Università degli Studi di Verona

Dipartimento di Informatica

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Tesi di laurea magistrale

Potenzialità e sfide nell'analisi formale di protocolli per l'identità digitale con Tamarin

Relatore:
Prof. Mariano Ceccato

Correlatori:
Dr. Marco Pernpruner
Dr. Giada Sciarretta

Candidato:
Federico Graziola
VR481884

Anno Accademico 2023–2024

Abstract

Negli ultimi anni abbiamo assistito ad un notevole progresso tecnologico, grazie al quale è ora possibile eseguire svariate operazioni direttamente dai propri dispositivi. Tuttavia, questa evoluzione ha riguardato anche gli attacchi informatici, che possono mettere a rischio la sicurezza e la privacy dei cittadini in modo sempre più sofisticato. Emerge quindi sempre più il bisogno di mettere in sicurezza i protocolli online, specialmente quelli che assumono un ruolo particolarmente sensibile all'interno dell'infrastruttura. A tal fine, tra le tecniche più promettenti ci sono quelle basate su metodi formali, che necessitano di una specifica formale del protocollo al fine di verificare la sussistenza o meno di specifiche proprietà di sicurezza. L'obiettivo di questa tesi è quello di approfondire il tool di verifica formale Tamarin, con lo scopo di fornire indicazioni e considerazioni utili per la modellazione precisa di protocolli; considereremo, ad esempio, alcune strategie per modellare le proprietà dei canali di comunicazione e le capacità degli attaccanti da considerare. Per esaminare concretamente le funzionalità e potenzialità di Tamarin, applicheremo tale strumento a protocolli reali per la gestione dell'identità digitale, discutendone i risultati. Nel corso della tesi, espliciteremo inoltre alcune lezioni apprese ed alcune sfide ancora aperte.

Indice

Elenco delle tabelle

Elenco delle figure

Elenco dei codici

Capitolo 1

Introduzione

1.1 Contesto e Motivazioni

Negli ultimi anni stiamo assististendo ad un notevole progresso tecnologico e sociale, con l'introduzione di nuove tecnologie che ci permettono d'eseguire svariate operazioni direttamente dai nostri dispositivi quali cellulari, tablet e PC.

Basti pensare alle operazioni bancarie tramite sistemi di online banking, come bonifici o pagamenti, ma anche l'accesso ai servizi online di SPID per usufruire dei servizi online della pubblica amministrazione. Questi servizi, che una volta non erano dispobili dai nostri dispositivi, possono essere ora utilizzati con comodità e velocità anche dalle nostre abitazioni senza recarci fisicamente nelle sedi dedicate.

Sfortunatamente questo ha visto anche una notevole e continua evoluzione degli attacchi informatici, che mirano a mettere a rischio la sicurezza e la privacy dei cittadini cercando di rubarne le identità e codici d'accesso. Siamo obbligati, quindi, a studiare sistemi e protocolli di sicurezza sempre più sicuri per mantenere i nostri dati al sicuro.

Questa condotta è emersa anche nei report "The State of Secure Identity 202" [15] e "Rapporto CLUSIT 2024" [19], dove possiamo trovare statistiche ufficiali degli attacchi avvenuti durante lo scorso anno e del loro relativo aumento.

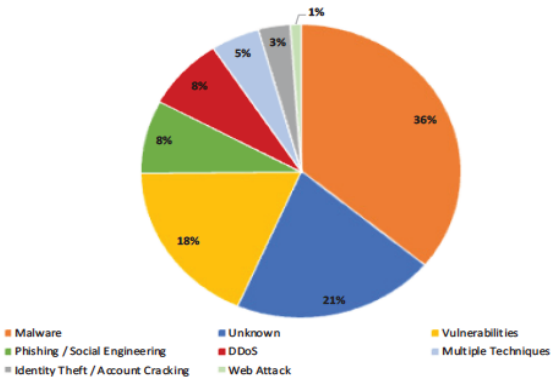

Figura 1.1: Distribuzione delle tecniche di attacco nell'anno 2023

Nella Figura 1.1, presa dal Rapporto CLUSIT 2024 (pagina 20), mostra la distribuzione delle tecniche di attacco seguite dal 2019 al 2023 nel suolo italiano.

Questa tesi vuole quindi concentrarsi principalmente sulla modellazione di nuovi protocolli di Identity Management per i servizi online, un esempio sono le carte d'identità elettroniche, tuttavia le soluzioni d'autenticazione con queste soffrono ancora di molte limitazioni d'usabilità, in quanto la progettazione di questi protocolli risulta essere molto complessa anche per colpa del numero di tecnologie da coinvolgere e i vincoli di sicurezza da rispettare per poter prevenire potenziali vulnerabilità,

Esistono molteplici tecniche e strumenti di supporto per le analisi formali per determinare la corretta implemetazione dei protocolli e per di renderli sicuri. I Model Checker sono uno di queste tecniche che ci siamo valsi per ottenere un livello di sicurezza più precisa; partendo prima da ASLan per poi investigare nuovi tool, uno di questi, allo stato dell'arte, è Tamarin che fornisce prove formali delle proprietà di sicurezza e modellazione dei protocolli anche con modalità interattive per una modellazione più precisa.

1.2 Contributi

Questa tesi nasce a seguito del tirocinio svolto nel Centro per la Cybersecurity della Fondazione Bruno Kessler, dove il topic di interesse è stato: "Valutazione degli strumenti di verifica per i protocolli di sicurezza".

L'obbiettivo e sfida è stata quella di scoprire il tool di Tamarin per la realizzazione di nuovi protocolli per l'identità digitale, nel dettaglio si è voluto capire quali potenzialità e sfide di definizione e formalizzazione di modellazione dei procolli esso offre a differenza del tool "ASLan++ e SATMC" utilizzato in Fondazione Bruno Kessler.

Non essendoci studi dettagliati su Tamarin nel centro di FBK sono state intraprese diverse strade per la definizione dei protocolli, con l'utilizzo di diversi costrutti per svolgere le analisi formali dei protocolli. Un obbiettivo principale è stato anche quello di cercare di modellare gli attaccanti e le strategie che questi ultimi sfruttano per interferire con la nella normale escuzione del protocollo allo scopo di violarlo.

1.3 Struttura della Tesi

La tesi è strutturata come segue:

- **Capitolo 2: Background**

  Introduzione allo studio del problema del Model Checking e presentazione degli strumenti Tamarin Prover e SATMC per l'analisi formale dei protocolli.

- **Capitolo 3: Stato dell'Arte**

  Presentazione del Thread di ricerca e dei protocolli realizzati nel centro Cybersecurity della Fondazione Bruno Kessler in ASLan++.

- **Capitolo 4: Analisi protocollo Needham-Schroeder**

  Studio del protocollo NSPK e della sua implementazione attraverso i tool per l'analisi formale con la presentazione delle sfide riscontrate in Tamarin.

- **Capitolo 5: Analisi protocollo basato su Password**

  Modellazione in Tamarin di un protocollo basato su Password con integrazione delle nuove tecniche finalizzate a migliorare e risolvere le sfide incontrate.

- **Capitolo 6: Conclusioni**

  Riassunto della ricerca condotta e presentazione dei lavori futuri con spunti per migliorare la formalizzazione dei protocolli.

Capitolo 2

Background

In questo capitolo, verranno forniti alcuni concetti preliminari che risultano fondamentali per la corretta comprensione della tesi.

2.1 Problema del Model Checking

La nostra attenzione ricade sulla definizione di che cosa sia un model checker ripresa e formalizzata in [5], più precisivamente sul model checking problem, ovvero un problema di verifica di una formula F che sia valida in un modello M:

$$M \models f$$

dove M rappresenta il progetto-protocollo, solitamente finito, mentre f è la proprietà desiderata di questa progettazione espressa in una logica temporale come *Computational Tree Logic* (CTL) o *Linear Temporal Logic* (LTL). Le tecniche di model checking sono basate sull'attraversamento esaustivo dello stato del modello, e sono spesso automatiche e molto efficienti. Tuttavia, il model checking è in gran parte limitato per modelli finiti e formule proposizionali. Un problema di dimostrazione di teoremi, invece, è un problema di verifica della validità di una formula, cioè, deve valere in ogni modello. La logica è spesso molto più espressiva della logica temporale proposizionale, e le proprietà su domini infiniti possono essere facilmente espressi. Un problema di dimostrazione di un teorema viene risolto trovando una dimostrazione della formula (che viene quindi chiamato teorema) in un sistema di dimostrazione. Per poter utilizzare la dimostrazione di teoremi, dobbiamo tradurre il nostro problema di verifica in un teorema dimostrativo. Poiché molti dimostratori di teoremi utilizzano la logica di ordine superiore (*Higher Order Logic*), il potere espressivo di questo approccio è praticamente illimitato. Nella pratica, però, la competenza richiesta e l'importo di guida manuale nella ricerca di una prova può diventare proibitivo.

Sempre nell'articolo [5] viene presentata l'architettura di un tipico dimostratore di teoremi (*theorem prover*); che sarà importante per comprendere gli strumenti che verranno usati in seguito, viene mostrata in Figura 2.1.

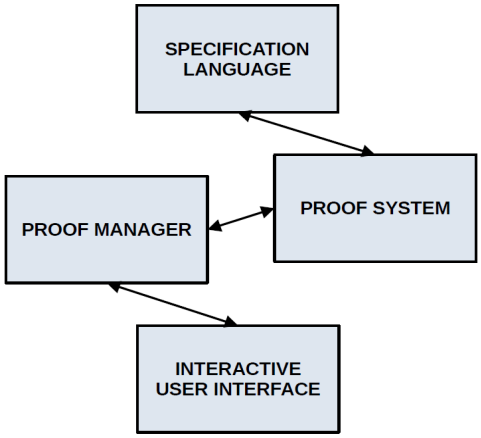

Figura 2.1: Architettura di un tipico dimostratore di teoremi

2.2 Strumenti per l'analisi formale

Come precedentemente spiegato, l'evoluzione dei servizi ha riguardato anche gli attacchi informatici, che sempre più risultano essere raffinati e con l'obbiettivo di mettere a rischio la sicurezza e la privacy dei cittadini in modo sempre più sofisticato.

Emerge quindi sempre più il bisogno di mettere in sicurezza i protocolli online, specialmente quelli che assumono un ruolo particolarmente sensibile all'interno delle infrastrutture pubbliche. Come però si può intuire non si possono direttamente testare sulle infrastutture i protocolli che si modellano, nasce quindi il bisogno di avvalersi di strumenti precisi per poter analizzare sia i protocolli che gli attacchi che possono comprometterne il funzionamento.

Questi strumenti sono tool per l'analisi formale dei protocolli che verificano la sussistenza di specifiche proprietà di sicurezza sui protocolli in analisi. Esistono molteplici proposte di soluzioni e strategie per la modellazione e la verifica dei protocolli, questa ricerca vuole concentrarsi sullo studio dei tool Tamarin Prover e SATMC. Nella Tabella 3.1 possimo notare le differenze principali di questi tool. Andremo quindi a studiarne le caratteristiche nel contesto della modellazione dei protocolli crittografici.

TOOL	INPUT	OUTPUT	TECNICA DI ANALISI	PROGETTO
SATMC	ASLAN	OF	Model checker basato su SAT con LTL	AVANTSSAR
Tamarin Prover	security protocol theory format (.spthy)	Spthy, Spthytyped, Msr	Dimostratore di teoremi: strumento di verifica dei protocolli di sicurezza, quest'ultimi vengono specificati come sistemi di riscrittura multiset e analizzati.	Tamarin Prover Project

Tabella 2.1: Principali differenze tra i tool SATMC e Tamarin Prover

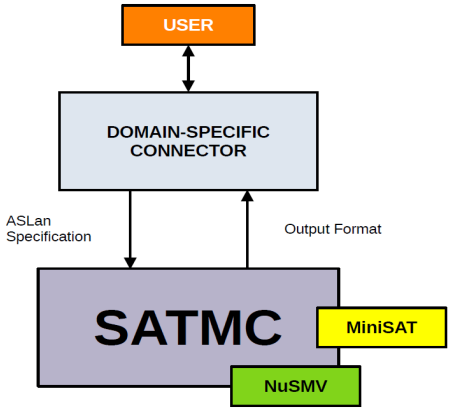

Figura 2.2: Architettura di SATMC

2.2.1 SATMC

Questo tool nasce nasce dal progetto AVANTSSAR [8], finanziato dall'Unione Europea nel 2008. L'obbiettivo di questo progetto era appunto dettato dall'esplosione dei servizi nelle infrastutture di rete, dove si vogliono risolvere importanti problemi di fiducia e di sicurezza nella realizzazione dei protocolli. Infatti quando si modellano i protocolli di sicurezza possono essere presenti vulnerabilità dovute all'interferenza tra i servizi e le policy di differenti componenti e della loro comunicazione. Pertanto, è necessaria la convalida dei componenti del servizio, in tal senso AVANTSSAR propone come soluzione a questa sfida una tecnologia rigorosa per la specifica formale e la validazione della fiducia e della sicurezza delle architetture di questi servizi. Tra le diverse componenti, AVANTSSAR, introduce il linguaggio ASLan++ per specificare formalmente i protocolli che devono essere analizzati. La specifica formale realizzata in ASLan++ viene poi tradotta automaticamente in ASLan, un linguaggio più a basso livello che viene dato poi in input a SATMC [1]. Ques'ultimo tool, in particolare, è un model checker basato su problemi di soddisfacibilità booleana (SAT) con LTL. SATMC, la cui architettura è illustrata in Figura 2.2, è integrato ed utilizzato come backend in diverse piattaforme basate su metodi formali come il tool AVISPA e la piattaforma AVANTSSAR [21].

Semantica di ASLan++

In ASLan++ possiamo trovare i seguenti principali elementi costitutivi quali le `entity`, simili alle classi in Java o ai ruoli in contesto dei protocolli di sicurezza e comunicazione. Le entity dichiarano vari simboli, macro, obiettivi e altri elementi. Inoltre possono avere parametri e contenere sottoentità nidificate.

(i) `Environment` – radice globale del sistema da specificare, in modo simile alla routine "main"

(ii) `Session` – contiene entità comunicanti, ovvero gli attori del protocollo

(iii) `Body` – viene specificato il comportamento dell'entità o gli obiettivi di sicurezza. Quando l'enità viene istanziata, viene eseguito il corpo

Bisogna avere almeno due sessioni in parallelo per evidenziare degli attacchi in corso, ciò si ottiene con `symbolic sessions`; immaginiamo d'avere due sessioni A e B, dove A potrebbe essere un'istanza onesta di Alice o un aggressore disonesto e B potrebbe essere un esempio onesto di Bob o un aggressore. Il model checker valuterà quindi le combinazioni per le due sessioni parallele, tenendo conto delle simmetrie.

La modellazione degli attaccanti considerati richiede preliminarmente la definizione delle loro capacità, che in ASLan++ devono essere espresse in funzione del loro impatto sul protocollo (ad esempio, proprietà sui canali che vengono modificate, valori che vengono appresi, eccetera). Tali capacità vengono poi mantenute o commentate a seconda che si consideri o meno lo specifico attaccante. Prendiamo come esempio le seguenti 3 tipologie di definizione delle proprietà:

- `weakly_authentic(Ch_U2B)` – Il canale `Ch_U2B` è debole: l'autenticità è debole se input del canale è accessibile da un singolo mittente che rimane sconosciuto.
- `authentic_on(Ch_U2B, User)` – Il canale `Ch_U2B` è autenticato da `User`.
- `confidential_to(Ch_U2B, Browser)` – Il canale `Ch_U2B` è confidenziale se al suo output può accedere solo il destinatario specificato, ovvero `Browser`.

Per simulare un attacco in ASLan++ occorrerà commentare o decommentare le specifiche proprietà dei canali e livelli di confidenzialità. Il Codice 2.1 mostra un esempio di definizione delle capacità degli attaccanti e l'attivazione delle proprietà per simularne il comportamento.

```
1  body { % of Session
2      % PCT
3      authentic_on(Ch_U2B, User);              % comment out
4      %weakly_authentic(Ch_U2B);               % uncomment
5      % use the same channel ch_U2B in the two sessions below
       comment out userOwnComputer in Environment body
6
7      % ES, SS
8      confidential_to(Ch_U2EICApp, EICApp);    % comment out
9      confidential_to(Ch_EICApp2U, User);      % comment out
10     confidential_to(Ch_U2B, Browser);        % comment out
11
12     % SE
13     %iknows(PIN);                            % uncomment
14     % uncomment iknows(OTP) in IdPServer comment out
       confidential_to(Ch_U2B, Browser) in ES, SS
15     authentic_on(Ch_B2U, Browser);           % comment out
16
17     % MM
18     % replace "eicapp" with "i" in one of the sessions below
19     authentic_on(Ch_EICApp2U, EICApp);
20     authentic_on(Ch_EICApp2EIC, EICApp);
21 ... }
```

Codice 2.1: Tracce d'attacco in ASLan++

2.2.2 Tamarin Prover

Tamarin Prover nasce grazie alla ricerca condotta da Simon Meier, Cas Cremers e David Basin, professori dell'ETH Zurich nel 2011. Essi propongono una ricerca per la formalizzazione dei problemi nella creazione e validazione dei protocolli di sicurezza: [12]. In questa ricerca, i ricercatori, propongono una semantica operativa per i protocolli di sicurezza nel dimostratore di teoremi interattivo *Isabelle/HOL* e ricaviamo due invarianti forti indipendenti dal protocollo. Questi invarianti permettono di ragionare sulla possibile origine dei messaggi e di giustificare un'ipotesi di tipizzazione locale per le variabili di protocollo altrimenti non tipizzate. Le due regole costituiscono il nucleo di una teoria che ben si adatta alla costruzione interattiva di prove di correttezza naturali e leggibili dall'uomo.

Inoltre, sempre in questo articolo, viene proposto un algoritmo che genera automaticamente script di prova basati su questi invarianti. Sia la costruzione di prove interattive che quella automatica sono più veloci rispetto agli approcci concorrenti. Inoltre, vi sono forti garanzie di correttezza poiché tutte le dimostrazioni, comprese quelle che derivano la teoria sottostante dalla semantica, vengono controllate automaticamente. Da questo lavoro, Staub Cedric, successivamente, propone un progetto di tesi che propone un interfaccia per la progettazione interattiva dei protocollo di sicurezza nel 2011 [20].

ETH di Zurigo pubblicherà, quindi, ufficialmente lo strumento Tamarin Prover [13] nel 2013, proponendosi come tool di analisi open source per la crittografia dei protocolli, fornendo supporto algoritmico per cercare un controesempio alla proprietà di sicurezza e costruendo una dimostrazione di correttezza o di eventuale attacco. Il tool è tuttora mantenuto da ETH Zurich, che elenca sul proprio sito web [23] tutte le pubblicazioni (articoli di ricerca, report e tesi) relative al tool, o che lo utilizzano per analizzare protocolli in contesti differenti.

ETH propone, annualmente, un manuale [24] per la comprensione iniziale di Tamarin, contenente le pricipali novità implementate. Questo manuale si appoggia, inoltre, ad un repository pubblico: [22], nella quale possiamo trovare diversi esempi.

Come si può intuire, il materiale a disposizione inizia ad essere notevole e richiederebbe troppo tempo per essere presentato nella sua totalità. Saranno quindi presentati solo i costrutti più importanti, che fondano il linguaggio di Tamarin. Esiste inoltre una comunity dedicata a Tamarin; in questa comunity avvengono scambio di idee e discussioni sui vari topic di Tamarin, oltre alla possibilità di interpellare gli stessi autori di Tamarin. La presenza di una community attiva rappresenta un aspetto estremamente importante e vantaggioso di Tamarin.

Normalmente, come nella maggior parte degli articoli, con il termine Tamarin ci si riferisce al linguaggio di programmazione con cui si scrive il file di input, mentre Tamarin Prover il tool che permette l'effettiva analisi della specifica.

