UNIVERSITÀ DEGLI STUDI DI VERONA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea in Informatica

Tesi di Laurea

Un'Analisi Simbolica per la Verifica di Protocolli di Autenticazione a Stati Infiniti

Relatore: Prof. Roberto Segala

Laureando: Zannone Nicola
Matricola: IN000612

Anno Accademico 2001/2002

When you have eliminated the impossible, whatever remains, however improbable, must be the truth.
Arthur Conan Doyle

As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.
Albert Einstein

Indice

Elenco delle tabelle

Elenco delle figure

Introduzione

La sicurezza delle comunicazioni è sempre stato un problema che l'uomo, in ogni epoca, ha dovuto affrontare e risolvere con i mezzi che aveva a propria disposizione. Corrieri fidati, sigilli, timbri, firma calligrafica e codici di riscrittura degli alfabeti sono solo alcuni degli strumenti che si sono susseguiti nei secoli per garantire l'autenticità, la segretezza e l'integrità nella circolazione delle informazioni. Questi mezzi, nonostante siano ancora oggi utilizzati, non sono idonei a garantire la sicurezza delle comunicazioni telematiche.
La diffusione sempre maggiore delle reti di calcolatori, grazie soprattutto all'espansione di Internet, sta portando profonde innovazioni nei processi economici e sociali offrendo l'opportunità di realizzare servizi di tipo amministrativo, commerciale e finanziario on line. L'utilizzo della rete consente, infatti, di scambiare informazioni in tempi rapidissimi e senza vincoli geografici. La necessità di avere sistemi sicuri di comunicazione nasce quando vengono trasmesse informazioni riservate attraverso la rete, in quanto possono essere intercettate e subire modifiche. Al fine di proteggere le comunicazioni sono quindi indispensabili contromisure mirate a garantire l'integrità, l'autenticità e la riservatezza delle informazioni trasmesse. La risoluzione di dette problematiche è fondamentale per il successo della sicurezza in un ambiente distribuito.
Lo studio della sicurezza tocca molti aspetti di sistemi di calcolatori, dalla costruzione di sistemi sicuri e verifica di programmi, alla progettazione e all'analisi di protocolli di comunicazione. La nostra tesi concentrerà la sua attenzione su quest'ultimo aspetto. Nella trasmissione dei dati l'unico strumento disponibile è l'informazione, per cui la sicurezza delle comunicazioni deve essere garantita dalla ricchezza e dalla forma delle informazioni stesse. Gli strumenti utilizzati per rendere sicure le trasmissioni sono chiamati sistemi crittografici di comunicazione. Questi definiscono il formato e la sequenza dei messaggi che, due o più agenti, in accordo sul loro uso, devono inviarsi, e utilizzano algoritmi crittografici per rendere le informazioni incomprensibili e difficilmente ricostruibili da eventuali intercettori.
Verificare la sicurezza di un sistema crittografico non è semplice. Infatti risulta più facile provare che un protocollo è soggetto ad un attacco, che dimostrare che non esiste alcuna tecnica in grado di violare le proprietà che garantiscono la correttezza del protocollo. Per decenni i protocolli sono stati costruiti basandosi su prove ed errori: una volta proposto uno schema, si cercava di trovare l'errore e se non veniva individuato, lo schema era considerato corretto. La storia ha dimostrato che questo modo di procedere non è soddisfacente ed infatti molti protocolli, presentati in letteratura come sicuri, si sono in seguito dimostrati

inadeguati [22, 56, 58], rilevando la necessità di qualche nuovo strumento in grado di verificare le proprietà garantite da un protocollo. La ricerca effettuata nel settore della sicurezza si è orientata allo studio e allo sviluppo di tecniche formali [12, 15, 22, 17, 32, 83] che si prefiggono di provare la correttezza dei protocolli. La prima parte di questo lavoro mostra come molte analisi dei sistemi crittografici non siano sufficienti a garantire la correttezza dei protocolli per l'incompletezza della descrizione di alcune componenti del sistema e per le assunzioni errate sul sistema di comunicazione che possono compromettere la verifica del protocollo.
Evidenziamo di seguito quelli che, a nostro avviso, sono i problemi principali che si frappongono tra una verifica formale della sicurezza di un protocollo e la sua sicurezza effettiva:

Completezza del sistema deduttivo. Molte tecniche di analisi prevedono una descrizione diretta delle capacità deduttive (o di manipolazione) dei dati delle entità maligne del sistema; poiché tale descrizione è affidata all'esperienza e all'intuito dell'analista, essa potrebbe erroneamente sottovalutare le capacità delle componenti maligne, fornendo una descrizione incompleta del sistema deduttivo. Tale errore può portare ad affermare che un protocollo crittografico è sicuro anche se non lo è.

Descrizione completa delle assunzioni. La mancanza di una descrizione esplicita delle proprietà degli algoritmi crittografici non consente di individuare quali, fra le ipotesi introdotte in una verifica, sono proprietà di tali algoritmi e quali invece sono assunzioni: l'analisi del protocollo può, quindi, non offrire un quadro completo delle condizioni necessarie a garantire il corretto funzionamento del protocollo stesso.

