.omg.org
    /spec/XMI/2.1" xmi:version="2.1" xmi:id="
    _b9ghgGVLEeurzoz2zKnPPg" name="Engineering">
  <eAnnotations xmi:id="_b9hIkGVLEeurzoz2zKnPPg" source="Objing">
    <contents xmi:type="uml:Property" xmi:id="
        _b9hIkWVLEeurzoz2zKnPPg" name="exporterVersion">
      <defaultValue xmi:type="uml:LiteralString" xmi:id="
          _b9hIkmVLEeurzoz2zKnPPg" value="3.0.0"/>
    </contents>
  </eAnnotations>
  <packagedElement xmi:type="uml:Package" xmi:id="
      _b9hwhGVLEeurzoz2zKnPPg" name="RSA">
    <packagedElement xmi:type="uml:Class" xmi:id="
        _b9hwhWVLEeurzoz2zKnPPg" name="Class"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _b9hwhmVLEeurzoz2zKnPPg" name="Scegli p" classifier="
        _b9hIqGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _b9hwh2VLEeurzoz2zKnPPg" name="scegli q" classifier="
        _b9hIqGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _b9hwiGVLEeurzoz2zKnPPg" name="test primalita" classifier="
        _b9hIr2VLEeurzoz2zKnPPg">
      <slot xmi:type="uml:Slot" xmi:id="_b9hwiWVLEeurzoz2zKnPPg">
        <eAnnotations xmi:id="_b9hwimVLEeurzoz2zKnPPg" source="
            Objing">
          <contents xmi:type="uml:Property" xmi:id="
              _b9hwi2VLEeurzoz2zKnPPg" name="isBindableInstance"/>
          <contents xmi:type="uml:Property" xmi:id="
              _b9hwjGVLEeurzoz2zKnPPg" name="Name">
            <defaultValue xmi:type="uml:LiteralString" xmi:id="
                _b9hwjWVLEeurzoz2zKnPPg" value="i"/>
          </contents>
        </eAnnotations>
      </slot>
    </packagedElement>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _b9hwjmVLEeurzoz2zKnPPg" name="calcola n=p*q" classifier="
        _b9hIqGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _b9hwj2VLEeurzoz2zKnPPg" name="calcola phi(n)=(p-1)(q-1)"
        classifier="_b9hIqGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _b9hwkGVLEeurzoz2zKnPPg" name="scegli e random &lt; phi(n)"
         classifier="_b9hIqGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _b9hwkWVLEeurzoz2zKnPPg" name="test coprimi" classifier="
        _b9hIqGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _b9hwkmVLEeurzoz2zKnPPg" name="calcola d= e*d=1 mod phi(n)"
         classifier="_b9hIqGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _b9hwk2VLEeurzoz2zKnPPg" name="Memory" classifier="
        _b9hIr2VLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
        _b9hwlGVLEeurzoz2zKnPPg" name="calcolo m^e" classifier="
        _b9hIqGVLEeurzoz2zKnPPg">
```