Struttura di Tamarin e Tamarin Prover

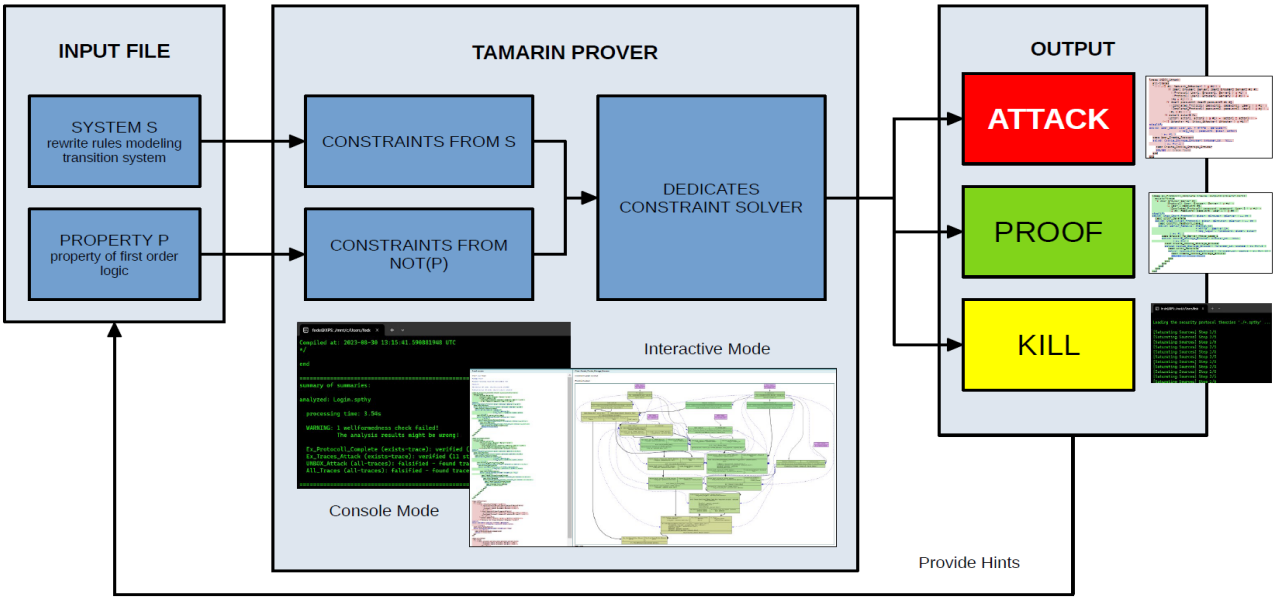

Figura 2.3: Tamarin Prover

```
// Struttura Programma

    // Nome della teoria
    theory NomeTeoria
    begin

    // Primitive crittografiche
    builtins: hashing, asymmetric-encryption

    // Regole
    rule Register_pk:
        ...

    rule Get_pk:
        ...

    ...

    // Proprietà
    lemma Client_session_key_secrecy:
        ...

    lemma Client_auth:
        ...

    ...

    lemma Client_session_key_honest_setup:
        exists-trace

    end
```

Figura 2.4: Formato Teorico del Protocollo di Sicurezza

La Figura 2.3, che prende spunto da quella presente nell'articolo [3], rappresenta la struttura completa del tool di Tamarin Prover. La Figura 2.4, invece, mostra un esempio di specifica in Tamarin, appartenente al formato spthy (*security protocol theory format*).

File di Input

Tamarin Prover prende un modello (si veda la Figura 2.4) che specifica il protocollo e le capacità dei possibili avversari, oltre alla proprietà di sicurezza da verificare nelle esecuzioni del protocollo. Il modello si suddivide in due parti:

1. **System S**

   I protocolli e gli avversari vengono specificati utilizzando un linguaggio espressivo basato su *multiset rewriting rules*. Queste descrivono un sistema di transizione etichettato il cui stato consiste in una rappresentazione simbolica della conoscenza dell'avversario, i messaggi scambiati attraverso i canali di comunicazionele, le informazioni sui valori generati durante l'esecuzione e lo stato delle entità che prendono parte al protocollo.

2. **Property P**

   Le proprietà di sicurezza sono modellate come *trace properties*, confrontato con le tracce del sistema di transizione. Le formule sono espresso in un frammento di logica del primo ordine (*Isabelle/HOL*) con quantificazione nel tempo.

La compresione totale del linguaggio Tamarin richiederebbe troppo tempo e questa tesi non vuole essere un riassunto del manuale, saranno quindi presentati solo i costrutti più importanti di Tamarin.

- `Regole`

  Possiamo considerare il costrutto *rule* come il vero e proprio cuore dei protocolli in Tamarin. Una ricerca che descrive il concetto alla base che forma una rule è: [6]. Sarà nostra premura considerare solo gli aspetti più interessanti di questo costrutto. Non esiste un IDE che visualizza gli errori in fase di scrittura delle rule, l'unico modo per capire se queste sono corrette è vederne la rappresentazione grafica sul sito generato da Tamarin Prover, dove però saranno visualizzati i soli valori che assumono le variabili e messaggi.

  Il Codice 2.2 mostra come viene definita una rule con 4 parti:

  1. `let .. in` = sezione della rule dove vengono definite le forme dei messaggi in ingresso e uscita
  2. `In() ..` = fatti che devono essere soffisfatti per poter verificare la rule, qui vengono inoltre specificati i messaggi e pacchetti necessari per completare la derivazione della rule
  3. `Create_Password() ..` = in questa sezione vengono definite le proprietà che verranno richiamate delle restrizioni, inoltre possiamo definire gli elementi necessari per i test
  4. `Out() ..` = fatti in uscita, nel dettaglio in questa sezione ci si occupa di rendere disponibili alle altre rule i messaggi per completare le loro derivazioni e verifiche

```
rule User_Rule:
  // Sezione dei messaggi
  let
    msg_in = <'msg_in', $User>
    msg_out = <'msg_out', ~password, $User>
  in
  // Sezione fatti in ingresso
  [
    In(msg_in),
    Fr(~password),
  ]--
  // Sezione proprieta'
  [
    Create_Password(),
    Password_Secure(~password, $User),
  ]->
  // Sezione fatti in uscita
  [
    Out(msg_out),
    !User_Send(msg_out)
  ]
```

Codice 2.2: Struttura Regole

- `Lemmi`

  I lemmi sono costrutti di Tamarin che vanno a verificare le proprietà del protocollo, queste proprietà vengono descritte nelle rule. Supponendo il caso base, nella quale un lemma analizza una singola proprietà, Tamarin Prover inizia la validazione della proprietà partendo dalla rule che la racchiude, procedendo a ritroso nel tentativo di risolvere tutti i fatti descritti dalla rule. Vi sono molteplici tipologie di derivazione, la più classica di Tamarin Prover è quella per induzione. Tamarin Prover, inoltre, analizza tutte le possibili tracce che possono portare alla verifica della proprietà di default, noi possiamo specificare di richiedere anche una singola traccia che soddisfa la proprietà. Grazie al comando `interactive`, viene istanziato il sito nella quale possiamo interagire con Tamarin Prover e vedere le rappresentazioni grafiche delle proprietà del protocollo. Possiamo anche derivare manualmente il lemma; Codice 2.3.

```
lemma Ex_Protocol_Complete:
    exists-trace
      "Ex User Browser #i. Protocol(User, Browser)@i"

lemma All_Traces:
      "(All User1 password1 User2 password2 #i #j.
                Password_Secure(password1, User1)@i &
                Password_Secure(password2, User2)@j  ==> #i=#
   j)"
```

Codice 2.3: Esempi di lemmi

Esistono molteplici composizioni dei lemmi e la possibilità i comporli tra loro grazie alle keyword `reuse` e `hide_lemma`, in questo caso solo il lemma `Protocol_Honest` considera anche la proprietà definita nel lemma `Private_Key`; Codice 2.4.

```
lemma Private_Key[reuse]:
    "not (Ex key #r. PrivateKey(key)@r )"

lemma Ex_Protocol_Complete:
    exists-trace
      "Ex User Browser #i. Protocol(User, Browser)@i"

lemma Protocol[sources]:
    "All User password #c.
    Completed_Protocol(password, password, User)@c
            ==> Ex #z. Password(password, User)@z"

lemma Protocol_Honest[hide_lemma=Private_Key]:
    exists-trace
    "Ex nonceA nonceB A B #i. AHasCompletedProtocol(A, B,
   nonceA, nonceB) @i &
    (Ex #j. BHasCompletedProtocol(A, B, nonceA, nonceB)@j & j
   <i )"
```

Codice 2.4: Esempi di lemmi con utilizzo di direttive avanzate

- `Restriction`

  Una restriction può essere vista come una proprietà di un lemma persistente nel protocollo. Supponiamo di descrivere un protocollo e di considerare una proprietà riccorrente in ogni lemma che definiamo, risulta essere più efficiente descrivere quest'ultima una sola volta. Questa soluzione è tanto efficiente quanto pericolosa e critica, in quanto potrebbe tagliare percorsi di derivazione del protocollo , di conseguenza, eventuali attacchi. L'utilizzo, quindi, di questi costrutti di Tamarin deve essere centro di particolare attenzione; Codice 2.5.

```
1  restriction OnlyUser:
2    "All User1 User2 #i #j. New_User(User1) @i & New_User(User2
     ) @j
3              ==> Equal(User1, User2) & Equal(#i,#j)"
4
5  restriction Create_Password_Generate:
6    "All #i #j. Create_Password() @i &
7                Create_Password() @j ==> Equal(#i,#j)"
8
9  restriction Password_Secure_Generate:
10   "All password user #i #j. Password_Secure(password, user)
     @i &
11                             Password_Secure(password, user)
     @j
12             ==> Equal(#i,#j) & Equal(password, password) &
13                 Equal(user, user)"
```

Codice 2.5: Esempio di restriction

Questi tre costrutti formano la base di un file Tamarin, che verrà dato al tool Tamarin Prover per verificarne se la proprietà del protocollo viene soddisfatta o se esistono traccie d'attacco che la violano.

Tamarin Prover

Fornisce supporto algoritmico per cercare un controesempio alla proprietà di sicurezza che si vuole analizzare, sfruttando le regole precedentemente descritte per il protocollo e l'attaccante.

Tamarin Prover è un dimostratore di teoremi, ovvero verifica le proprietà di sicurezza di un protocollo tramite un insieme di multi-set di regole crittografiche. Avremo quindi:

1. `Sistema R` = insieme di regole crittografiche
2. `Formula @` = proprietà da verificare

Tamarin Prover può verificarne la validità o la soddisfacibilità di `@` per le tracce di R. Un aspetto fondamentale di Tamarin Prover, come già anticipato, è come esegue le derivazioni dei protocolli. Sappiamo che un lemma verifica una o più propietà di un protocollo. Per eseguire questi calcoli, il tool parte dalla regola (rule) nella quale è istanziata la proprietà in esame e procede a derivare tutti i fatti in ingresso alla regola, in quanto precondizioni per l'esecuzione di quest'ultima.

Questo processo si protrae per le nuove rule nella quale sono definiti i fatti in uscita e così fino a quando non si arriva alle regola iniziali, avremo quindi una derivazione delle regola per induzione. Se Tamarin Prover non trova le regola per soddisfare il lemma questo sarà considerato falso. Supponiamo d'avere una rule, questa, per essere soddifatta, ha bisogno che i fatti in entrata siano soddisfatti, quest'ultimi si possono trovare nelle restanti rule che compongono il protocollo.

Nasce quindi tutto il discorso delle Restiction, ovvero proprietà persistenti del protocollo che escludono un insieme di regole, ovvero tracce d'esecuzione del protocollo, riducendo così quelle disponibili per la derivazione del protocollo.

Una restiction, quindi, potrebbe escludere proprio la rule che ci serve per validare il lemma, questo non significa che dobbiamo sempre ottenere dei lemmi validati, infatti il risultato di un lemma a volte deve risultare falso. Ciò dipende dalla proprietà che analizza.

In conclusione, i lemmi vengono definiti prima della formalizzazione di un protocollo, in quanto modellano le proprietà di sicurezza che devono essere verificate. Se questi invece presentano valori inattesi possiamo consultare il sito generato da Tamarin Prover per vedere la rappresentazione grafica del protocollo e vedere in quali punti non vengono rispettate le proprietà.

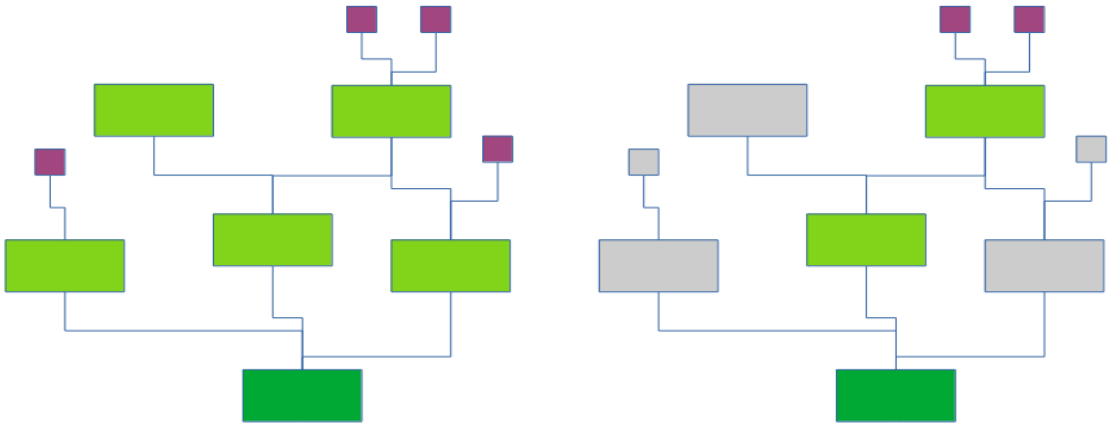

Figura 2.5: Derivazione Tamarin Prover

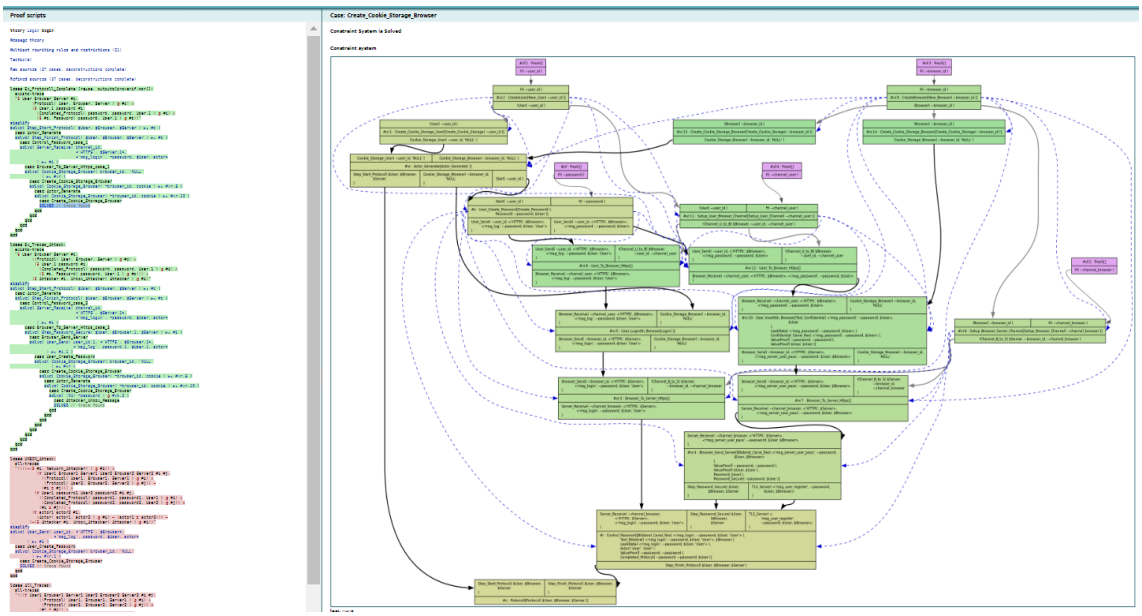

Figura 2.6: Modalità Interattiva

La Figura 2.5 presenta i due scenari, il primo senza restiction e quindi tutte le tracce di esecuzione del protocollo, mentre il secondo con le restriction e quindi un insieme di tracce ridotte.

Tamarin Prover fornisce due modi per costruire dimostrazioni. Ha una modalità efficiente completamente automatizzata che combina deduzione e ragionamento equazionale con euristiche per guidare la ricerca delle prove.

- **Modalità Interattiva**

  Fornisce all'utente le modalità per interagire con il prover durante la ricerca di prove, come mostrato in Figura 2.6. Ritorna una prova di correttezza o un controesempio, che rappresenta un attacco che viola la proprietà dichiarata. L'applicazione di questa modalità di Tamarin Prover risulta essere più utile per protocolli complessi, infatti altri strumenti supportano solo la costruzione di prove completamente automatizzate senza

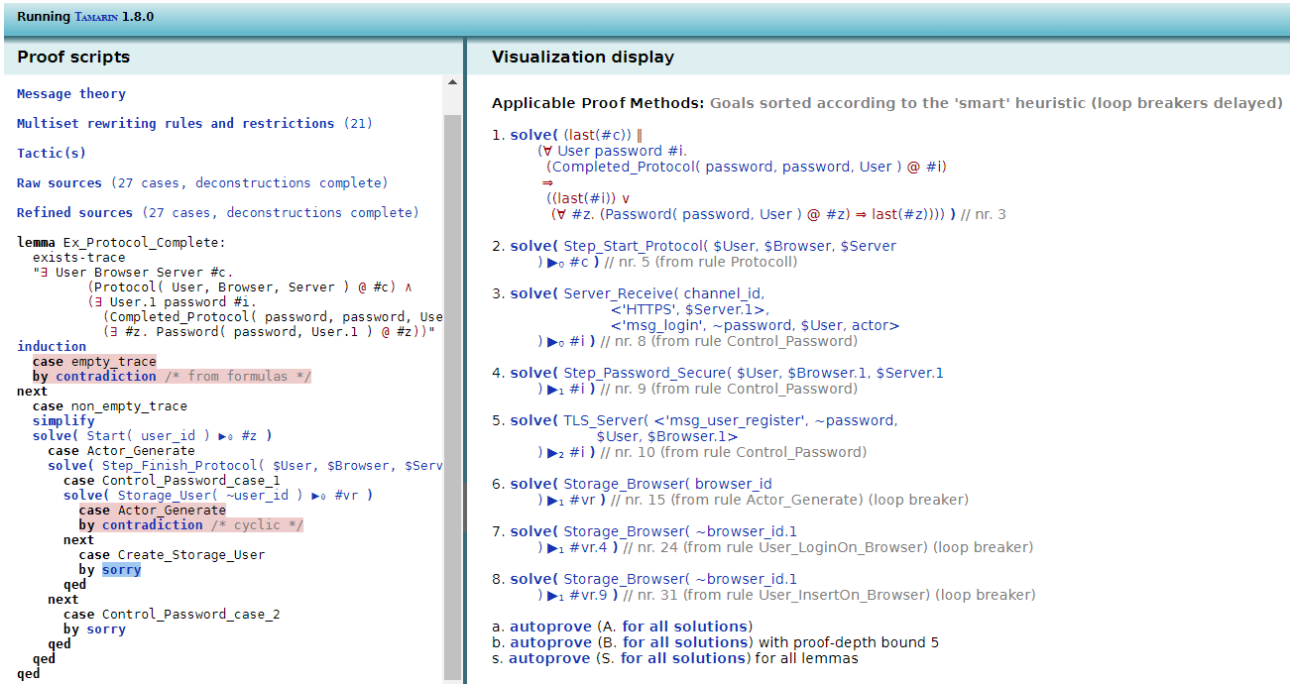

Figura 2.7: Modalità Interattiva con Derivazione Induttiva

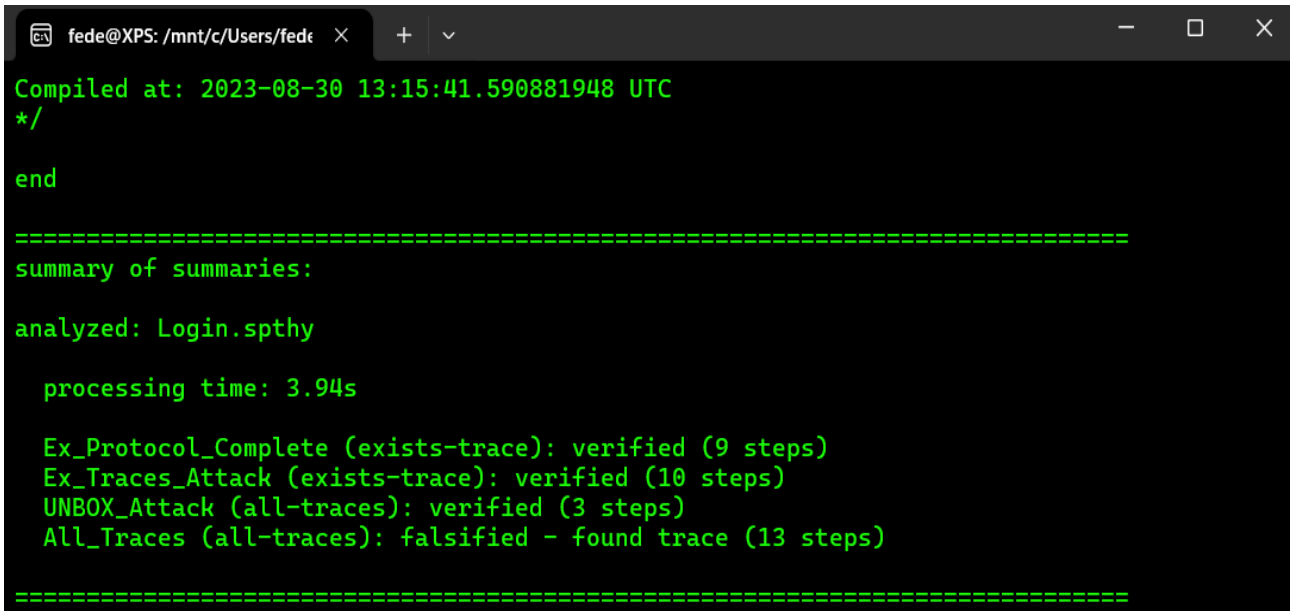

Figura 2.8: Modalità Console

poterne svolgere un analisi più dettagliata. La Figura 2.7 rappresenta la modalità di interazione del tool per la derivazione di una proprietà di sicurezza, possimo notare i fatti precedentemente scritti nelle regole per completare le derivazioni per induzione.