Distinzione di tipo dei dati che costituiscono i messaggi. Una tecnica di verifica che adotta la distinzione del tipo di informazione circolante in rete può non rilevare gli attacchi che si basano sull'incapacità delle parti oneste del sistema di distinguere effettivamente un tipo di dato dall'altro.

In un metodo formale per l'analisi dei protocolli, tali problematiche possono generare l'inattendibilità dei risultati ottenuti. Il nostro obiettivo è sviluppare una tecnica per verificare automaticamente protocolli di autenticazione a stati infiniti tenendo conto dei problemi individuati.
L'autenticazione risulta una componente fondamentale nella progettazione di sistemi distribuiti sicuri. L'autenticazione è il processo mediante il quale gli agenti di un sistema distribuito dimostrano la propria identità ed è utilizzata per proteggerli da eventuali attacchi esterni che tentano di impadronirsi di informazioni riservate.
La progettazione di un'analisi necessita della definizione di un modello, che rappresenti tutte le possibili esecuzioni dei protocolli, e della definizione della nozione di autenticazione, utilizzata per verificare la correttezza dei protocolli stessi.
La nostra analisi si basa sul modello di Dolev e Yao [3, 31]. Detto modello assume che l'agente ostile controlla la rete di comunicazione, cioè l'ambiente, in cui il protocollo agisce, è visto come l'avversario da cui dobbiamo difenderci. L'ambiente è in grado di intercettare i messaggi prodotti dalle parti, replicarli e introdurne di nuovi sfruttando le informazioni ottenute dai messaggi visti. Assume, inoltre, che i meccanismi crittografici, alla base del

sistema, non siano vulnerabili, consentendo attacchi solo alla struttura del protocollo stesso. Tali assunzioni semplificano in modo considerevole la costruzione del modello.
Per modellare i protocolli utilizziamo le algebre di processo, in particolare lo spi-calculus [1], in quanto le tecniche basate su queste algebre hanno il vantaggio di provare le specifiche di sicurezza richieste grazie all'utilizzo di strumenti di verifica dei modelli. La verifica automatica della correttezza di un protocollo, però, non può essere eseguita studiando le tracce di computazione del processo calcolate con la semantica classica dell'algebra. Infatti, poiché l'ambiente può spedire infiniti messaggi, anche l'insieme delle tracce di computazione risulta essere infinito. L'obiettivo della ricerca è definire un modello finito che rappresenti tutte le esecuzioni computazionalmente valide del protocollo.
Basandoci sul Fiore e Abadi [32], otteniamo il risultato voluto attraverso un'analisi simbolica in due fasi. Definiamo, prima, una riduzione simbolica di processi in cui gli input sono valutati formalmente, introducendo una nuova variabile ad ogni azione di input, e quindi senza verificare se il messaggio ricevuto sia deducibile dalla conoscenza dell'ambiente. Il modello ottenuto è si finito, ma contiene esecuzioni non computazionalmente valide. Nella seconda fase è definita una procedura simbolica che analizza la conoscenza dell'ambiente, cioè un algoritmo in grado di stabilire i messaggi deducibili dalla conoscenza dell'ambiente. Combinando i due passi, costruiamo modelli simbolici di processi costituiti da esecuzioni simboliche computazionalmente valide e adatti ad una verifica automatica delle proprietà di sicurezza.
L'uso di tecniche simboliche per l'analisi di sistemi crittografici non è completamente nuovo. In letteratura esistono altri lavori [4, 17] che utilizzano la stessa tecnica, ma si limitano a trattare il caso in cui le chiavi condivise sono vincolate a nomi. Questa tesi si prefigge, invece, di definire un'analisi in grado di studiare il caso in cui le chiavi possano essere qualsiasi messaggio. Questa generalità risulta utile per modellare protocolli in cui certe chiavi non sono nomi, ma sono ottenute applicando, per esempio, funzioni hash a qualche nuovo segreto condiviso da agenti che partecipano al protocollo [38].
L'altro aspetto fondamentale per definire una verifica è formalizzare la nozione di correttezza per protocolli di autenticazione. Gli scopi di tali protocolli sono di garantire l'identità degli agenti e lo scambio di nuove chiavi segrete di sessione per future comunicazioni. Consideriamo separatamente i concetti di autenticazione e di scambio di chiavi convinti di una differenza di intenti. Tra le innumerevoli definizioni di correttezza per protocolli di autenticazione [8, 30, 58], proposte in letteratura, riteniamo che quella di Woo e Lam [96] sia adeguata a risolvere problematiche di autenticazione e si adatti meglio al nostro modello. Tale definizione si basa su due tipi di proprietà primitive: correspondence, cioè la comunicazione tra gli agenti deve avvenire seguendo la sequenza di passi fissati dal protocollo, e secrecy, cioè le informazioni riservate non devono essere conosciute dall'ambiente dopo l'esecuzione del protocollo.
Il nostro obiettivo è calcolare un modello finito, in grado di descrivere il comportamento dei protocolli, e dare definizioni formali adeguate per verificare la correttezza dei protocolli di autenticazione.