```
    <ownedComment xmi:type="uml:Comment" xmi:id="
        _b9hwlWVLEeurzoz2zKnPPg">
      <body>&lt;Enter note text here&gt;</body>
    </ownedComment>
  </packagedElement>
  <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
      _b9hwlmVLEeurzoz2zKnPPg" name="calcolo m^e mod n"/>
  <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
      _b9hwl2VLEeurzoz2zKnPPg" name="Memory" classifier="
      _b9hIr2VLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
      _b9hwmGVLEeurzoz2zKnPPg" name="input"/>
  <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
      _b9hwmWVLEeurzoz2zKnPPg" name="output"/>
  <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
      _b9hwmmVLEeurzoz2zKnPPg" name="input"/>
  <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
      _b9hwm2VLEeurzoz2zKnPPg" name="calcola c^d" classifier="
      _b9hIqGVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
      _b9hwnGVLEeurzoz2zKnPPg" name="calcola c^d mod n"
      classifier="_b9hIqGVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
      _b9hwnWVLEeurzoz2zKnPPg" name="output"/>
  <packagedElement xmi:type="uml:InstanceSpecification" xmi:id="
      _b9hwnmVLEeurzoz2zKnPPg" name="Memory" classifier="
      _b9hIr2VLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwn2VLEeurzoz2zKnPPg" name="p" informationSource="
      _b9hwhmVLEeurzoz2zKnPPg" informationTarget="
      _b9hwiGVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwoGVLEeurzoz2zKnPPg" name="q" informationSource="
      _b9hwh2VLEeurzoz2zKnPPg" informationTarget="
      _b9hwiGVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwoWVLEeurzoz2zKnPPg" name="p,q" informationSource="
      _b9hwiGVLEeurzoz2zKnPPg" informationTarget="
      _b9hwjmVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwomVLEeurzoz2zKnPPg" name="p,q" informationSource="
      _b9hwjmVLEeurzoz2zKnPPg" informationTarget="
      _b9hwj2VLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwo2VLEeurzoz2zKnPPg" name="phi(n)" informationSource="
      _b9hwj2VLEeurzoz2zKnPPg" informationTarget="
      _b9hwkWVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwpGVLEeurzoz2zKnPPg" name="phi(n)" informationSource="
      _b9hwj2VLEeurzoz2zKnPPg" informationTarget="
      _b9hwkGVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwpWVLEeurzoz2zKnPPg" name="e" informationSource="
      _b9hwkGVLEeurzoz2zKnPPg" informationTarget="
      _b9hwkWVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwpmVLEeurzoz2zKnPPg" name="e, phi(n)" informationSource
      ="_b9hwkWVLEeurzoz2zKnPPg" informationTarget="
      _b9hwkmVLEeurzoz2zKnPPg"/>
  <packagedElement xmi:type="uml:InformationFlow" xmi:id="
      _b9hwp2VLEeurzoz2zKnPPg" name="p,q" informationSource="
      _b9hwiGVLEeurzoz2zKnPPg" informationTarget="
```

```
        _b9hwk2VLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwqGVLEeurzoz2zKnPPg" name="n" informationSource="
        _b9hwjmVLEeurzoz2zKnPPg" informationTarget="
        _b9hwk2VLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwqWVLEeurzoz2zKnPPg" name="e,d" informationSource="
        _b9hwkmVLEeurzoz2zKnPPg" informationTarget="
        _b9hwk2VLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwqmVLEeurzoz2zKnPPg" name="e" informationSource="
        _b9hwl2VLEeurzoz2zKnPPg" informationTarget="
        _b9hwlGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwq2VLEeurzoz2zKnPPg" name="n" informationSource="
        _b9hwl2VLEeurzoz2zKnPPg" informationTarget="
        _b9hwlmVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwrGVLEeurzoz2zKnPPg" name="m" informationSource="
        _b9hwmGVLEeurzoz2zKnPPg" informationTarget="
        _b9hwlGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwrWVLEeurzoz2zKnPPg" name="m^e" informationSource="
        _b9hwlGVLEeurzoz2zKnPPg" informationTarget="
        _b9hwlmVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwrmVLEeurzoz2zKnPPg" name="c" informationSource="
        _b9hwlmVLEeurzoz2zKnPPg" informationTarget="
        _b9hwmWVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwr2VLEeurzoz2zKnPPg" name="c" informationSource="
        _b9hwmmVLEeurzoz2zKnPPg" informationTarget="
        _b9hwm2VLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwsGVLEeurzoz2zKnPPg" name="c^d" informationSource="
        _b9hwm2VLEeurzoz2zKnPPg" informationTarget="
        _b9hwnGVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwsWVLEeurzoz2zKnPPg" name="m" informationSource="
        _b9hwnGVLEeurzoz2zKnPPg" informationTarget="
        _b9hwnWVLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hwsmVLEeurzoz2zKnPPg" name="d" informationSource="
        _b9hwnmVLEeurzoz2zKnPPg" informationTarget="
        _b9hwm2VLEeurzoz2zKnPPg"/>
    <packagedElement xmi:type="uml:InformationFlow" xmi:id="
        _b9hws2VLEeurzoz2zKnPPg" name="n" informationSource="
        _b9hwnmVLEeurzoz2zKnPPg" informationTarget="
        _b9hwnGVLEeurzoz2zKnPPg"/>
  </packagedElement>
</uml:Model>
```

Listing 19: File delle strutture estratto dal tool