- **Modalità Console**

  Questa modalità, Figura 2.8, non offre la possibilità di derivare manualemente le derivazioni come la precedente, inoltre non offre alcun tipo di immagine. In questa modalità andiamo a vedere solo se le proprietà sono soddisfatte.

Output

Tamarin Prover fornisce supporto algoritmico per cercare un controesempio alla proprietà di sicurezza, che rappresenta quindi un potenziale attacco al protocollo. In alternativa, quando non esiste un controesempio, il tool costruisce una dimostrazione.

- *Attacco (Attack)* Se è presente un attacco o la proprietà non può essere verificata. Tamarin Prover, in modalità interattiva, proporrà un controesempio o la traccia del attacco, ovviamente il lemma deve essere realizzato secondo le specifiche adatte.
- *Dimostrazione (Proof)* Se le rule descritte sono sufficienti e la proprietà del lemma viene verificata.
- *Terminazione (Kill)* Se sono presenti errori di implementazione o impossibilità di completare l'analisi terminerà in modo prematuro. A seconda della gravità di tale errore Tamarin Prover potrebbe anche non produrre alcun riscontro sia in modalità console sia in interattiva. Nel caso peggiore bisognerà revisionare i file Tamarin in input e cercare, con l'aiuto del manuale, l'errore.

Bisogna prestare attenzione ai risultati di tipo *Proof* e *Attack*, infatti quest'ultimi potrebbero essere dei falsi positivi o negativi. Tamarin Prover utilizza, per verificare la proprietà, le regole e per essere certi che queste descrivano esattamente gli step che vogliamo nel protocollo è sempre opportuno studiare i grafici prodotti dalla modalità interattiva. Infatti una proprietà potrebbe essere verificata, ma la sequenza delle azioni del protocollo potrebbero non essere quelle che si vogliono, in quanto è Tamarin Prover che sceglie le regole che ritiene migliori.

Capitolo 3

Stato dell'Arte

3.1 Model Checker

Esistono e vengono realizzati molteplici protocolli crittografici dal punto di vista teorico, il problema è che eventuali problemi del protocollo compaiono solo in fase di implementazione finale. Nasce quindi la domanda di capire se esistono strumenti efficaci per vederne la loro reale concretezza, in modo tale da avere prove formali della proprietà di sicurezza attesa.

3.1.1 Thread di Ricerca

Un contributo molto importante nel modo della ricerca viene dalla pubblicazione [7] di Edmund Melson Clarke, E Allen Emerson e Aravinda Prasad Sistla nel 1986. Questo è il primo lavoro che propone un nuovo approccio per la modellazione dei protocolli, ovvero viene formalizzata una procedura efficiente per verificare che un sistema simultaneo a stati finiti soddisfi una specifica, quest'ultima viene espressa in una logica temporale (proposizionale, tempo di ramificazione), proponendo un algoritmo con complessità lineare sia nella dimensione della specifica che nella dimensione del grafico di stato globale per il sistema simultaneo. Inoltre, in questa ricerca, si sostiene che la tecnica proposta può offrire un'alternativa pratica alla costruzione manuale di dimostrazioni o all'uso di un dimostratore di teoremi meccanico, in modo tale da verificare più sistemi concorrenti a stati finiti. I risultati sperimentali mostrano che le macchine con diverse centinaia di stati possono essere controllati in pochi secondi. Questo algoritmo e tecnica prendono il nome di model checking.

La ricerca nel campo della verifica dei protocolli crittografici vede anche un'altro importante contributo con la ricerca di Lawrence C. Paulson, [16] del 1998. Infatti, precedentemente, si utilizzavano gli approcci dell'esplorazione dello stato e una logica temporale. L'approccio proposto è basato sul calcolo dei predicati nei stati infiniti, generando delle prove utilizzando il Tool di *Isabelle/HOL*. Lo svantaggio iniziale risiedeva sullo sforzo umano richiesto per analizzare un protocollo che può richiedere anche più di una settimana, con questo approccio, invece, viene prodotto uno script di prova che ne richiede alcuni minuti. In questo contesto i protocolli

sono induttivamente definiti come insiemi di tracce. Una traccia è un elenco di eventi di comunicazione, includendo anche attacchi e perdite accidentali dei messaggi.

Un ulteriore contributo alle tecniche di verifica dei model checker è stato dato con [5] di Sergey Berezin nel 2002. La problematica, messa in luce in questa ricerca, risiede sugli strumenti sempre più moderni e efficienti che rendendo l'analisi degli stati finiti sempre più complessa. Si vuole quindi trovare una combinazione efficiente tra controllo del modello e miglioramento del modello; la ricerca di tale combinazione è stata a lungo una delle sfide più importanti nel campo della verifica formale. In questo lavoro, quindi, viene presentato un nuovo framework che include sia il controllo del modello (*model checking*) che il miglioramento dei teoremi (*teorem proving*), combinando insieme queste tecniche otteniamo *SyMPY (Symbolic Model Prover)*, ovvero un dimostratore di modelli simbolici. Il sistema risultante, quindi, ha la stessa potenza espressiva di un dimostratore di teoremi, ma allo stesso tempo gode dello stesso grado di automazione e velocità del model-checker per la parte di verifica a stati finiti.

3.2 Analisi di protocolli

I ricercatori del Centro per la Cybersecurity della Fondazione Bruno Kessler hanno recentemente presentato il design di alcuni protocolli di Identity Management e la relativa analisi di sicurezza basata su metodi formali.

3.2.1 Protocollo di Autenticazione basato su Push Notifications

Articolo di Marco Pernpruner, Roberto Carbone, Silvio Ranise e Giada Sciarretta: [17]. Viene presentato il progetto di un sistema senza password, soluzione di autenticazione a più fattori che consente ai cittadini di autenticarsi e di accedere ai servizi online dal proprio computer, utilizzando le loro carte d'identità elettroniche. Il protocollo si basa su tecniche quali notifiche push, ovvero strumenti e modelli raccomandati dal NIST. In questo lavoro sono stati rilevati, inoltre, alcuni attacchi imprevisti: sebbene questi non siano in grado di compromettere esplicitamente tutto il fattori di autenticazione, riescono a rompere implicitamente il protocollo, inducendo la vittima a compromettere i restanti fattori per conto degli aggressori.

3.2.2 Protocollo di Autenticazione basato su QR Codes

Pubblicazione di Marco Pernpruner, Roberto Carbone, Giada Sciarretta e Silvio Ranise: [18]. Questo articolo vuole migliorare alcuni aspetti del precedente articolo [17], infatti propone l'utilizzo dei QR-code come uno dei passaggi per l'autenticazione. Viene inoltre presentata una metologia di sicurezza a più livelli: analisi combinatoria, analisi simbolica, e analisi dei rischi. In particolare, lo strato combinatorio richiede specifiche ad alto livello ed esegue una rapida analisi, permettendo di ridurre il numero di aggressori dei test nel livello successivo, che è più computazionale costoso. Il secondo livello, rappresentato dall'analisi simbolica, fornisce un quadro completo dei risultati, basandosi su framework avanzati che richiedono specifiche

formalità con dettagli crittografici. Infine, l'analisi dei rischi raccoglie i risultati delle analisi precedenti e li integra con una valutazione del rischio che fornisce la base per pianificare le mitigazioni.

La Tabella 3.1 mostra la modellazione degli attacchi nel protocollo [18] scritti in ASLan++.

Tipologia Attacco	Senza Attaccante	Con Attaccante

PCT	`authentic_on(Ch_U2B,User);`
	`userOwnComputer;`
	–
	–
		–
		–
		`weakly_authentic(Ch_U2B);`
		Use same channel `ch_U2B` in sessions

MDT	`authentic_on(Ch_U2EICApp,User);`
	`userOwnSmartphone;`
	–
	–
		–
		–
		`weakly_authentic(Ch_U2EICApp);`
		Use same channel `ch_U2EICApp` in sessions

CT	`authentic_on(Ch_U2EIC,User);`
	`userOwnEIC;`
	–
	–
		–
		–
		`weakly_authentic(Ch_U2EIC);`
		Use same channel `ch_U2EIC` in sessions

D	–	`iknows(PIN);`

ES, SS	`confidential_to(Ch_U2EICApp,EICApp);`
	`confidential_to(Ch_U2B, Browser);`
	`confidential_to(Ch_EICApp2U,User);`
		–
		–
		–

SE	`confidential_to(Ch_U2B,Browser);`
	`authentic_on(Ch_B2U,Browser);`
	–
	–
		–
		–
		`iknows(PIN);`
		`iknows(OTP);`

MB	–	Replace `browser` with `i` in one session

MM	`authentic_on(Ch_EICApp2U,EICApp);`
	`authentic_on(Ch_EICApp2EIC,EICApp);`
	–
		–
		–
		Replace `eicapp` with `i` in one session

Tabella 3.1: Esempio di modellazione degli attaccanti in ASLan++

Capitolo 4

Analisi protocollo Needham-Schroeder

4.1 Descrizione del protocollo

Introduciamo il protocollo di Needham-Schroeder [14]. La scelta di questo protocollo non è casuale, infatti è presente come spunto iniziale in molti studi in campo di sicurezza informatica, come per ASLan++ e Tamarin. Esistono diverse versioni di questo protocollo, noi ci focalizzeremo su quello a chiave pubblica, basato sulla crittografia asimmetrica, dove ad ogni attore coinvolto nella comunicazione viene associata una coppia di chiavi:

(i) *chiave pubblica – distribuita*

(ii) *chiave privata – personale e segreta*

Ciò permette di assicurare la mutua autenticazione tra due entità di rete, in quanto solo la chiave privata riesce ad aprire un messaggio cifrato con la chiave pubblica associata. Si andrà quindi a stabilire una chiave di sessione utilizzabile da due entità di rete per proteggere le successive comunicazioni, ovvero si passa a una crittografia a chiave simmetrica. Le seguenti sezioni descrivono il protocollo NSPK, le figure che rappresentano gli step del protocollo sono state prese dal manuale di ASLan++ [21].

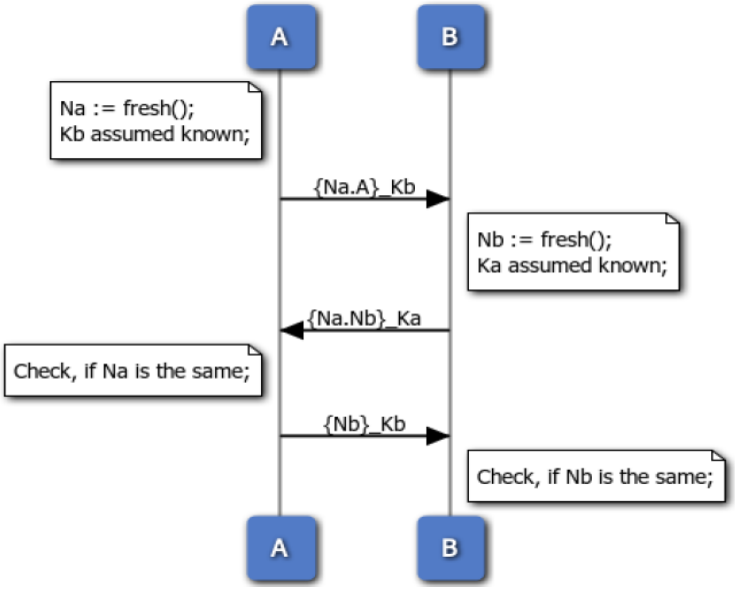

Figura 4.1: NSPK Base

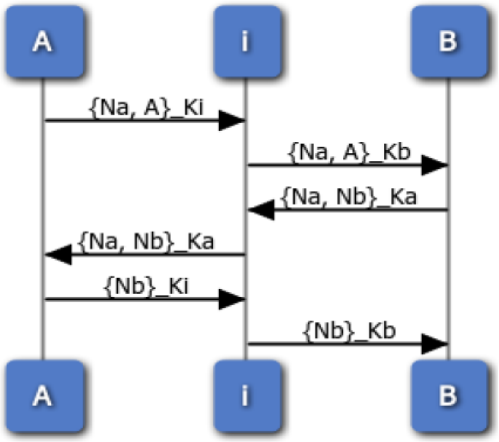

Figura 4.2: NSPK Attack

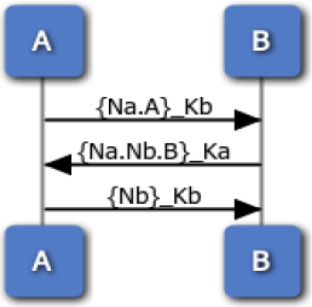

Figura 4.3: NSPK Fix

NSPK BASE

La Figura 4.1 rappresenta gli step che compongono il protocollo di Needham-Schroeder: A e B creano i corrispettivi nonce e si autenticano mutualmente sfruttando le rispettive chiavi pubbliche e private.

Il protocollo inizia quindi con A che cripta e spedisce il messaggio `[Na.A]_Kb` con la chiave pubblica di B; successivamente B, con la sua chiave privata, potrà aprire il messaggio e leggere il suo contenuto: `[[Na.A]_Kb]_Kbpri = Na.A`. In seguito B spedisce il proprio nonce `Nb`, insieme a quello ricevuto `Na`, ad A che controllerà il messaggio e invierà un messaggio di conferma del nonce ricevuto `Nb` ad B. B quindi verificherà la correttezza di `Nb` terminando il protocollo. Il protocollo si concluderà con A e B che si sono mutualmente autenticati e sono gli unici a conoscenza dei nonce `Na` e `Nb`.

NSPK ATTACK

La Figura 4.2 rappresenta l'attacco conosciuto per NSPK. Questo attacco fu scoperto da Gavin Lowe nel 1995 [11]: consideriamo un attaccante "i" che si insinua nello scambio di messaggi di A e B, questa tipologia d'attacco è conosciuta come "Man in the Middle", ovvero "uomo nel mezzo - *MIM*".

Questo attacco avrà quindi "i" che vuole ingannare A, convincendola ad iniziare una sessione di comunicazione con lui. Successivamente "i" inoltra i messaggi a B, convincendolo di essere in comunicazione con A. Otteniamo così che "i" intercetta i messaggi tra A e B completando il protocollo e facendo credere a B di essere in comunicazione sicura con A. L'attaccante sarà quindi a conoscenza dei nonce `Na` e `Nb`.

NSPK FIX

La Figura 4.3 rappresenta la versione sicura del protocollo: B inserisce la propria identità oltre ai due nonce, mitigando così l'attacco precedentemente descritto.

Nel dettaglio B, quando invia il messaggio ad A, modifica il messaggio della versione base `[Na.Nb]_Ka` con `[Na.Nb.B]_Ka` aggiungendo `B` al messaggio, ovvero l'identità di B. Questo permette ad A d'accorgersi che i messaggi non arrivano dall'intruso "i", con cui aveva iniziato la comunicazione, ma da B interrompendo il protocollo. Infatti "i" non è in grado di aprire e modificare il messaggio senza la chiave privata di A: `[[Na.Nb.B]_Ka]_Kapri != [[Na.Nb.B]_Ka]_Kipri` e quindi A capisce che è in corso un attacco.

L'obbiettivo di questo capitolo è l'analisi e formalizzazione di questo protocollo in entrambi i tool di Tamarin Prover e SATMC, in modo tale da studiarne le differenti tecniche e in particolar modo, con Tamarin, scoprire a pieno le funzionalità.

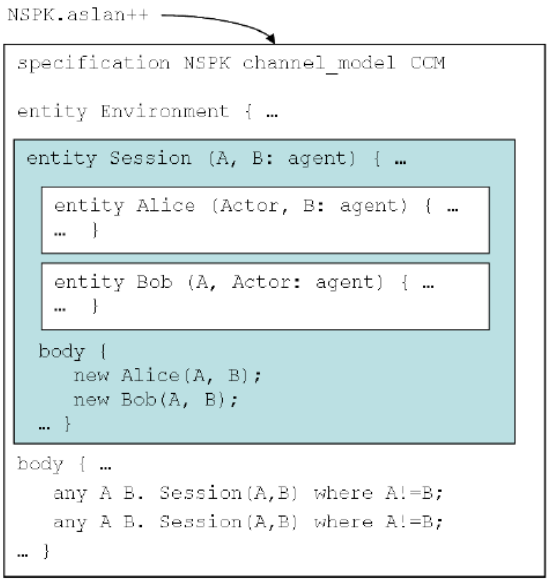

Figura 4.4: NSPK ASLan++

4.2 Modellazione in ASLan++

La figura 4.4 mostra lo pseudo codice del protocollo di NSPK in ASLan++. Possiamo notare i principali elementi costitutivi di ASLan++ quali le "entity", simili alle classi in Java o ai ruoli in contesto dei protocolli di sicurezza e comunicazione. Le entity dichiarano vari simboli, macro, obiettivi e altri elementi. Inoltre possono avere parametri e contenere sottoentità nidificate.

(i) `Environment` – radice globale del sistema da specificare, in modo simile alla routine "main"

(ii) `Session` – contiene entità comunicanti, ovvero gli attori del protocollo

(iii) `Body` – viene specificato il comportamento dell'entità o gli obiettivi di sicurezza. Quando l'enità viene istanziata, viene eseguito il corpo

Per trovare un attacco al NSPK in ASLan++ è necessario condurre almeno due sessioni in parallelo: Codice 4.1.

```
1 body { % need two sessions for Lowe's attack
2     any A B. Session (A,B) where !A=B;
3     any A B. Session (A,B) where !A=B;
4 }
```

Codice 4.1: Sessioni parallele NSPK in ASLan++

Vengono istanziate due "symbolic sessions" tra A e B, dove A potrebbe essere un'istanza onesta di Alice o un aggressore disonesto e B potrebbe essere un esempio onesto di Bob o un aggressore. Il model checker valuterà quindi le combinazioni per le due sessioni parallele, tenendo conto delle simmetrie.

4.3 Modellazione in Tamarin

Viene presentato questo protocollo in quanto presente nel manuale di Tamarin per fungere come base di partenza per la comprensione del linguaggio.

Utilizzeremo questo protocollo per introdurre i diversi aspetti e costrutti di Tamarin, oltre a fornire uno studio più approfondito rispetto a quello presentato nel manuale, in modo tale da indirizzarci alle tecniche e modelli presentati nei capitoli successivi.

4.3.1 Protocollo

- **Inizializzazione degli Attori**

  La formulazione del protocollo di Needham-Schroeder in Tamarin prevede la definizione di 2 attori che si scambiano un insieme di messaggi cifrati al fine di autenticarsi mutualmente: Codice 4.2.

```
1  rule CreateAlice:
2    // Chiave privata Alice
3    [ Fr(~privateKeyA) ]
4    --[ Alice($A)]->
5    // Fatti per le successive Rule
6    [ !PrivateKeyAssociationA($A, ~privateKeyA),
7      !PublicKeyAssociationA($A, pk(~privateKeyA)) ]
8
9  rule CreateBob:
10   // Chiave privata Bob
11   [ Fr(~privateKeyB) ]
12   --[ Bob($B) ]->
13   // Fatti per le successive Rule
14   [ !PrivateKeyAssociationB($B, ~privateKeyB),
15     !PublicKeyAssociationB($B, pk(~privateKeyB)) ]
```

Codice 4.2: Inizializzazione Alice e Bob

- **Esecuzione del Protocollo**

  Alice e Bob quindi procedono a scambiarsi i messaggi cifrati. Nel nostro esempio viene considerata la variante sicura con il messaggio aggiornato con l'identità di Bob: Codice 4.3.

```
rule Step1_A_sends_to_B_nonceA_and_identityA:
let
  firstMessage = aenc{<'1', ~nonceA, $A>}publicKeyOfB
in
[ // Alice crea il suo nonce
  Fr(~nonceA) ]
--[
  Out(firstMessage),
  Step1($A, $B, ~nonceA, ~privateKeyOfA, publicKeyOfB) ]

rule Step3_A_sends_to_B_nonceB:
let
  answerFromB = aenc{<'2', ~nonceA, nonceB, identityOfB>} pk(
    ~privateKeyOfA)
  myNonce = fst(snd(adec{answerFromB}~privateKeyOfA))
in
[
  Step1(A, identityFromB, myNonce, ~privateKeyOfA,
   publicKeyOfB),
  In(answerFromB)
]--[
  // Alice controlla l'identita' di Bob
  IdentityProof(identityFromB, IdentityFromB( adec{
   answerFromB} ~privateKeyOfA) )
]->[ Out(thirdMessage) ]

rule Step2_B_sends_to_A_nonceA_and_nonceB:
let
  firstMessage = aenc{<'1', nonceA, $A>}pk(~privateKeyOfB)
  secondMessage = aenc{<'2', receivedNonce, ~nonceB, $B>}
   publicKeyOfA
in
[ // Bob crea il suo nonce
  Fr(~nonceB), In(firstMessage)
]--[
  // Bob controlla l'identita' di Alice
  IdentityProof(receivedNonce, nonceA), IdentityProof(
   receivedEntity, $A)
]->[
  Out(secondMessage),
  Step2(receivedEntity, $B, receivedNonce, ~nonceB, ~
   privateKeyOfB , publicKeyOfA) ]
```

Codice 4.3: Step Protocollo

- **Conclusione**

  L'ultima Rule prevede che Bob controlli l'identità di Alice grazie al messaggio aggiornato, quindi completa con successo il protocollo validando l'identità di Alice: Codice 4.4.

```
rule Step4_B_receives_last_message:
let
  myNonce = snd(adec{aenc{<'3', ~nonceB>}pk(~privateKeyOfB)}~
   privateKeyOfB)
in
[
  Step2($A, $B, nonceA, myNonce, ~privateKeyOfB, publicKeyOfA
   ),
  In(thirdMessage)
]
   // Nonce finale del protocollo
--[ Test_Nonce_Final_AB(nonceA, myNonce), Protocol() ]-> []
```

Codice 4.4: Completamento Protocollo

Tutti i codici presentati nelle sezioni di `Actor`, `Step Nonce Protocol` e `Complete Protocol` sono solo una parte del protocollo completo, infatti sono presenate solo parti fondamentali. I codici completi sono disponibili nel repository offerto da Zurigo, [22].

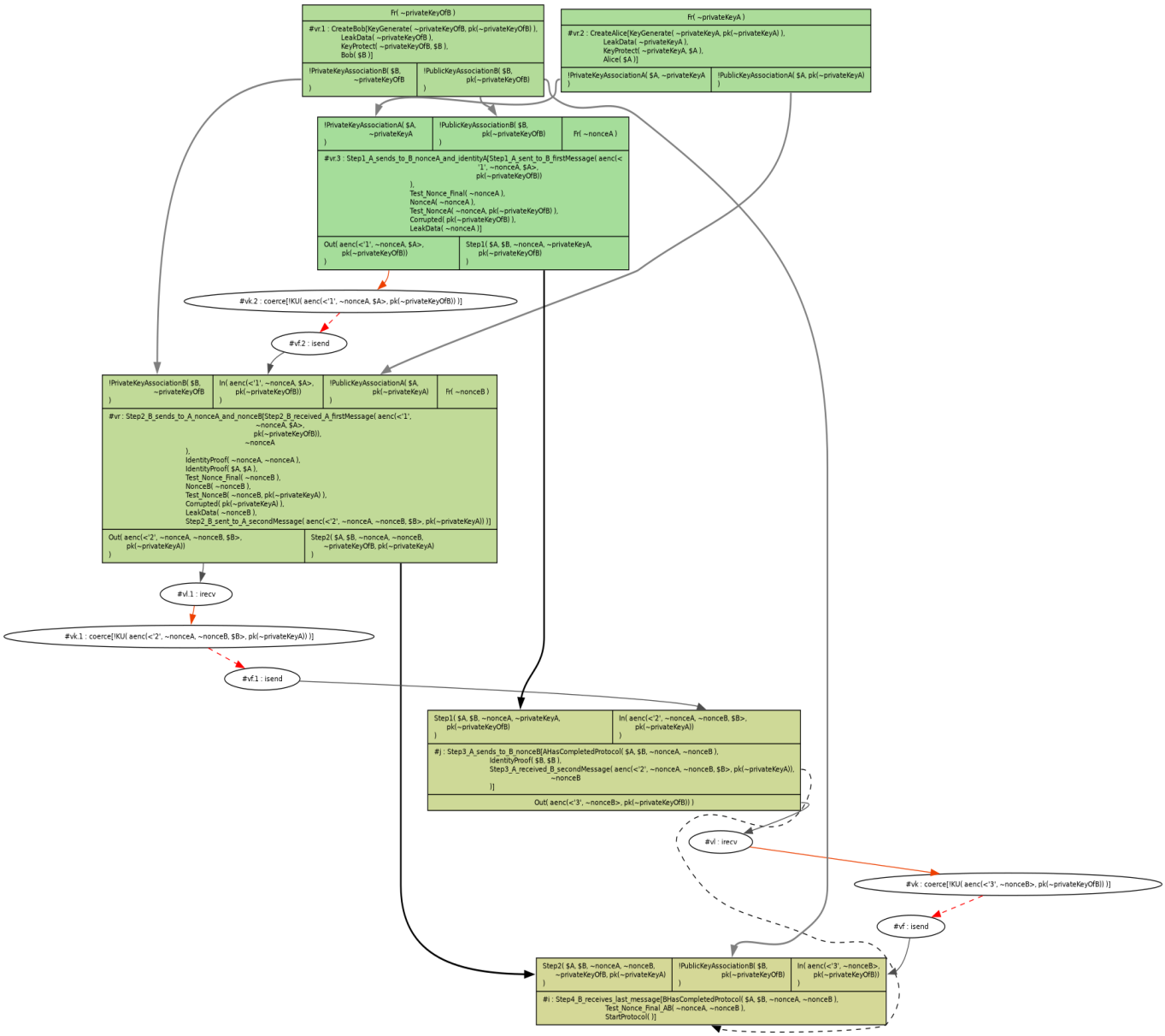

Figura 4.5: Traccia del protocollo NSPK ottenuta da Tamarin Prover

La Figura 4.5 presenta la rappresentazione finale del protocollo prodotta da Tamarin Prover. Possiamo osservare diverse tipologie di rule e frecce, non potendoci soffermare su tutti i casi qui di seguito una veloce considerazione:

- **Rule**

  Le Rule di colore verde rappresentano quelle descritte, mentre quelle bianche vengono generate da Tamarin per completare le derivazioni.

- **Frecce**

  Notiamo 3 differenti colori: il grigio e nero sono per il normale funzionamento del protocollo e il passaggio dei messaggi tra le rule, le rosse rappresentano l'attaccante, fortunatamente, in questo caso, i messaggi sono cifrati dalle chiavi.

4.3.2 Sessione Parallela

Formalizzazione di un lemma che prevede la stessa proprietà del protocollo in istanti di tempo diversi, dalla quale si ottengono due sessioni parallele del protocollo: Codice 4.5.

```
lemma Two_Session:
    exists-trace
    "Ex #i. Protocol() @i & (Ex #j. Protocol() @j & j<i )"
```

Codice 4.5: Sessioni Parallele

Ovviamente bisogna prestare attenzione alle definizioni delle "Restiction" e "Rule", in quanto le sessioni potrebbero utlizzare le stesse variabili. Questo discorso apre un nuovo capitolo fondamentale sulle analisi dei protocolli, in quanto le "Restiction" possono tagliare tracce di derivazione del protocollo, impedendo la rappresentazione di attacchi ed eventuali vulnerabilità del protocollo.

4.3.3 Tempi di Esecuzione

Presentata la stuttura del protocollo vogliamo ora concentrarci sui differenti tempi di esecuzione, con l'utilizzo di un insieme di tecniche e costrutti differenti offerti da Tamarin, analizzando nel dettaglio le proprietà del protocollo.

I lemmi che seguiranno sono i medesimi per tutte e tre le tipologie di analisi. La differenza sta nella definizione differente di specifici punti del protocollo, ovvero la gestione dei messaggi che intercorrono tra gli attori. Quello che si otterranno saranno quindi 3 strutture differenti del protocollo, ma che analizzano lo stesso spazio di definizione e proprietà.

Tamarin offre diverse metologie per descrivere stessi punti del linguaggio, noi vedremo le seguenti:

- **Macro**

  Possiamo considerare le Macro come una sorta di funzione simile ai linguaggi più comuni, in realtà sono una precompilazione di costrutti già presenti di Tamarin, in questo caso `fst` e `snd`: Codice 4.9.

```
macros: IdentityFromB(x) = snd(snd(snd(x))),
        NonceFromB(x) = fst(snd(snd(x))),
        MyNonce(x) = fst(snd(x))
```

Codice 4.6: Macro

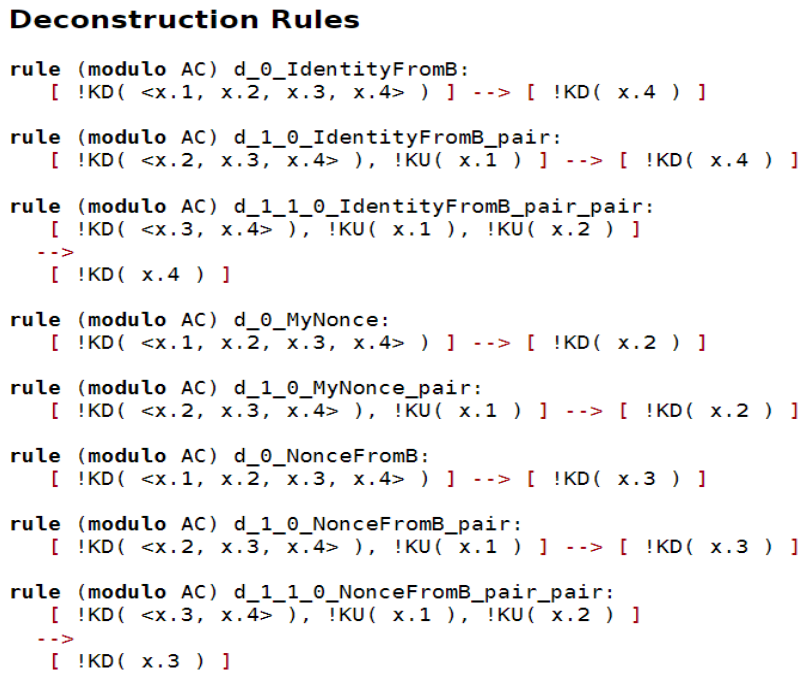

Figura 4.6: Regole di decostruzione in Tamarin

- **Equazioni**

  Le Equazioni sono completamente diverse dalle Macro, anche se la stuttura sembra ricordarle. Tamarin Prover, per ogni equazione, va a definirsi delle rule che le compongono, ovvero, con l'ausili dei costrutti fondamentali `!KU` e `!KD`, ricostruisce, per induzione, tutti i passaggi per definire un equazione, come visibile in Figura 4.6. Inoltre questa procedura di derivazione viene effettuata per ogni esecuzione nel protocollo, infatti non vi è una fase di precompilazione: Codice 4.10.

```
1 functions: IdentityFromB/1, MyNonce/1, NonceFromB/1
2
3 equations:,MyNonce(<x.1, x.2, x.3, x.4>) = x.2
4           ,NonceFromB(<x.1, x.2, x.3, x.4>) = x.3
5           ,IdentityFromB(<x.1, x.2, x.3, x.4>) = x.4
```

Codice 4.7: Equazioni

Qui vengono presentate le analisi finali delle tre tipologie di analisi. La prima analisi rappresenta la definizione del protocollo senza l'utilizzo di costrutti particolari, le altre due, invece, racchiudono i costrutti presentati. Noteremo, come prima cosa, una notevole differenza tra i tempi di esecuzione, soprattutto tra Macro e Equazioni, oltre ad un notevole aumento degli step che i lemmi eseguono per verificare le proprietà.

Protocollo base

```
processing time: 16.31s:

  Honest (exists-trace): verified (20 steps)
  protocol_honest_execution (exists-trace): verified (9 steps)
  protocol_authentication (all-traces): verified (35 steps)
  types (all-traces): verified (111 steps)
```

Codice 4.8: Tempi di esecuzione del protocollo base

Macro

```
processing time: 4.06s:

  Honest (exists-trace): verified (20 steps)
  protocol_honest_execution (exists-trace): verified (9 steps)
  protocol_authentication (all-traces): verified (10 steps)
  types (all-traces): verified (43 steps)
```

Codice 4.9: Tempi di esecuzione del protocollo con macro

Equazioni

```
processing time: 35.02s

    Honest (exists-trace): verified (14 steps)
    protocol_honest_execution (exists-trace): verified (7 steps)
    protocol_authentication (all-traces): verified (442 steps)
    types (all-traces): verified (99 steps)
```

Codice 4.10: Tempi di esecuzione del protocollo con equazioni

4.3.4 Lezioni Apprese

Durante lo studio iniziale del protocollo di NSPK sono emerse diverse sfide, sia nei risultati dell'analisi che nei comportamenti di Tamarin Prover, che hanno richiesto un ulteriore tempo per la scoperta sia del Tool che del linguaggio, carpendone aspetti anche complessi e considerazioni che hanno portato alla definizione degli apporti innovativi, questi verranno presentati nel prossimo capitolo.

Attori del protocollo

Un aspetto fondamentale nella costruzione e formalizzazione di ogni protocollo di sicurezza è quella della rappresentazione e conoscenza degli attori che ne fanno parte; questa caratteristica sembra non essere scontata in Tamarin, o meglio in Tamarin Prover. Il comportamento anomalo riscontrato era quello di creazione e inserimento di nuovi attori durante l'esecuzione del protocollo, questa condotta è stata evidenziata soprattutto nei test offerti da Tamarin, ovvero dei lemmi speciali nella quale si proteggono valori specifici: Codice 4.11.

```
1 // key_pri da proteggere
2 test leak_KeyGenerate:
3   "Ex key_pri #i. KeyGenerate(key_pri, key_pub)@i"
4
5 lemma Key_Test:
6   leak_KeyGenerate accounts for
7   "All key_pri key_pub #i. KeyGenerate(key_pri, key_pub)@i ==>
    not (Ex #z. LeakData(key_pri)@z )"
```

Codice 4.11: Esempio di test

Più nel dettaglio, un test viene creato con un lemma nel quale si specifica una keyword da proteggere, un esempio può essere un'informazione contenuta in un messaggio in ingresso alla rule. Questo lemma andrà a generare automaticamente 7 proprietà, ognuna delle quali analizza un aspetto preciso della proprietà del lemma, la principale che condiziona le altre è la "non-empty", infatti se quest'ultima non è verificata bisogna revisionare il lavoro fatto, evitiamo quindi la presentazione nel dettaglio in quanto richiedebbe troppo tempo.

Possiamo però affermare che per ognuna delle proprietà, il Tool Tamarin Prover sempre in modalità `interactive`, crea le corrispondenti rappresentazioni grafiche del comportamento, queste però presentano solo alcune sezioni del protocollo totale e quindi possono anche diferire dal comportamento totale del protocollo.

Ciononostante queste sezioni grafiche del protocollo hanno evidenziato dei comportamenti anomali che si pensavano già trattati o considerati in fase di analisi del tool.

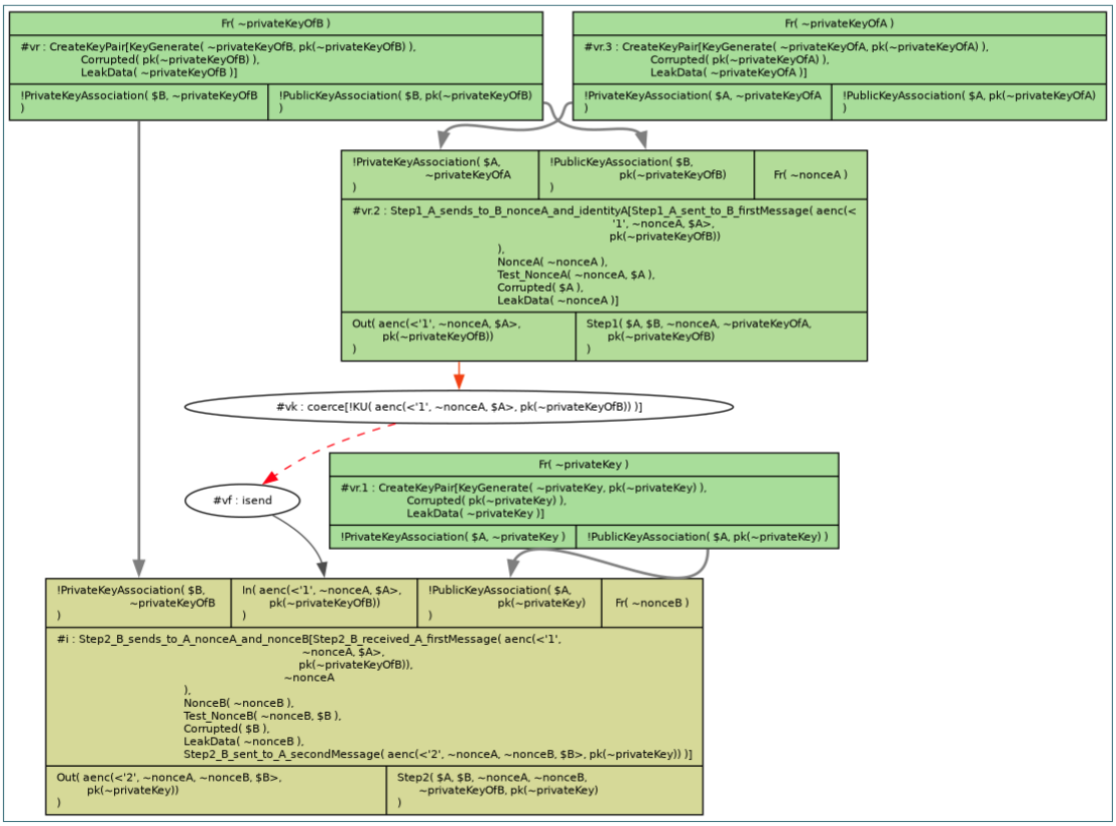

Figura 4.7: Inizializzazione ridondante delle entità

Tra i problemi più rilevanti e critici riscontrati durante l'esecuzione del protocollo è stato quello dell'utilizzo della chiave privata di A da parte di B, oppure la creazione di nuove chiavi.

La Figura 4.7 è un esempio di quanto detto, infatti essa rappresenta uno scambio di messaggi nel protocollo di Needham-Schroeder: notiamo le 2 rule in alto che creano le chiavi pubbliche e private degli attori e altre 2 rule più grandi che rappresentano gli step di scambio dei messaggi. Tra queste compare un'altra rule che ricrea le chiavi pubbliche e private già precedentemente create, capiamo subito che questo non è ammissibile in quanto in un protocollo di sicurezza, a meno che non sia un passaggio del protocollo stesso, un attore non ha motivo di cambiare le chiavi, soprattuto perchè non avrebbe più modo di aprire i messaggi.

Questo succede in quanto Tamarin Prover, per soddisfare le propiertà del lemma in esame e le rule descritte, va a creare una seconda volta le chiavi del secondo attore, infatti non ci sono rule o dichiarazioni nel lemma che ne impediscano questo tipo di comportamento.