```
Blocks:
{'_b9hwh2VLEeurzoz2zKnPPg': {'name': 'scegli q', 'type': '
    FunctionalBlock'},
'_b9hwhmVLEeurzoz2zKnPPg': {'name': 'Scegli p', 'type': '
    FunctionalBlock'},
'_b9hwiGVLEeurzoz2zKnPPg': {'name': 'test primalita',
                            'type': 'FunctionalArchitecture'},
'_b9hwj2VLEeurzoz2zKnPPg': {'name': 'calcola phi(n)=(p-1)(q-1)',
                            'type': 'FunctionalBlock'},
'_b9hwjmVLEeurzoz2zKnPPg': {'name': 'calcola n=p*q', 'type': '
    FunctionalBlock'},
'_b9hwk2VLEeurzoz2zKnPPg': {'name': 'Memory', 'type': '
    FunctionalArchitecture'},
'_b9hwkGVLEeurzoz2zKnPPg': {'name': 'scegli e random < phi(n)',
                            'type': 'FunctionalBlock'},
'_b9hwkWVLEeurzoz2zKnPPg': {'name': 'test coprimi', 'type': '
    FunctionalBlock'},
'_b9hwkmVLEeurzoz2zKnPPg': {'name': 'calcola d= e*d=1 mod phi(n)',
                            'type': 'FunctionalBlock'},
'_b9hwl2VLEeurzoz2zKnPPg': {'name': 'Memory', 'type': '
    FunctionalArchitecture'},
'_b9hwlGVLEeurzoz2zKnPPg': {'name': 'calcolo m^e', 'type': '
    FunctionalBlock'},
'_b9hwlmVLEeurzoz2zKnPPg': {'name': 'calcolo m^e mod n', 'type': '
    other'},
'_b9hwm2VLEeurzoz2zKnPPg': {'name': 'calcola c^d', 'type': '
    FunctionalBlock'},
'_b9hwmGVLEeurzoz2zKnPPg': {'name': 'input', 'type': 'other'},
'_b9hwmWVLEeurzoz2zKnPPg': {'name': 'output', 'type': 'other'},
'_b9hwmmVLEeurzoz2zKnPPg': {'name': 'input', 'type': 'other'},
'_b9hwnGVLEeurzoz2zKnPPg': {'name': 'calcola c^d mod n',
                            'type': 'FunctionalBlock'},
'_b9hwnWVLEeurzoz2zKnPPg': {'name': 'output', 'type': 'other'},
'_b9hwnmVLEeurzoz2zKnPPg': {'name': 'Memory', 'type': '
    FunctionalArchitecture'}}

Flows:
{1: {'information': 'p',
   'source': '_b9hwhmVLEeurzoz2zKnPPg',
   'target': '_b9hwiGVLEeurzoz2zKnPPg'},
2: {'information': 'q',
   'source': '_b9hwh2VLEeurzoz2zKnPPg',
   'target': '_b9hwiGVLEeurzoz2zKnPPg'},
3: {'information': 'p,q',
   'source': '_b9hwiGVLEeurzoz2zKnPPg',
   'target': '_b9hwjmVLEeurzoz2zKnPPg'},
4: {'information': 'p,q',
   'source': '_b9hwjmVLEeurzoz2zKnPPg',
   'target': '_b9hwj2VLEeurzoz2zKnPPg'},
5: {'information': 'phi(n)',
   'source': '_b9hwj2VLEeurzoz2zKnPPg',
   'target': '_b9hwkWVLEeurzoz2zKnPPg'},
6: {'information': 'phi(n)',
   'source': '_b9hwj2VLEeurzoz2zKnPPg',
   'target': '_b9hwkGVLEeurzoz2zKnPPg'},
7: {'information': 'e',
   'source': '_b9hwkGVLEeurzoz2zKnPPg',
   'target': '_b9hwkWVLEeurzoz2zKnPPg'},
8: {'information': 'e, phi(n)',
   'source': '_b9hwkWVLEeurzoz2zKnPPg',
   'target': '_b9hwkmVLEeurzoz2zKnPPg'},
```