Queste tipologie di condotte sono inaccettabili per un protocollo, ed è da queste osservazioni e scoperte del linguaggio Tamarin e del tool Tamarin Prover che è stato necessario e ovvio l'inizio dello studio più approfondito nella formulazione e creazione degli attori all'interno di un protocollo di sicurezza realizzato con questi strumenti.

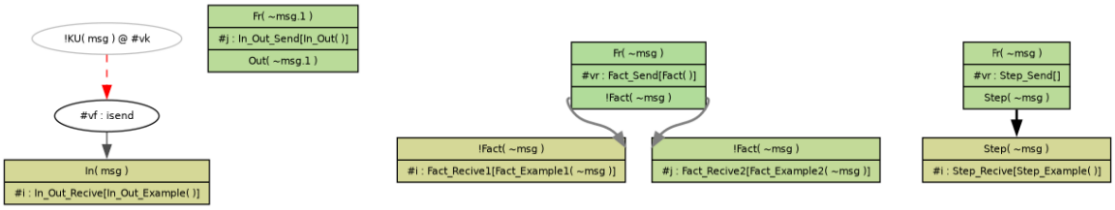

Figura 4.8: Canali Comunicazione

Canali di Comunicazione

Nel manuale di Tamarin vengono presentati 3 differenti tipologie di fatti per la comunicazione tra le rule, per comodità chiameremo quest'ultimi canali, la Figura 4.8 è un esempio di questi canali offerti dal mauale.

Ognuno di questi ha caratteristiche e specifiche diverse, noi ci focalizzeremo sul comportamento della prima immagine nella quale vengono implementati i fatti di `In(..)` e `Out(..)`. Dalle rule in esame, notiamo infatti che queste non sono direttamente collegate e vi è una seconda rule `!KU(msg)`, dalla quale parte con una freccia rossa tratteggiata; quest'ultima è molto importante inquanto rappresenta il comportamento e le azioni che un attaccante compie durante le analisi dei protocolli.

Sfortunatamente questa rappresentazione è l'unica che Tamarin Prover può offrirci, rendendo molto più complesso lo studio delle analisi e considerazioni quando si utilizzano le altre tipologie di canali, in quanto non prevedono l'esistenza di un attaccante durante le analisi.

Nel protocollo di Needham-Schroeder avviene una comunicazione tra gli attori del protocollo con `In(..)` e `Out(..)` poichè si utilizza espressamente la crittografia a chiave asimmetrica, ovvero l'utilizzo delle chiavi pubbliche e private che permettono di criptare i messaggi scambiati e quindi l'impedimento da parte dell'attaccante di scomporre i messaggi. Questa condizione cade nel momento in cui un nuovo modello di protocollo crittografico non si orienta sulla crittografia a chiave asimmetrica, concentrandosi su altri aspetti e peculiarità.

Il problema risiede proprio nei canali standard di Tamarin, che non permettono di definire e cogliere alcuni aspetti nella progettazione e definizione degli attacchi informatici.

Questo scenario trova già soluzione con ASLan++, ovvero vengono già presentati differenti tipologie di canali di comunicazione tra gli attori di un protocollo. Questo aspetto è stato già presentato nel capitolo di ASLan++ nei protocolli illustrati in [17] e[18].

In conclusione bisogna approfondire le tecniche di comunicazione tra gli attori del protocollo, implementando nuovi costrutti e soluzioni per i canali che permettano una definizione più precisa anche in Tamarin.

Figura 4.9: Cyber Kill Chain

Modello dell'Attaccante

In Tamarin, come già presentato, solo nella prima tipologia dei fatti `In(..)` e `Out(..)` viene considerata l'esistenza di un attaccante. Questo pone ancora un enorme problema nella fase di verifica e analisi del protocollo, infatti non si ha una precisa rappresentazione dei comportamenti malevoli.

Ogni attacco ha un obbiettivo diverso, ma segue un filo logico comune, questa caratteristica è stata analizzata e formalizzata dando una struttura precisa: questa catena di eventi prende il nome di *Cyber Kill Chain* [10]. L'analisi di questa ricerca punta a presentare e capire come un avversario, per raggiungere il suo obiettivo, debba riuscire a progredire attraverso tutta la catena, mettendo bene in evidenza quali azioni di mitigazione sono efficaci per evitare un attacco; la Figura 4.9, presa da [2], mette in luce le differenti fasi che un attacco compie.

Sappiamo, però, che ogni attacco ha un suo comportamento e caratteristiche peculiari. Quindi, per iniziare lo studio degli attacchi informatici e della loro rappresentazione con Tamarin, è stato preso come esempio "*MIM attack*", ovvero un attacco "*man in the middle*", nella quale un attore malevolo si posiziona in una conversazione tra due attori, un esempio classico è tra un utente e browser, dove lo scopo del attacante è origliare la conversazione o addirittura impersonare uno dei due attori del protocollo, facendo sembrare che vi sia un normale scambio di informazioni.

Capitolo 5

Analisi protocollo basato su Password

In questo capitolo, passeremo ad analizzare un altro protocollo di gestione dell'identità digitale, ossia un protocollo d' autenticazione basato su password. Come già introdotto, la vita quotidiana continua ad assistere ad un notevole progresso, con tecnologie sempre più raffinate che richiedono svariate operazioni sui propri dispositivi, offrendo comodità e velocità. Tuttavia, questa evoluzione ha riguardato anche gli attacchi informatici, che mettono ogni giorno a rischio la sicurezza e la privacy dei cittadini.

Ciò richiede la necessità di autenticare gli utenti in modo corretto e sicuro tramite protocolli di autenticazione. Questi rappresentano il punto di ingresso ai servizi online, quindi devono essere progettati in modo robusto per consentire solo utenti autorizzati ad accedere ai dati sottostanti.

Prima di poter modellare protocolli complessi e sofisticati abbiamo quindi bisogno di nuovi strumenti per poter rappresentare e formalizzare ogni circostanza sia dei protocolli che degli attaccanti, cosichè da esaminare ogni sfaccettatura del problema e scenario di definizione. Per affrontare questa sfida si presenta quindi un classico protocollo di autenticazione password-based, in quanto esso racchiude l'idea iniziale di ogni protocollo di autenticazione, ovvero un utente si registra e accede ad un servizio tramite id e password.

Il contributo innovativo di questa autenticazione basata su password stà nella creazione di nuovi modelli di sicurezza e rappresentazione degli attacchi che a volte vengono solo abbozzati o addirittura tralasciati. Andremo quindi a implementare le nuove soluzioni e tecniche, che puntano a definire le proprietà descritte in ASLan++ e non ancora presenti in Tamarin. Bisogna, però, affrontare un'ulteriore sfida, ovvero quella di mantenere l'idea e struttura originale di Tamarin, infatti vi è il rischio di produrre modelli che si discostano dal naturale funzionamento del tool Tamarin Prover, con il rischio di produrre analisi e risultati errati.

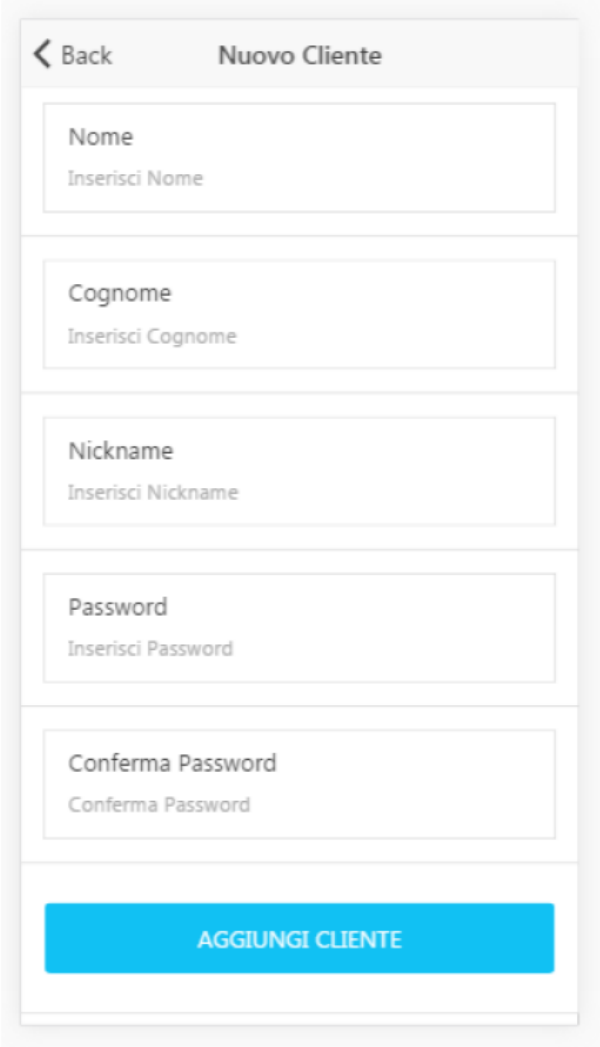

Figura 5.1: Nuovo Cliente

La Figura 5.1 rappresenta un classico esempio di applicazione nella quale un utente si registra e accede ad un servizio basato su password.

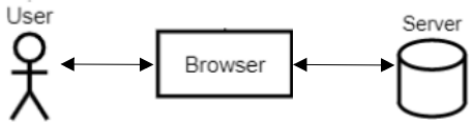

Figura 5.2: Attori Password-Based

5.1 Descrizione del protocollo

Prima di presentare l'apporto innovativo della ricerca presentiamo la struttura e gli attori che formano il protocollo basato su password.

Lo scenario è quindi quello di un classico protocollo di login, nella quale un utente vuole, con l'apporto di un dispositivo, collegarsi ad un servizio tramite le credenziali precedentemente create e salvate in un server. Il contesto di specifica di questo protocollo vedrà quindi 3 attori distinti, come mostarato in figura 2.1:

(i) **User** - utente che vuole registrarsi e accedere ad un servizio

(ii) **Browser** - sito internet che riceve le informazioni del utente, quali codice identificativo e password, che verranno inviate al server

(iii) **Server** - database che gestisce le informazioni del utente ricevute dal browser.

Notiamo che, rispetto a NSPK, sono presenti molte più specifiche e sfide che dobbiamo ora specificare nel dettaglio, infatti la modellazione delle operazioni e azioni degli attori saranno il fulcro della procedura di analisi.

1. L'utente, dopo aver creato una password, invierà al browser la password e il suo codice utente, specificando che vuole registrarsi. Il browser, quindi, prenderà in carico la richiesta del utente e la invierà al server che registrerà le informazioni.

2. L'utente, dopo essersi registrato, utilizzerà la stessa password e il suo codice identificativo per accedere, inviando un messaggio al browser, che prenderà in carico la richiesta dell'utente e la invierà al server.

3. Il server avrà quindi la password e codice identificativo del punto 1 e la password e codice del punto 2, se le password coincideranno e anche i codici allora l'autenticazione sarà avvenuta con successo.

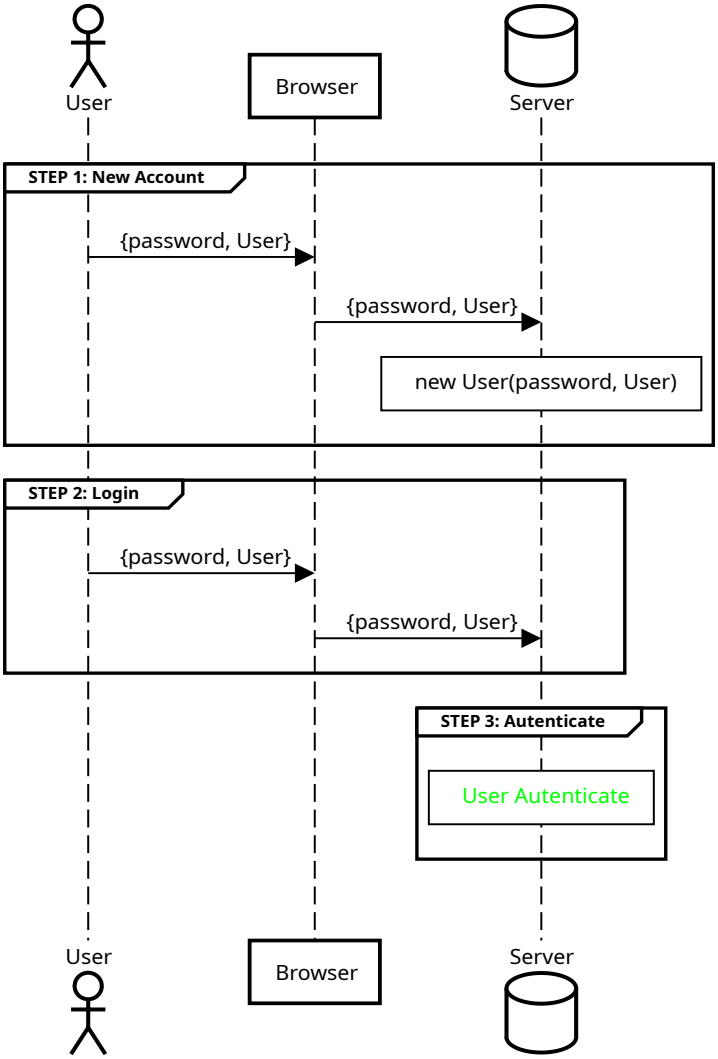

Figura 5.3: Protocollo basato su Password

La figura 5.3 mostra più nel dettaglio la struttura del protocollo che riassume gli studi e osservazioni condotti durante il tirocinio in FBK.

5.2 Modellazione in Tamarin

Rispetto al protocollo di NSPK, questo nuovo protocollo di Login i messaggi, scambiati tra gli attori, sono più complessi e devono essere illustrati, in modo tale che, durante l'esecuzione del protocollo, sia chiaro chi è il mittente e destinatario dei messaggi. Saranno quindi presentate le azioni che gli attori compiranno nel protocollo, i codici presentati non funzionanti e hanno solo lo scopo illustrativo dello scambio dei messaggi tra gli attori.

Messaggi User-Browser

Qui di seguito lo scambio di messaggi che l'Utente invia al Browser, come presentati nella sezione precedente: Codice 5.1.

```
// -------------------- User create password --------------------
rule User_Create_Password:
  let
    // Message
    msg_password = <'msg_password', ~password, $User>
    msg_log = <'msg_log', ~password, $User, 'User'>
    url = <'HTTP', $Browser>
  in
  [ Fr(~password) ]-->
  [
    // User send Browser for point 1-2
    User_Send(user_id, url, msg_log),
    User_Send(user_id, url, msg_password)
  ]

// ----------------------- Browser point 1 ----------------------
rule User_InsertOn_Browser:
  let
    // Same message of User
    msg_password = <'msg_password', password, $User>
    url = <'HTTP', $Browser>
  in
  [ Browser_Receive(channel_id, url, msg_password) ]
  -->[ .. ]

// ----------------------- Browser point 2 ----------------------
rule User_LoginOn_Browser:
  let
    msg_log = <'msg_log', password, $User, actor>
    url = <'HTTP', $Browser>
  in
  [ Browser_Receive(channel_id, url, msg_log) ]
  -->[ .. ]
```

Codice 5.1: Messaggi User-Browser

Messaggi Browser-Server

Qui di seguito lo scambio di messaggi tra Browser e Server, inoltre l'ultimo punto del protocollo è quello del server, che ricevendo dal browser i messaggi di creazione e accesso all'account effettuerà il controllo del contenuto per confermare l'identità dell'utente e quindi l'autenticazione: Codice 5.2.

```
// ------------------------- Point 1 -------------------------
rule User_InsertOn_Browser:
  let
    msg_password = <'msg_password', password, $User>
    url = <'HTTP', $Browser>
    url2 = <'HTTPS', $Server>
    msg_server_user_pass = <'msg_server_user_pass', password, $User, $Browser>
  in
  [ Browser_Receive(channel_id, url, msg_password) ]
  -->[ Browser_Send(browser_id, url2, msg_server_user_pass) ]
// ------------------------- Point 2 -------------------------
rule Browser_Send_Server:
  let
    msg_server_user_pass = <'msg_server_user_pass', password, $User, $Browser>
    msg_user_register = <'msg_user_register', password, $User, $Browser>
    url = <'HTTPS', $Server>
  in
  [ Server_Receive(channel_id, url, msg_server_user_pass) ]
  -->[ TLS_Server(msg_user_register) ]
// ------------------- Server Check point 3 -------------------
rule Control_Password:
  let
    // Message from Browser of point 1-2
    msg_login = <'msg_login', password, $User, actor>
    msg_user_register = <'msg_user_register', password, $User, $Browser>
    url = <'HTTPS', $Server>
  in
  [ Server_Receive(channel_id, url, msg_login),
    TLS_Server(msg_user_register)
  ]--[
    // Check Values
    Actor(actor_login, 'User'),
    ValueProof(password_user, password_login),
    Completed_Protocol(password_user, password_login , $User),
  ]->[ Step_Finish_Protocol($User, $Browser, $Server) ]
```

Codice 5.2: Messaggi Browser-Server

Canali

Abbiamo parlato della costruzione e forma dei messaggi scambiati tra gli attori del protocollo, tralasciando però i canali di comunicazione attraverso i quali essi transitano, qui di seguito le 4 tipologie di canali disponibili in questo nuovo protocollo.

1. **Broadcast**

   Utilizziamo i costrutti `In(..)` e `Out(..)`, ovvero una sottoclasse dei fatti presenti in Tamarin. L'idea è quella di rappresentare un messaggio *plain-text* sulla rete, ovvero senza nessuna protezione e leggibile da tutti gli attori del protocollo, compresi anche gli attaccanti.

2. **Confidential**

   Messaggi scambiati tra User e Browser: utilizzeremo nel protocollo 2 canali che vogliono rappresentare le azioni dell'Utente di registrazione e login. Queste azioni sono l'inserimento da tastiera, l'idea è quindi anche quella di un canale che rappresenta anche i possibili attacchi legati alla appropriazione non legittima dei dati dell'utente.

3. **Bilateral**

   Messaggi scambiati tra Browser e Server: sarà un singolo canale nella quale il browser si connette al server per inoltrare le informazioni del utente, qui si possono specificare i vari protocolli di sicurezza e racchiudere attacchi più complessi.

4. **TLS**

   Supponendo che TLS sia un protocollo dimostrabilmente sicuro e non violabile, possiamo trattarlo diversamente, in quanto non vi ci sono possibili attacchi che possano violare il protocollo. La soluzione è quindi quella si considerare lo scambio tra 2 attori, o il medesimo come nel nostro protocollo con il Server, come uno scambio tramite i fatti speciali di Tamarin chiamati Step. In altre parole lo si può vedere come un "passaggio di stato" delle variabili tra due rule.

La creazione e struttura dei canali sarà ripresa nei prossimi capitoli.

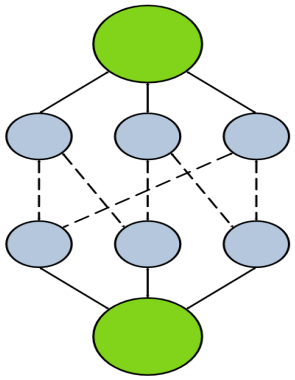

Figura 5.4: Super Nodi

5.3 Nuove Implementazioni

Vogliamo ora riprendere le lezioni apprese grazie a NSPK e fornire una soluzione, infatti ora saranno presentati gli apporti innovativi di questa ricerca che saranno successivamente implementati nel protocollo basato su password.

5.3.1 Incapsulamento del protocollo

Come precedentemente presentato, uno dei più problemi più rilevanti e critici riscontrati durante l'esecuzione del protocollo di NSPK è stato quello dell'utilizzo della chiave privata di A da parte di B, oppure la creazione di nuove chiavi. Una soluzione parziale al problema del numero di attori è la composizione di Restiction specifiche che vadano ad evitare queste circostanze.

Questa risoluzione del problema però, come già presentato e illustrato nel capitolo del backgound, potrebbe andare a "mascherare" o addirittura togliere specifiche tracce d'esecuzione del protocollo in esame, e questo è ancora peggio in quanto questi cammini potrebbero contenere un attacco al protocollo, rendendo la definizione del protocollo non attendibile e le analisi incomplete.

Per ovviare a questo ulteriore problema vienne proposta una nuova implementazione del protocollo, che definiamo come "incapsulamento del protocollo".

Questo approccio prende spunto da una tecnica algoritmica implementata per risolvere i problemi di *Flusso Massimo* nei grafi, ovvero l'utilizzo di una *super-sorgente* e di un *super-pozzo*, ovvero una tecnica nella quale vi è un nodo principale nella quale tutti i percorsi degli altri nodi coincidono, come mostrato in Figura 5.4.

Per spiegare l'efficacia di questa tecnica bisogna pensare a come Tamarin Prover analizza il protocollo, infatti quest'ultimo, per verificare le proprietà del lemma, parte dalla fine e risolve il protocollo con le rule precedentemente predisposte. Essendo a conoscenza di questo approccio ci basterà vincolare i fatti in ingresso e d'uscita, forzando il tool a seguire una traccia precisa, senza però togliere possibili tracce durante l'esecuzione.

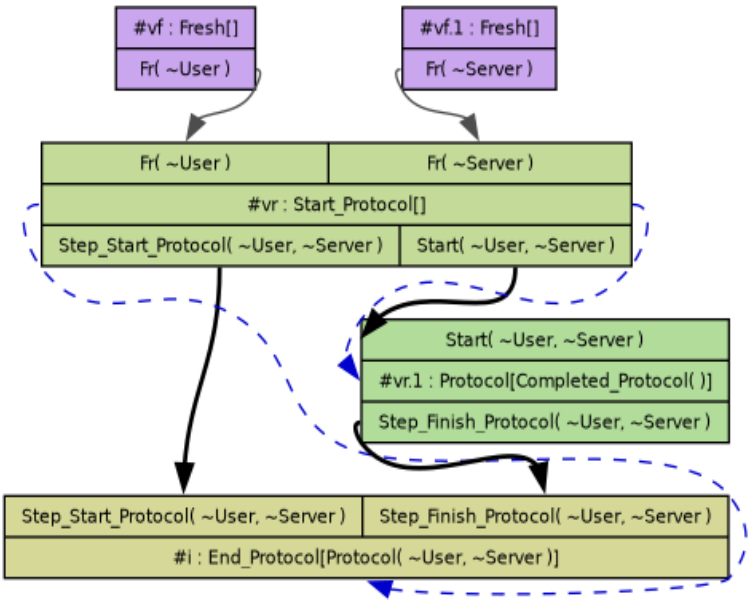

Figura 5.5: Incapsulamento Protocollo

Questo concetto è più chiaro osservando la Figura 5.5, dove notiamo 3 diverse regole:

(i) `Start_Protocol`: inizializzazione degli attori nel protocollo

(ii) `Protocol`: definizione del protocollo

(iii) `End_Protocol`: conclusione del protocollo

Andremo così a definire una proprietà *Protocol(User, Server)* in `End_Protocol`. Questa, per poter essere verificata da Tamarin Prover, avrà bisogno della regola `Start_Protocol` e `Protocol`, infatti quest'ultime contengono i fatti `Step_Start_Protocol` e `Step_Finish_Protocol`, necessari a `End_Protocol` per essere verificato. L'idea è quindi quella di porre nella prima e ultima rule gli stessi attori, cosicché al completamento della analisi del protocollo quest'ultimo dovrà contenere gli stessi attori.

Osserviamo però un legame vincolante tra le regole `Start_Protocol` e `Protocol`, infatti `Protocol`, per essere derivato, ha bisogno del fatto `Start` contenuto in `Start_Protocol`. Questo ci permette di incapsulare `Protocol`, infatti ricordiamo che la proprietà *Protocol(User, Server)* in `End_Protocol`, per essere verificata e derivata ha bisogno di `Step_Start_Protocol` e questa forza la derivazione indiretta del fatto `Start` e quindi la derivazione della regola `Protocol`. Se i fatti di `Start_Protocol` non vengono soddisfatti, la proprietà *Protocol(User, Server)* non sarà verificata, in quanto mancano dei fatti per concludere la verifica della proprietà del lemma. Questo ci permette inoltre di avere in tutte le regole gli stessi attori durante la validazione della proprietà.

Questo pone un grande vantaggio, infatti non siamo obbligati a utilizzare le Restiction per far funzionare il protocollo, e quindi mantenere tutte le possibili tracce e percorsi che il tool Tamarin Prover andrà ad analizzare, compresi tutti i possibili comportamenti di un attaccante.

Il vantaggio di questo approccio è che possiamo commentare queste regole speciali, ed attivarne altre più semplici, per ottenere delle analisi che includano anche i comportamenti critici presentati. Il Codice 5.3 compone una parte della Figura 5.5.

```
// --------------------------- START ----------------------------
rule Start_Protocol:
  [ Fr(~User), Fr(~Server) ]
  -->[ Step_Start_Protocol(~User, ~Server),
    Start(~User, ~Server) ]

// --------------------------- PROTOCOL -------------------------
rule Protocol:
  [ Start(User, Server)]
  --[ Completed_Protocol() ]->
  [ Step_Finish_Protocol(User, Server) ]

// ---------------------------- END -----------------------------
rule End_Protocol:
  [ Step_Start_Protocol(User, Server),
    Step_Finish_Protocol(User, Server),
  ]--[ Protocol(User, Server) ]->[]

// --------------------------- LEMMA ----------------------------
lemma Protocol_Complete:
    exists-trace
    "Ex User Server #i. Protocol(User, Server)@i "
```

Codice 5.3: Codice per l'incapsulamento del protocollo

5.3.2 Canali di Comunicazione

Nel manuale di Tamarin vengono presentati 3 differenti tipologie di comunicazione tra le rule. Queste, però, non permettono di definire e cogliere alcuni aspetti nella progettazione e definizione degli attacchi informatici, cosa che in ASLan++ trova già una soluzione. Bisogna quindi approfondire le tecniche di comunicazione tra gli attori del protocollo, implementando nuovi costrutti e soluzioni per i canali che permettano una definizione più precisa anche in Tamarin, mantenendo però lo stile di programmazione e definizione per non produrre modellazioni incoerenti.

Un lavoro molto interessante, che ha permesso di intraprendere uno studio più dettagliato per i canali di communicazione, è stato la tesi magistrale [9]: "Modelling and Analysis of Web Applications in Tamarin", sempre disponibile nella pagina dedicata di Tamarin del ETH di Zurigo.

In questa tesi viene presentato un modello generico per la modellazione degli elementi chiave di un'infrastruttura Web, il risultato è quindi un framework generico che può essere utilizzato anche per realizzazione applicazioni Web e protocolli concreti in Tamarin. Viene quindi presentata una nuova forma nella modellazione di un protocollo, nella quale avviene un'assegnazione di un codice identificativo per ogni canale alla quale vengono dedicate delle rule specifiche.

L'idea è quindi quella di estrapolare e scoprire le caratteristiche base di questa nuova struttura per andare a ridefinire le differenti tipologie di canali, e relative proprietà, già implementate e disponibili in ASLan++, in modo tale d'avere analisi e considerazioni più precise sullo studio degli attacchi informatici.

Supponiamo che un utente voglia comunicare con un browser: l'utente non utilizzerà direttamente uno dei canali standard di Tamarin, ma utilizzerà un canale dedicato contrasegnato con il suo codice identificativo.

Tralasciando la costruzione degli attori, qui di seguito viene presentato un esempio di struttura di un canale che un utente sfrutta per interfacciarsi con un browser; avremo quindi User e Browser che non comunicano direttamente, ma sfruttano un canale dedicato e identificato da un codice. Codice 5.4.

```
// -------------------------- SETUP ----------------------------
rule Setup_User_Browser_Channel:
  [ !User(user_id)
  , Fr(~channel_user) ]
  --[ Setup_User_Channel(~channel_user) ]->
    [ !Channel_U_to_B($Browser, user_id, ~channel_user) ]

// -------------------------- CHANNEL ----------------------------
rule User_To_Browser_Https:
let url = <'HTTPS', $Browser>
in
  [ User_Send(user_id, url, message)
  , !Channel_U_to_B($Browser, user_id, channel_id) ]
  -->
  [ Browser_Receive(channel_id, url, message) ]
```

Codice 5.4: Setup Attori

Il codice qui riportato mostra un User che, dopo aver creato la sua password e il proprio canale, si collega al Browser con le proprie credenziali: Codice 5.5.

```
// ---------------------------- ACTOR -----------------------------
rule CreateUser:
  [ Fr(~user_id) ]
  --[ New_User(~user_id) ]->
  [ !User(~user_id) ]

// ---------------------------- SETUP -----------------------------
rule Setup_User_Browser_Channel:
    [ !User(user_id)
    , Fr(~channel_user) ]
    --[ Setup_User_Channel(~channel_user) ]->
    [ !Channel_U_to_B($Browser, user_id, ~channel_user) ]

// ---------------------------- CHANNEL ---------------------------
rule User_To_Browser_Https:
  let
    url = <'HTTPS', $Browser>
  in
    [ User_Send(user_id, url, message)
    , !Channel_U_to_B($Browser, user_id, channel_id) ]
    -->
    [ Browser_Receive(channel_id, url, message) ]

// ---------------------------- PROTOCOL --------------------------
rule User_Create_Password:
  let
    message = <'msg_log', ~password, $User>
  in
  [!User(user_id),
    Fr(~password),
  ]-->[ User_Send(user_id, url, message) ]

rule User_LoginOn_Browser:
  let
    message = <'msg_log', password, $User>
    url = <'HTTPS', $Browser>
  in
  [
    Browser_Receive(channel_id, url, message),
  ]--[ Login() ]->[]

// ---------------------------- LEMMA -----------------------------
lemma Channel_Example:
  exists-trace
    "Ex #i. Login()@i "
```

Codice 5.5: Esempio Canale

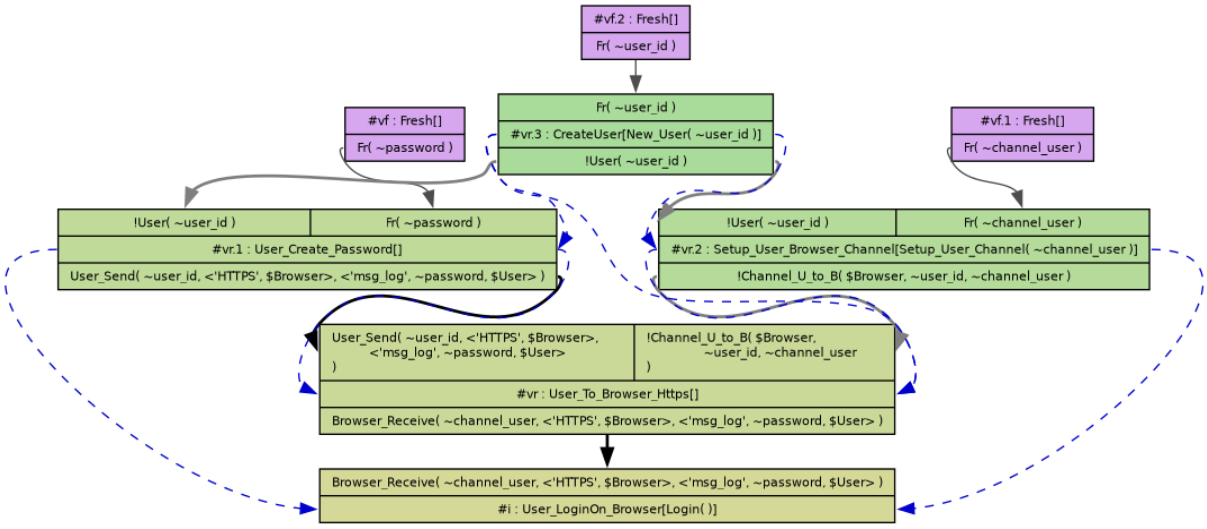

Figura 5.6: Esempio Canale

La Figura 5.6 mostra un "istanza" grafica del protocollo descritto nella pagina precedente, infatti i nomi e valori possono assumere forme diverse in fase di validazione.

Come spiegato in precedenza, il tool Tamarin Prover, per verificare e derivare la proprietà di un lemma, parte dalla rule in cui è contenuta la proprità che solitamente viene posta nell'ultima, per poi derivare tutte quelle richieste, avremo quindi il seguente schema risolutivo del tool:

1. `User_LoginOn_Browser`
   - Proprietà del lemma da verificare: *Login()*
   - Richiede: `Browser_Receive(channel_user, <'HTTPS', $Browser>, <'msg_log', password, $User>`
2. `User_To_Browser_Https`
   - Soddisfa: `Browser_Receive(channel_user, <'HTTPS', $Browser>, <'msg_log', password, $User>`
   - Richiede: `User_Send(user_id, url, message) - !Channel_U_to_B( $Browser , user_id , channel_id)`
3. `User_Create_Password - Setup_User_Browser_Channel`
   - Soddisfano: `User_Send(user_id, url, message) - !Channel_U_to_B($Browser, user_id, channel_id)`
   - Richiedono: `!User(user_id)` e valori `Fr(password)` e `Fr(channel_user)`
4. `CreateUser`
   - Soddisfa: `!User(user_id)`
   - Richiede: `Fr(user_id)` ... Completamento Derivazione

Questo tipo di costruzione dei canali ci permette di valutare e considerare un nuovo tipo di modellazione, infatti, un attore ha la possibilità di utilizzare uno o più canali per comunicare con un altro attore o diversi grazie al utilizzo della seconda struttura dei fatti disponibili di default in Tamarin, marchiati con `!Fact(..)`. Quando un attore crea un canale di comunicazione viene specificato anche il destinatario alla quale arriveranno i messaggi con `url = <'HTTPS', $Browser>`, con questa dicitura specifichiamo il destinatario "Browser" e anche il tipo di protocollo "HTTPS", nulla toglie che possiamo aggiungere altri flag o specifiche che andremo a sfruttare per Restiction o proprietà dei lemmi in fase di validazione dei lemmi.

Sorge quindi la domanda su come Tamarin Prover capisce che deve usare un determinato canale per un messaggio per continuare la validazione di un lemma, banalmente questo è già stato detto nella rule `User_To_Browser_Https`, infatti il tool, per valutare questa rule, ha bisogno di:

- Input: `User_Send(user_id, url, message)` - `!Channel_U_to_B($Browser, user_id, channel_id)`
- Output: `Browser_Receive(channel_id, url, message)`

Questi fatti utilizzano la stessa struttura di "message" e "url" che viene specificato nella sezione "`let .. in`" delle rule `User_LoginOn_Browser` e `User_Create_Password`: Codice 5.6.

```
1 rule User_LoginOn_Browser:
2   // Section for values
3   let
4     message = <'msg_log', password, $User>
5     url = <'HTTPS', $Browser>
6   in
7     ...
```

Codice 5.6: Codice per il login dell'utente tramite browser

Abbiamo quindi presentato le basi per la formalizzazione e verifica di un nuovo modello dei canali. Questo ci permette di definire più nel dettaglio caratteristiche e proprietà dei canali e scambio di informazioni degli attori in un protocollo. Sfortunatamente questo modello non è ancora pronto per la definizione dei canali come per ASLan++.

5.3.3 Attaccante

In Tamarin, come già presentato, solo nella prima tipologia dei fatti `In(..)` e `Out(..)` viene considerata l'esistenza di un attaccante. Questo pone ancora un enorme problema nella fase di verifica e analisi del protocollo, infatti non si ha una precisa rappresentazione dei comportamenti malevoli.

Precedentemente abbiamo visto una nuova formulazione di comunicazione tra gli attori di un protocollo dove si utilizzano canali specifici, per poter permettere all'attacante di intervenire e interagire con il protocollo dobbiamo, quindi, definire anche per l'attaccante un nuovo modello, in modo tale che anche quest'ultimo possa impersonalizzarsi nel protocollo.

Una soluzione interessante, per lo studio di un attaccante, è rappresentare quest'ultimo con specifiche rule, fornendogli quindi degli strumenti necessari per poter interagire con il protocollo. Avremo quindi la possibilità di raffigurare tutti i possibili comportamenti malevoli che può intraprendere.

L'attaccante non sarà quindi solo rappresentato come una singola freccia rossa trattegiata, ma assumerà la forma di un vero e proprio attore nel protocollo, con formalizzazioni e stutture simili per User e Browser, in modo tale da ottenere rappresentazioni grafiche e propietà dei lemmi più precisi che ne rappresentino il comportamento.

Qui di seguito la costruzione e rappresentazione del canale che un utente malevolo utilizzerà nel protocollo: Codice 5.7.

```
1 // ---------------------- Attacker Channel ----------------------
2
3 rule Setup_Attacker_User_Channel:
4   [ Fr(~channel_attack) ]
5   --[ Network_Attacker(),
6       Setup_Attacker_Channel(~channel_attack)  ]->
7     [ !Malicious_Channel($User, ~channel_attack) ]
```

Codice 5.7: Codice per l'inizializzazione del canale dell'attaccante

L'attaccante avrà quindi a disposizione gli strumenti necessari per interagire con il protocollo e le analisi, se in presenza di un atttacco in corso, evidenzieranno le tracce che compongono le azioni e comportamenti che un utente malevolo compie nel protocollo.

Proprietà Attaccante

Finora sono stati presentati lemmi e proprietà che valutano un normale funzionamento di un protocollo, studiandone specifiche proprietà. Vogliamo ora presentare dei nuovi lemmi specifici per gli attacchi, infatti la modellazione e rappresentazione di quest'ultimi risulta essere più facile grazie alle nuove rule che ne rappresentano singolarmente i comportamenti.

Anche se i comportamenti malevoli sono differenti tra loro, proponiamo 3 proprietà comuni che ne analizzano il comportamento. Ognuna di esse analizza un aspetto differente dello stesso attacco. Tamarin Prover, inoltre, ne una traccia grafica.