```
9: {'information': 'p,q',
   'source': '_b9hwiGVLEeurzoz2zKnPPg',
   'target': '_b9hwk2VLEeurzoz2zKnPPg'},
10: {'information': 'n',
    'source': '_b9hwjmVLEeurzoz2zKnPPg',
    'target': '_b9hwk2VLEeurzoz2zKnPPg'},
11: {'information': 'e,d',
    'source': '_b9hwkmVLEeurzoz2zKnPPg',
    'target': '_b9hwk2VLEeurzoz2zKnPPg'},
12: {'information': 'e',
    'source': '_b9hwl2VLEeurzoz2zKnPPg',
    'target': '_b9hwlGVLEeurzoz2zKnPPg'},
13: {'information': 'n',
    'source': '_b9hwl2VLEeurzoz2zKnPPg',
    'target': '_b9hwlmVLEeurzoz2zKnPPg'},
14: {'information': 'm',
    'source': '_b9hwmGVLEeurzoz2zKnPPg',
    'target': '_b9hwlGVLEeurzoz2zKnPPg'},
15: {'information': 'm^e',
    'source': '_b9hwlGVLEeurzoz2zKnPPg',
    'target': '_b9hwlmVLEeurzoz2zKnPPg'},
16: {'information': 'c',
    'source': '_b9hwlmVLEeurzoz2zKnPPg',
    'target': '_b9hwmWVLEeurzoz2zKnPPg'},
17: {'information': 'c',
    'source': '_b9hwmmVLEeurzoz2zKnPPg',
    'target': '_b9hwm2VLEeurzoz2zKnPPg'},
18: {'information': 'c^d',
    'source': '_b9hwm2VLEeurzoz2zKnPPg',
    'target': '_b9hwnGVLEeurzoz2zKnPPg'},
19: {'information': 'm',
    'source': '_b9hwnGVLEeurzoz2zKnPPg',
    'target': '_b9hwnWVLEeurzoz2zKnPPg'},
20: {'information': 'd',
    'source': '_b9hwnmVLEeurzoz2zKnPPg',
    'target': '_b9hwm2VLEeurzoz2zKnPPg'},
21: {'information': 'n',
    'source': '_b9hwnmVLEeurzoz2zKnPPg',
    'target': '_b9hwnGVLEeurzoz2zKnPPg'}}
```

Listing 20: Output VerifPal

```
authentication? -> : e2
When:
    gb → G^nil ← mutated by Attacker (originally G^b)
    e1 → PKE_ENC(G^nil, m_a)
    m_b → PKE_DEC(b, PKE_ENC(G^nil, m_a))
    e2 → PKE_ENC(G^b, m_a) ← mutated by Attacker (originally
        PKE_ENC(ga, m2_b))
    m2_a → PKE_DEC(a, PKE_ENC(G^b, m_a))
    e2 (PKE_ENC(G^b, m_a)), sent by Attacker and not by Bob, is
        successfully used in PKE_DEC(a, PKE_ENC(G^b, m_a)) within
        Alice's state.
```

Listing 21: Modellazione del protocollo in Applied Pi Calcolus

```
type skey.
type pkey.
type host.

fun pk(skey): pkey.
fun aenc(bitstring , pkey): bitstring .

reduc forall m: bitstring , k: skey ; adec(aenc(m, pk(k)) ,k) = m.

free c : channel.

free s : bitstring [private].

free A, B : host.

query attacker(s).

let clientA=
  in (c, pkB:pkey);
  out(c, aenc(s,pkB)).