1. Nel primo lemma si vuole rappresentare la normale traccia d'esecuzione del protocollo con anche l'inserimento dell'attacco. Ciò può essere formulato grazie alla proprietà `Attack(Attacker)@i`, che deve essere specificata in una delle rule che compongono l'attacco. Permettendo così a Tamarin Prover di rappresentare graficamente l'inserimeto dell'attaccante nella normale esecuzione: Codice 5.8.

```
1 lemma Ex_Traces_Attack:
2 exists-trace
3     "Ex actors #c. Protocol(actors)@c &
4     // Proprietà ...
5     & ( (Ex Attacker #i. Attack(Attacker) @i) )"
```

Codice 5.8: Lemma per la verifica della presenza di attaccanti

2. Il secondo lemma, invece, va a modellare solo la singola traccia d'esecuzione dell'attaccante. Più precisamente in quale punto l'attacco compare e tutte le rule che lo compongono. Questo lemma, mostrato nel Codice 5.9, è più complesso del precedente, infatti è basato sulle seguenti considerazioni: nel caso in cui non esista un attaccante a livello di rete e le proprietà di sicurezza siano rispettate, allora non esisterà un attaccante in grado di compromettere il protocollo. Qualora il lemma sia *FALSO*, possiamo desumere che sia presente un attacco e ottenere la relativa traccia.

```
1 lemma Attacker:
2 "(
3     not (Ex #i. Network_Attacker() @i) &
4     (All actors #i #j. Protocol(actors)@i & Protocol(actors)
    @j ==> #i=#j)
5     // Proprietà ...
6 )
7 ==> not ( (Ex Attacker #i. Attack(Attacker) @i) ) "
```

Codice 5.9: Lemma per la modellazione della sola traccia dell'attaccante

3. Nel terzo lemma vengono specificate TUTTE le proprietà del protocollo che verranno valutate per TUTTE le tracce d'esecuzione del protocollo. Inoltre e non viene specificata l'esistenza del attaccante come precedentemente fatto. L'idea è quella di sfruttare le tecniche di derivazione di Tamarin Prover, quest'ultime, infatti, devono analizzare TUTTE le rule per verificare le proprietà del lemma. Questo ci porterà ad avere 2 possibili risultati:

   (i) *FALSO*: vengono mantenute tutte o solo parte delle rule che compongono l'attacco e quindi il protocollo viene violato

   (ii) *VERO*: vengono tolte tutte o solo parte delle rule che compongono l'attacco, queste, però, non sono sufficienti a violare il protocollo

   Questi risultati ci permettono di capire quali combinazioni delle rule degli attacchi violano il protocollo, infatti se il lemma risulta *FALSO* allora le proprietà del protocollo sono violate. Tamarin Prover ci fornisce anche le tracce nella quale le proprietà vengono violate dall'attacco: Codice 5.10.

```
1 lemma All_Traces_Protocol:
2     "(All actors #i #j. Protocol(actors)@i & Protocol(actors)
    @j ==> #i=#j)
3     // Proprietà ..."
```

Codice 5.10: Lemma per la verifica di tutte le proprietà del protocollo

5.4 Modellazione con Nuove Implementazioni in Tamarin

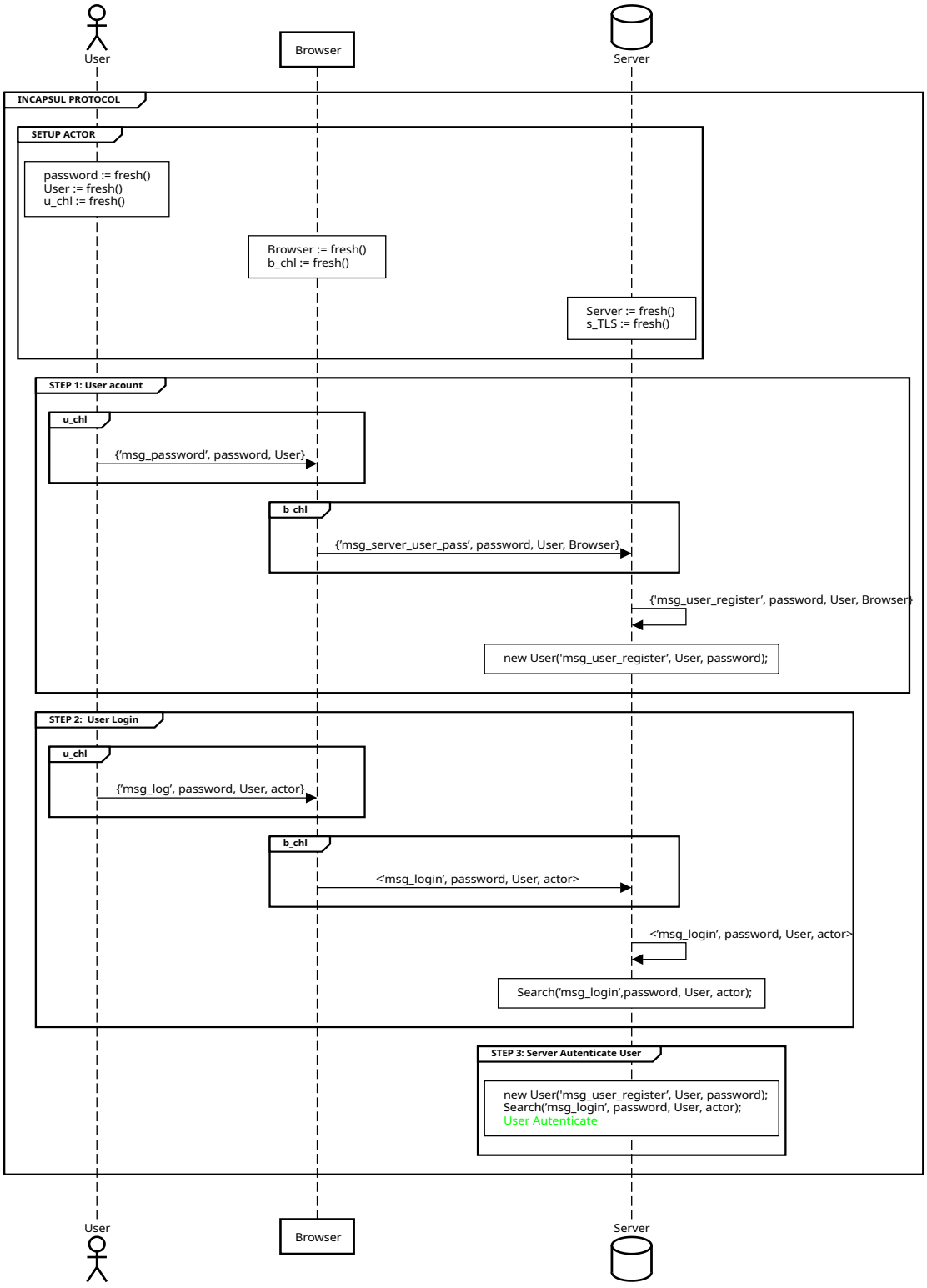

Figura 5.7: Protocollo basato su Password con Nuove Implementazioni

La Figura 5.7 rappresenta le specifiche del nuovo protocollo basato su password con l'implementazione delle tecniche innovative presentate.

Il protocollo, per poter essere esaminato e analizzato nella sua interezza deve essere così suddiviso:

(i) **Protocollo Base**

Istanza del protocollo che verifica i seguenti lemmi:

- una singola traccia d'esecuzione SICURA
- tutte le possibili tracce esistenti

(ii) **Protocollo con Attacco**

Istanza del protocollo base con le rule di un specifico attacco, dove si verificano i seguenti lemmi:

- una singola traccia d'esecuzione SICURA
- i tre lemmi specifici per l'attacco

Mantenendo una versione base del protocollo, in cui vengono verificate esclusivamente le tracce base del procollo e quindi sicure, e istanze specifiche di attacchi ci permette di studiare nel dettaglio specifici problemi e implementazioni.

Inoltre, per mantenere separato lo studio di ogni singolo attacco, bisogna mantenere il primo lemma della versione base del protocollo e far si che sia sempre valido, infatti in caso contrario sarebbe da considerare un errore d'implementazione, infatti il protocollo deve sempre avere almeno una traccia d'esecuzione verificabile.

Il vantaggio nel aver realizzato un modello degli attaccanti che prevede l'esistenza di rule specifiche dei suoi comportamenti rende possibile considerare una programmazione modulare, ovvero se io includo un comportamento ulteriore per un attacco questo comportamento potebbe far emergere traccie d'esecuzione nuove prima non considerate per l'attacco originale.

Nella pagina seguente viene presentata la Figura 5.8, essa raffigura la traccia base di completamento del protocollo con implementazione delle tecniche presentate precedentemente: Codice 5.15.

```
1 lemma Ex_Protocol_Complete:
2   exists-trace
3     // Esiste almeno una traccia di completamento del protocollo
4     "Ex User Browser Server #c. Protocol(User, Browser, Server)@c
     & (Ex User password #i. Completed_Protocol(password, password
    , User)@i & Ex #z. Password(password, User)@z)"
```

Codice 5.11: Lemma per la verifica della corretta esecuzione del protocollo

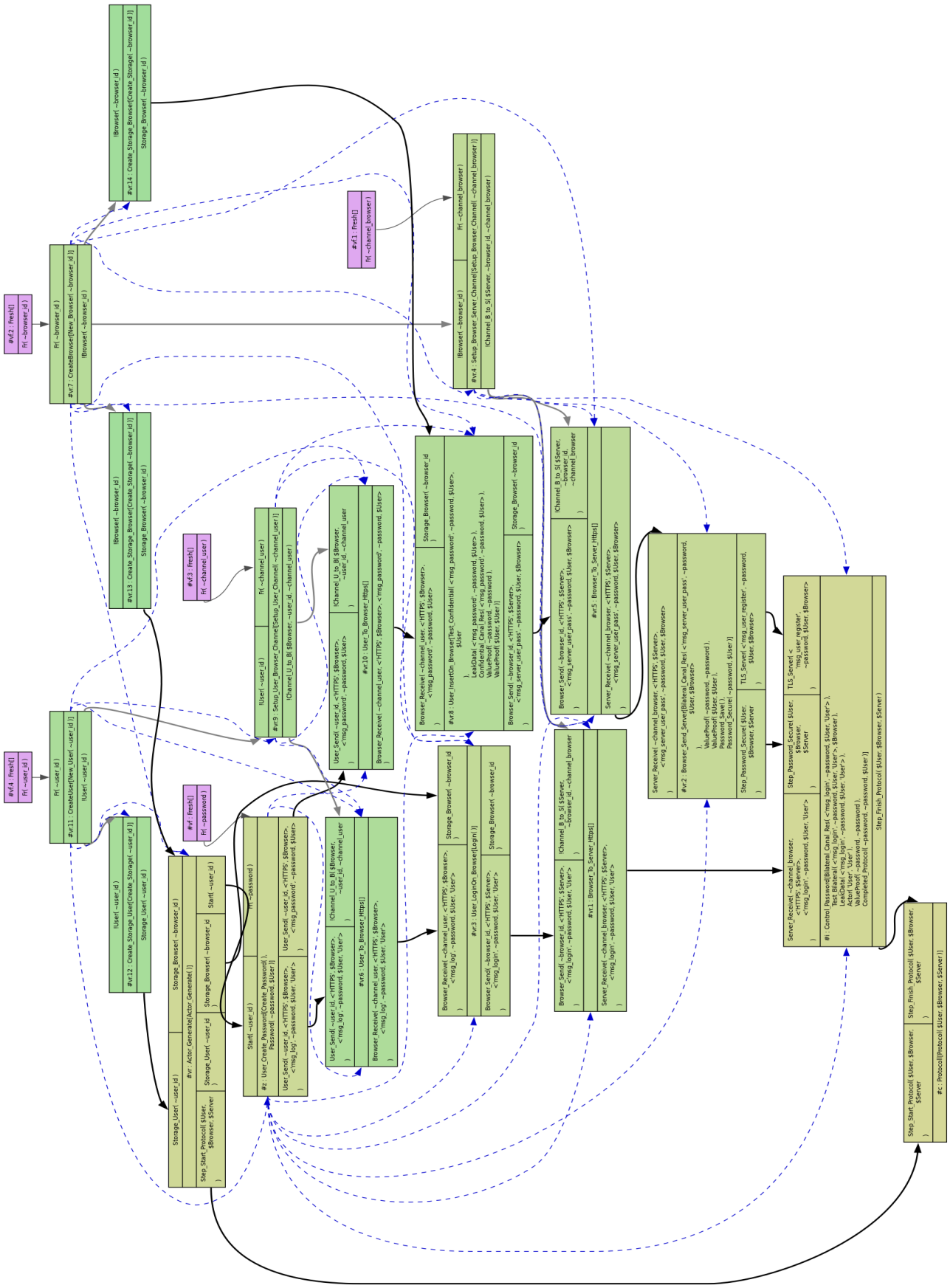

Figura 5.8: Protocollo basato su Password con Nuove Implementazioni in Tamarin

In questa figura non sono presenti frecce rosse, ovvero attaccanti, in quanto è l'esecuzione base del protocollo.

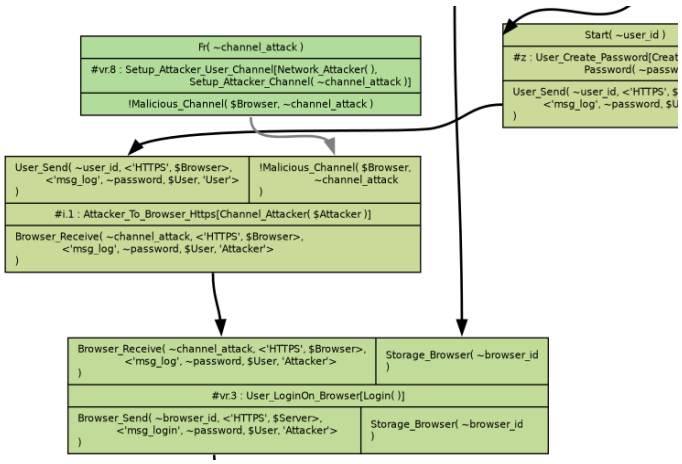

Figura 5.9: Attacco Base

5.5 Modellazione attacchi password-based in Tamarin

Come già anticipato questo metodo di autenticazione risulta essere il più comune e anche il più facile da violare per i criminali.

Questo ci porta ad uno dei problemi più interessanti e delicati di Tamarin, ovvero la corretta implementazione di un attaccante. Infatti, quest'ultimo, non è previsto nella compilazione di analisi del protocollo, deve essere adeguatamente implementato a seconda delle caratteristiche che lo compongano.

5.5.1 Implementazione Attaccante

Inoltre bisogna specificare in quale punto del protocollo esso si insinua, questo porta ad un ulteriore situazione molto delicata, infatti la stuttura del protocollo e le rule devono essere implementate per permettere di descrivere un possibile ingresso del attaccante nel protocollo.

ATTACCO BASE

Qui di seguito la rappresentazione di un attaccante che cattura un messaggio e lo spedisce al browser con le informazioni del utente, come mostrato in Figura 5.9 e dal Codice 5.12.

```
1  // ------------------------ BASE ATTACK -------------------------
2  rule Attacker_To_Browser_Https:
3  let url = <'HTTPS', $Browser>
4      message = <'msg_log', password, $User, actor>
5      message_ack =  <'msg_log', password, $User, 'Attacker'>
6  in
7      [ User_Send(user_id, url, message)
8      , !Malicious_Channel($Browser, channel_id) ]
9    --[ Channel_Attacker($Attacker) ]->
10     [ Browser_Receive(channel_id, url, message_ack) ]
```

Codice 5.12: Attacco base

ATTACCO CON APERTURA DEL MESSAGGIO

Qui di seguito, invece, la rappresentazione di un attaccante che cattura un messaggio e lo spedisce al browser con le informazioni del utente, ma inoltre è in grado di scomporre il messaggio originale, ricavando la password dell'utente che potrà sfruttare per altri attacchi più complessi: Codice 5.13.

```
// ---------------------- ATTACK with UNBOX --------------------

rule Attacker_Unbox_Message:
let url = <'HTTPS', $Browser>
  message = <'msg_log', password, $User, actor>
in
  [ User_Send(user_id, url, message) ]
  --[ Unbox_Attacker($Attacker) ]->
  [ Out(message) ]

rule Attacker_To_Browser_Https:
let url = <'HTTPS', $Browser>
    message = <'msg_log', password, $User, actor>
    message_ack =  <'msg_log', password, $User, 'Attacker'>
in
  [ In(message)
  , !Malicious_Channel($Browser, channel_id) ]
  --[ Channel_Attacker($Attacker) ]->
  [ Browser_Receive(channel_id, url, message_ack) ]
```

Codice 5.13: Attacco con apertura del messaggio

Notiamo che in questa implementazione sono presenti i costrutti `In(..)` e `Out(..)`, questi ci permettono di rappresentare la scomposizione dei messaggi e le operazioni che svolge un attaccante, infatti questi sono gli unici strumenti di Tamarin che rappresentano un attaccante.

Nell'immagine della pagina seguente vengono presentate le regole e derivazioni che Tamarin Prover produce per la definizione e modellazione del attacco: ATTACK with UNBOX.

Notiamo le freccie rosse continue e tratteggiate, queste rappresentano le operazioni che l'attaccante compie tra le rule di `In(..)` e `Out(..)`. Notiamo anche i costrutti `!KU`, quest'ultimi sono i terminali di Tamarin, grazie a questi possiamo vedere come l'attaccante compone il nuovo messaggio utilizzando la password del utente da mandare al browser.

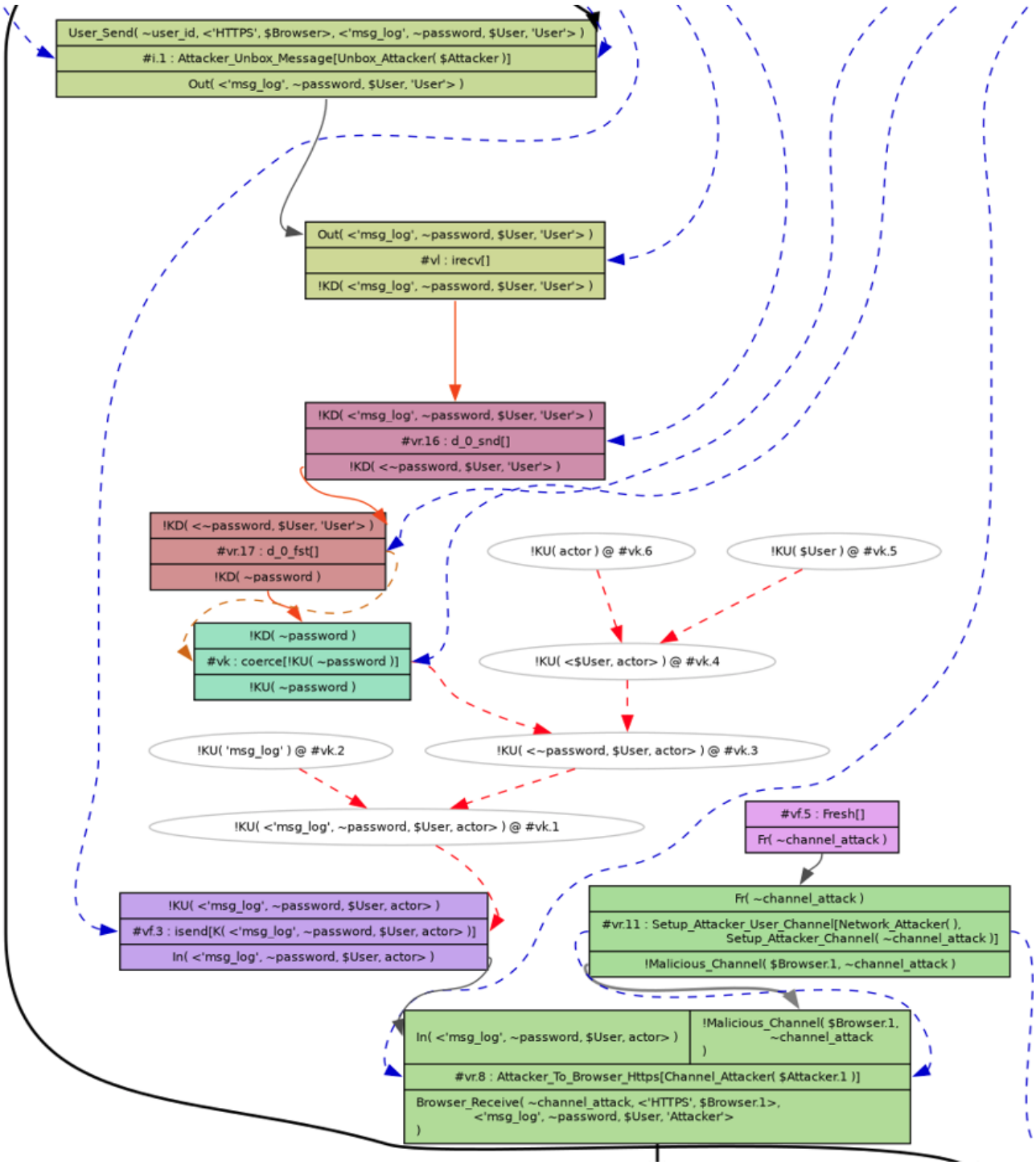

Figura 5.10: Attacante che scompone il messaggio

La Figura 5.10. rappresenta solo la parte le operazioni del attaccante, infatti è un estratto dell'immagine generata del lemma `Ex_Traces_Attack` che verrà trattato nel dettaglio nella prossima sezione.

5.5.2 Analisi Attaccante

L'analisi di un attacco prevede, come già anticipato, 3 diversi lemmi. Forniamo inoltre anche il lemma che descrive il normale funzionamento del protocollo. Il Codice 5.14 presenta i lemmi che catturano e rappresentano l'attacco UNBOX-Attack, grazie al comando di Tamarin Prover "`tamarin-prover UNBOX_Attack.spthy -prove`".

```
processing time: 3.83s

  Ex_Protocol_Complete (exists-trace): verified (8 steps)
  Ex_Traces_Attack (exists-trace): verified (11 steps)
  UNBOX_Attack (all-traces): falsified - found_trace (4 steps)
  All_Traces (all-traces): falsified - found_trace (12 steps)
```

Codice 5.14: Tempi di esecuzione dell'analisi

Ex_Protocol_Complete

Il primo lemma verifica che vi ci sia almeno una traccia in cui la proprietà di completamento del protocollo sia verificata, come già presentato dalla Figura 5.8. Il Codice 5.15 presenta il lemma.

```
lemma Ex_Protocol_Complete:
exists-trace
    "Ex User Browser Server #c. Protocol(User, Browser, Server)@c
    & (Ex User password #i. Completed_Protocol(password, password
   , User)@i & Ex #z. Password(password, User)@z)"
```

Codice 5.15: Lemma per la verifica del completamento del protocollo

Ex_Traces_Attack

Il secondo lemma verifica che vi ci sia almeno una traccia in cui sia presente anche l'attaccante, in questo caso, se lasciamo le regole che descrivono l'attacco, quest'ultime sono sufficienti a rompere il protocollo. La rappresentazione grafica di Tamarin Prover sarà quindi come quella di `Ex_Protocol_Complete`, dove però sarà presente anche l'inserimento e interazione dell'attaccante. Il Codice 5.15 presenta il lemma.

```
lemma Ex_Traces_Attack:
exists-trace
    "Ex User Browser Server #c. Protocol(User, Browser, Server)@c
    &( Ex User password #i. Completed_Protocol(password, password
   , User)@i
       & Ex #z. Password(password, User)@z)
       & ( (Ex Attacker #i. Unbox_Attacker(Attacker) @i) ) "
```

Codice 5.16: Lemma per la verifica della presenza dell'attaccante

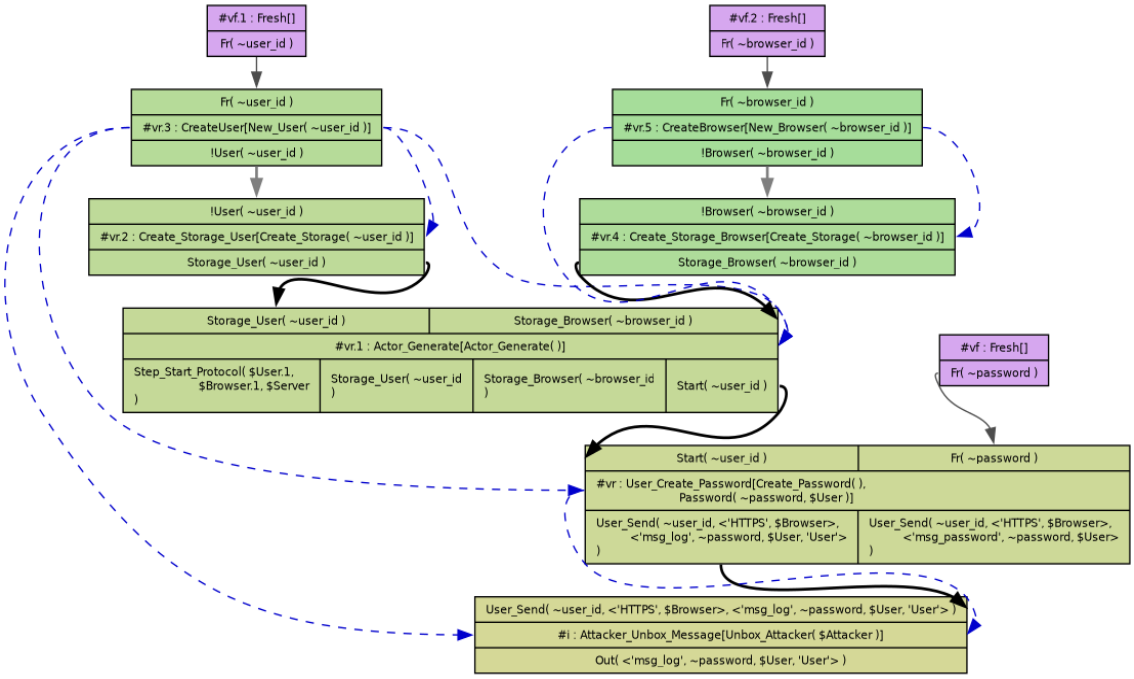

Figura 5.11: Inserimento attaccante nel protocollo

UNBOX_Attack

Il terzo lemma fornisce la sola rappresentazione grafica del punto in cui appare l'attante considerando tutte le possibili tracce. Il Codice 5.17 presenta il lemma.

```
lemma UNBOX_Attack:
"(not(Ex #i. Network_Attacker() @i) &
     (All User1 Browser1 Server1 User2 Browser2 Server2 #i #j.
      Protocol(User1, Browser1, Server1)@i &
      Protocol(User2, Browser2, Server2)@j ==> #i=#j)
     & (All User1 password1 User2 password2 #i #j.
   Completed_Protocol(password1, password1, User1)@i &
   Completed_Protocol(password2, password2, User2)@j ==> #i=#j)
     & (All actor1 actor2 #i. Actor(actor1, actor2)@i ==> actor1=
   actor2 )
)==> not ( (Ex Attacker #i. Unbox_Attacker(Attacker) @i) ) "
```

Codice 5.17: Lemma per la verifica di attacchi di tipo unbox

La figura 5.11 mostra la traccia d'esecuzione prodotta da Tamarin Prover.

All_Traces

Questo lemma non prevede la proprietà del attaccante, infatti vuole sfruttare le operazioni di derivazione di Tamarin Prover. L'idea è di avere tutte le traccie di sicurezza del protocollo, se manteniamo anche le rule dell'attaccante scritte in Tamarin le analizzerà indirettamente, queste sono sufficienti a rompere il protocollo, quindi il risultato sarà un'analisi falsificata, come ci aspettavamo. Il Codice 5.17 presenta il lemma.

```
1 lemma All_Traces:
2   "(All User1 Browser1 Server1 User2 Browser2 Server2 #i #j.
    Protocol(User1, Browser1, Server1)@i &
3   Protocol(User2, Browser2, Server2)@j ==> #i=#j)
4   & (All User1 password1 User2 password2 #i #j.
5   Password_Secure(password1, User1)@i &
6   Password_Secure(password2, User2)@j  ==> #i=#j)
7   & (All User1 password1 User2 password2 #i #j.
    Completed_Protocol(password1, password1, User1)@i &
    Completed_Protocol(password2, password2, User2)@j ==> #i=#j)
8   & (All actor1 actor2 #i. Actor(actor1, actor2)@i
9     ==> actor1=actor2 )"
```

Codice 5.18: Lemma per la verifica di tutte le proprietà del protocollo

5.6 Sessione Parallela

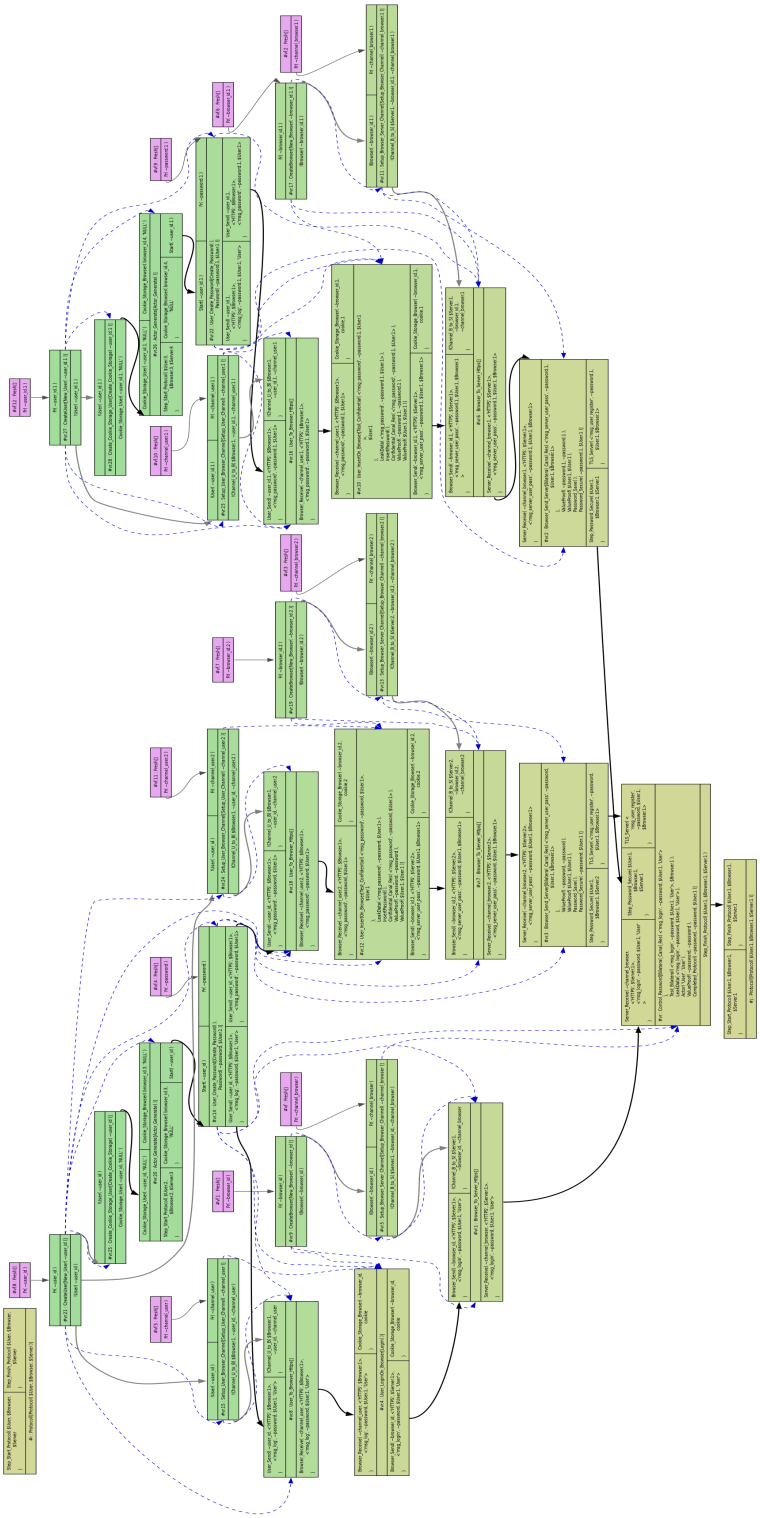

Figura 5.12: Sessione Parallela del Protocollo Basato su Password

L'esecuzione di due sessioni in parallelo dello stesso protocollo permette di rappresentare possibili attacchi più complessi come anche in ASLan++.

Con Tamarin Prover la complessità di derivazione risulta essere ancora più onerosa e con il materiale a disposizione non è stato possibile avere una rappresentazione completa di due sessioni in parallelo. La Figura 5.12 ne mostra una parte.

Capitolo 6

Conclusioni

La sfida originale di questa tesi era la scoperta e studio di Tamarin e il tool Tamarin Prover per trovare e formalizzare strumenti e modelli che permettessero il passaggio da ASLan++. Nel dettaglio provare a mettere sullo stesso piano i due tool e linguaggi in modo tale da confrontarne i risultati, sfortunatamente questo passaggio si è rilevato più complesso di quanto ci si aspettasse, infatti sono comparse sfide e problemi di modellazione con il linguaggio Tamarin inattese che invece in ASLan++ sono già prese in considerazione, un esempio è il problema degli attori nel protocollo che sembrano non essere scontati in Tamarin Prover.

Le tecniche presentate sono i risultati finali della ricerca condotta, infatti sono state intraprese molteplici strade per trovare una risoluzione dei problemi, andando alla scoperta e implementazione di gran parte del manuale di Tamarin. Grazie a queste implementazioni siamo in possesso di strumenti più precisi per la modellazione di un protocollo di sicurezza, infatti possiamo rappresentare un attacco e l'attaccante con modelli precisi e tracce di attacco che mettono in luce diversi aspetti e comportamenti che un utente malevolo può compiere. Inoltre, sempre grazie allo studio dettagliato degli attori nel protocollo, abbiamo un maggior controllo di chi opera durante l'esecuzione.

Per la stesura di questa tesi, sono stati studiati manuali e articoli scientifici per un totale di circa 800 pagine; sono stati inoltre prodotti circa 1600 file di analisi..

Le analisi sono state condotte attraverso l'utilizzo di una WSL con UBUNTU 22.04 LTS, come consigliato sul manuale di Tamarin di ETH, su un DELL XPS 15 del 2023 con i7-13700H e 16 di ram DDR5 di mia proprietà, quest'ultimo, però, risulta essere insufficente per condurre certe tipologie di analisi.

6.1 Lavori futuri

Non siamo ancora in grado di proporre un paragone diretto dei due tool, questi sono lavori e techniche iniziate e non ancora concluse che potrebbero portare a definire costrutti e modelli in grado di completare il passaggio da ASLan++ a Tamarin. Infatti sono stati testati gran parte dei costrutti del manuale di Tamarin, sfortunatamente non possiamo ancora presentarli in quanto privi di revisione e test necessari per validarne il corretto funzionamento.

Qui una proposta di idee e tecniche che potrebbero permettere la formalizzazione più precisa di protocolli.

(i) **Messaggi Complessi**

Concentrarsi sullo studio e la costruzione dei messaggi scambiati tra le regole.

(ii) **Object Oriented Attack**

L'ideale sarebbe creare attacchi modulari per permettere la definizione di proprietà e lemmi più flessibili.

(iii) **Modularità protocollo**

Suddividere il protocollo in sezioni collegabili, in modo da limitare la complessità e gestione delle parti in movimento del protocollo.

(iv) **Modularità Lemmi**

L'ideale sarebbe creare lemmi modulari e componibili per permettere la definizione di proprietà più flessibili.

(v) **Oracoli**

Molte ricerche recenti propongono l'utilizzo delle euristiche nei protocolli in modo tale da lasciare allo stesso Tamarin Prover la creazione delle regole. Una ricerca molto interessante che utilizza questo approccio viene da David Basin, Ralf Sasse e Jorge Toro-Pozo con [4].

Tamarin Prover, grazie alla modalità interattiva, permette di modellare e interagire nella costruzione di un protocollo, questo ci offre la possibilità di svolgere analisi più dettagliate e osservare graficamente le tracce del protocollo con o senza la presenza di attacchi. Questa interazione è possibile solo con regole e vincoli adeguatamente formalizzati, ma questi richiedono tempi considerevoli. Nasce quindi il bisogno di strumenti pronti all'utilizzo.

L'insieme delle implementazioni presentate in questa tesi e quelle future vogliono quindi porsi un nuovo obiettivo: rendere la scrittura di un protocollo più facile e veloce senza rimodellare ogni volta le sezioni del protocollo; questa idea e prospettiva vede la realizzazione di un nuovo linguaggio, ovvero Tamarin++.

Ringraziamenti

Vorrei dedicare questo spazio a chi, con dedizione e pazienza, ha contribuito alla realizzazione di questo elaborato.

Un ringraziamento particolare va al mio relatore prof. Mariano Ceccato che mi ha permesso di intraprendere questo percorso nel centro di ricerca della Fondazione Bruno Kessler.

Un sentito grazie ai ricercatori dr. Marco Pernpruner e dr. Giada Sciarretta che mi hanno seguito durante il tirocinio e la realizzazione dell'elaborato, offrendomi un argomento di studio stimolante e affascinante. Ringrazio anche l'intera unità del centro di "Security & Trust" di FBK che mi ha ospitato durante il mio percorso.

Ringrazio infinitamente i miei familiari, senza i loro insegnamenti e il loro supporto non sarei mai potuto arrivare fin qui.

Ringrazio anche la mia coiquilina che mi ha supportato durante tutto il periodo degli studi.

Infine, vorrei dedicare questo traguardo a me stesso, che possa essere l'inizio di una lunga e brillante carriera professionale.

Bibliografia

[1] Armando Alessandro, Roberto Carbone e Luca Compagna. "SATMC: A SAT-Based Model Checker for Security Protocols, Business Processes, and Security APIs." In: *International Journal on Software Tools for Technology Transfer 18 (2)*. 2016, pp. 187–204. URL: https://doi.org/10.1007/s10009-015-0385-y.

[2] Amin Azmoodeh e Ali Dehghantanha. *Deep Fake Detection, Deterrence and Response: Challenges and Opportunities*. 2022. arXiv: 2211.14667 [cs.CR].

[3] David Basin, Cas Cremers, Jannik Dreier e Ralf Sasse. "Tamarin: Verification of Large-Scale, Real-World, Cryptographic Protocols". In: *IEEE Security Privacy* 20.3 (2022), pp. 24–32. DOI: 10.1109/MSEC.2022.3154689.

[4] David Basin, Ralf Sasse e Jorge Toro-Pozo. "Card Brand Mixup Attack: Bypassing the PIN in non-Visa Cards by Using Them for Visa Transactions". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, ago. 2021, pp. 179–194. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/basin.

[5] Sergey Berezin. "Model checking and theorem proving: a unified framework". Tesi di dott. Carnegie Mellon University, 2002.

[6] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell e A. Scedrov. "A meta-notation for protocol analysis". In: *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. 1999, pp. 55–69. DOI: 10.1109/CSFW.1999.779762.

[7] E. M. Clarke, E. A. Emerson e A. P. Sistla. "Automatic verification of finite-state concurrent systems using temporal logic specifications". In: *ACM Trans. Program. Lang. Syst.* 8.2 (apr. 1986), pp. 244–263. ISSN: 0164-0925. DOI: 10.1145/5397.5399. URL: https://doi.org/10.1145/5397.5399.

[8] CORDIS. "EUROPA - AVANTSSAR Project". In: URL: https://cordis.europa.eu/project/id/216471/it.

[9] Sandra Dünki. "Modelling and Analysis of Web Applications in Tamarin". en. Master Thesis. Zurich: ETH Zurich, 2019. DOI: 10.3929/ethz-b-000372337.

[10] Eric Michael Hutchins, Michael J. Cloppert e Rohan M. Amin. "Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains". In: 2010. URL: https://api.semanticscholar.org/CorpusID:6421896.

[11] Gavin Lowe. “An attack on the Needham-Schroeder public-key authentication protocol”. In: *Information Processing Letters* 56.3 (1995), pp. 131–133. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(95)00144-2. URL: https://www.sciencedirect.com/science/article/pii/0020019095001442.

[12] Simon Meier, Cas Cremers e David Basin. “Strong Invariants for the Efficient Construction of Machine-Checked Protocol Security Proofs”. In: *2010 23rd IEEE Computer Security Foundations Symposium*. 2010, pp. 231–245. DOI: 10.1109/CSF.2010.23.

[13] Simon Meier, Benedikt Schmidt, Cas Cremers e David Basin. “The TAMARIN Prover for the Symbolic Analysis of Security Protocols”. In: *Computer Aided Verification*. A cura di Natasha Sharygina e Helmut Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 696–701. ISBN: 978-3-642-39799-8. URL: https://link.springer.com/chapter/10.1007/978-3-642-39799-8_48.

[14] Roger M. Needham e Michael D. Schroeder. “Using encryption for authentication in large networks of computers”. In: *Commun. ACM* 21.12 (dic. 1978), pp. 993–999. ISSN: 0001-0782. DOI: 10.1145/359657.359659. URL: https://doi.org/10.1145/359657.359659.

[15] Okta. “The State of Secure Identity 2023”. In: 2024. URL: https://www.okta.com/resources/whitepaper-the-state-of-secure-identity-report/.

[16] Lawrence C. Paulson. “The inductive approach to verifying cryptographic protocols”. In: *Journal of Computer Security* 6.1–2 (gen. 1998), pp. 85–128. ISSN: 0926-227X. DOI: 10.3233/jcs-1998-61-205. URL: http://dx.doi.org/10.3233/JCS-1998-61-205.

[17] Marco Pernpruner, Roberto Carbone, Silvio Ranise e Giada Sciarretta. “The Good, the Bad and the (Not So) Ugly of Out-of-Band Authentication with eID Cards and Push Notifications: Design, Formal and Risk Analysis”. In: *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*. CODASPY ’20. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 223–234. ISBN: 9781450371070. DOI: 10.1145/3374664.3375727. URL: https://doi.org/10.1145/3374664.3375727.

[18] Marco Pernpruner, Roberto Carbone, Giada Sciarretta e Silvio Ranise. “An Automated Multi-Layered Methodology to Assist the Secure and Risk-Aware Design of Multi-Factor Authentication Protocols”. In: *IEEE Transactions on Dependable and Secure Computing* (2023), pp. 1–16. DOI: 10.1109/TDSC.2023.3296210.

[19] Associazione Italiana per la Sicurezza Informatica. “Rapporto CLUSIT 2024”. In: 2024. URL: https://clusit.it/rapporto-clusit/.

[20] Cedric Staub. “A user interface for interactive security protocol design”. en. Bachelor Thesis. Zürich: ETH Zurich, 2011. DOI: 10.3929/ethz-a-007554462.

[21] Luca Viganó. "Automated validation of trust and security of service-oriented architectures with the AVANTSSAR platform". In: *2012 International Conference on High Performance Computing Simulation (HPCS)*. 2012, pp. 444–447. DOI: 10.1109/HPCSim.2012.6266956.

[22] ETH Zürich. "GitHub repository". In: URL: https://github.com/tamarin-prover/manual.

[23] ETH Zürich. "Tamarin-Prover". In: URL: https://tamarin-prover.com/.

[24] ETH Zürich. "Tamarin-prover manual". In: 2023. URL: https://tamarin-prover.com/manual/master/book/001_introduction.html.