let clientB=
  new skB:skey;
  out(c, pk(skB));
  in(c, m:bitstring);
  let z=adec(m,skB) in 0.

process
  clientA|clientB
```

Listing 22: Output ProVerif

```
Verification summary:

Query not attacker(s[]) is false.
```

Tutto il codice presentato nella Sezione 7 e nelle Appendici A-B-C è consultabile presso il sito: `edu.v-research.it`

Riferimenti bibliografici

[ABF16] Martin Abadi, Bruno Blanchet e Cedric Fournet. ≪The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication≫. In: *CoRR* abs/1609.03003 (2016). arXiv: 1609.03003. URL: http://arxiv.org/abs/1609.03003.

[AG97] Martıin Abadi e Andy Gordon. ≪A Calculus for Cryptographic Protocols: The Spi Calculus≫. In: CCS'97: Proceedings of the 4th ACM Conference on Computer and Communications Security. 1997, pp. 36–47.

[AR00] Martin Abadi e Phillip Rogaway. ≪Reconciling two views of cryptography (the computational soundness of formal encryption)≫. In: Proc. 1st IFIP International Conference on Theoretical Computer Science (IFIP–TCS'00). Vol. 1872 of LNCS. 2000.

[Bla12] Bruno Blanchet. ≪Security Protocol Verification: Symbolic and Computational Models≫. In: *Principles of Security and Trust*. A cura di Pierpaolo Degano e Joshua D. Guttman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 3–29. ISBN: 978-3-642-28641-4.

[Che08] S. Cheshire. *IPv4 Address Conflict Detection*. RFC 5227. http://www.rfc-editor.org/rfc/rfc5227.txt. RFC Editor, lug. 2008. URL: http://www.rfc-editor.org/rfc/rfc5227.txt.

[CKW11] Véronique Cortier, Steve Kremer e Bogdan Warinschi. ≪A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems≫. In: *Journal of Automated Reasoning* 46 (2011), pp. 225–259. URL: https://doi.org/10.1007/s10817-010-9187-9.

[DS81] Dorothy E. Denning e Giovanni Maria Sacco. ≪Timestamps in Key Distribution Protocols≫. In: *Commun. ACM* 24.8 (ago. 1981), pp. 533–536. ISSN: 0001-0782. DOI: 10.1145/358722.358740. URL: https://doi.org/10.1145/358722.358740.

[DY83] D. Dolev e A. Yao. ≪On the security of public key protocols≫. In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 198–208. DOI: 10.1109/TIT.1983.1056650.

[Low95] Gavin Lowe. ≪An Attack on the Needham-Schroeder Public-Key Authentication Protocol≫. In: *Inf. Process. Lett.* 56.3 (nov. 1995), pp. 131–133. ISSN: 0020-0190. DOI: 10.1016/0020-0190(95)00144-2. URL: https://doi.org/10.1016/0020-0190(95)00144-2.

[NS78] Roger Needham e Mike Schroeder. ≪Using encryption for authentication in large networks of computers≫. In: *Commun. ACM* 21 (1978), pp. 993–999.

[Plu82] David C. Plummer. *Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*. STD 37. http://www.rfc-editor.org/rfc/rfc826.txt. RFC Editor, nov. 1982. URL: http://www.rfc-editor.org/rfc/rfc826.txt.

[RSA78] Ronald L. Rivest, Adi Shamir e Leonard M. Adleman. «A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.» In: *Commun. ACM* 21.2 (1978), pp. 120–126. URL: http://dblp.uni-trier.de/db/journals/cacm/cacm21.html#RivestSA78.

[RVV17] Marco Rocchetto, Luca Viganó e Marco Volpe. «An interpolation-based method for the verification of security protocols». In: *Journal of Computer Security* 25.3 (2017), pp. 1–48. DOI: 10.3233/JCS-16832.

[SC14] Spyridon Samonas e David Coss. «The CIA Strikes Back: Redefining Confidentiality, Integrity and Availability in Security». In: *Journal of Information System Security* 10.3 (2014).