Questa tesi è strutturata in tre parti.
La *prima parte* definisce le problematiche che vogliamo affrontare, presenta i meccanismi

attuali di comunicazione sicura e le tecniche utilizzate per la loro verifica.
La *seconda parte* introduce la nostra analisi simbolica per protocolli di autenticazione a stati infiniti. Definisce la sintassi del calcolo con cui rappresentiamo i protocolli in analisi e il concetto di riduzione simbolica. Presenta, inoltre, un algoritmo simbolico capace di calcolare la conoscenza dell'ambiente. Quindi, fornisce una procedura che combina la riduzione simbolica con l'algoritmo per l'analisi della conoscenza per costruire modelli di tracce con cui verificare le proprietà di sicurezza.
La *terza parte* definisce la nozione di sicurezza che consente di stabilire la correttezza dei protocolli e presenta la verifica delle proprietà di sicurezza di alcuni protocolli di autenticazione attraverso l'applicazione della nostra analisi. Inoltre, confronta la nostra analisi con quelle presenti in letteratura e conclude mostrando i risultati raggiunti e indicando alcuni spunti per estendere l'analisi.

Parte I

Sicurezza e Protocolli di Autenticazione

Introduzione

La prima parte è divisa in cinque capitoli e identifica gli argomenti di questa tesi.
Il *Capitolo 1* presenta il concetto di comunicazione e le sue proprietà di sicurezza. Introduce inoltre le basi della crittografia quale componente essenziale nell'implementazione di un sistema sicuro. Fa, infine, una panoramica delle tipologie di attacco da cui ci si deve difendere.
Il *Capitolo 2* definisce il significato di protocollo, dandone una classificazione. Presenta poi i modi in cui un agente ostile può interagire con un protocollo allo scopo di compromettere il funzionamento del protocollo stesso. Mostra, inoltre, come differenti autori in letteratura abbiano definito il significato di autenticazione e conclude fornendo una tassonomia dei protocolli di autenticazione.
Il *Capitolo 3* affronta le tecniche di analisi, sia formali che informali, dei sistemi di crittografia (crypto system) presenti in letteratura, descrivendone i pregi e i difetti.
Il *Capitolo 4* approfodisce gli strumenti per modellare i sistemi distribuiti studiati in letteratura e le loro proprietà di sicurezza. In particolare, concentra l'attenzione sulle algebre di processo.
Il *Capitolo 5* intrododuce l'analisi simbolica mettendo in rilievo l'utilità di un tale metodo per la verifica di sistemi di crittografia.

Capitolo 1

Sicurezza delle Comunicazioni e Crittografia

La sicurezza può essere definita come la misura necessaria per proteggersi da situazioni di spionaggio, di attacco o di crimine in genere. Dal punto di vista dell'informatica per sicurezza si intendono tutte le attività che permettono la protezione di sistemi informatici attraverso la verifica delle autorizzazioni, dove solo utenti identificati sono abilitati all'accesso alle reti, alle applicazioni, ai dati e questo sulla base di liste per il controllo degli accessi. I servizi di sicurezza sono ottenuti mediante l'uso di algoritmi che si basano sulla crittografia.

1.1 Sicurezza delle Comunicazioni

Sicurezza delle comunicazioni significa proteggere la segretezza e l'integrità dei dati trasmessi da un agente ad un altro. Fino alla prima guerra mondiale la sicurezza era stata caratterizzata da una certa staticità e le tecniche utilizzate sono oggi considerate obsolete; negli anni seguenti, ha avuto un grande sviluppo soprattutto per interessi militari.
La sicurezza dei sistemi informatici nasce negli anni '70 e diventa presto un requisito essenziale per molte applicazioni a causa della diffusione di sistemi distribuiti e di reti di calcolatori. La mobilità sta fortemente modellando questi sistemi, conducendo a nuovi scenari in cui il problema della sicurezza diventa sempre più urgente. Il software eseguito su una macchina non necessita di essere prodotto su di essa, ma può essere scaricato da un server da qualche parte nella rete. Conseguentemente, ogni ambiente di comunicazione offre una piattaforma distribuita ai programmi che possono essere eseguiti concorrentemente dagli utenti o localmente o in remoto. Questo ha causato un aumento considerevole della ricerca nel settore della sicurezza e in particolar modo nel settore della *computer security*. Con computer security si intendono i diversi aspetti di sicurezza che devono essere garantiti in un sistema informatico: protezione fisica dell'hardware, protezione dei dati memorizzati e protezione dei dati trasmessi.
Questa tesi affronta alcuni aspetti della sicurezza della trasmissione telematica delle informazioni. Alcune delle proprietà che si vogliono garantire nella trasmissione dei dati sono:

- **Segretezza e riservatezza**: le informazioni private scambiate tra le parti comunicanti non devono entrare in possesso di entità non autorizzate, cioè ogni agente desidera che nessuno, eccetto lui o altri agenti autorizzati, possa conoscere quelle informazioni.

- **Integrità**: i dati trasmessi non devono essere alterati senza che le parti comunicanti se ne avvedono, cioè ogni agente desidera che nessuna entità ostile possa modificare o distruggere quelle informazioni.

- **Non Interferenza**: questo è un caso particolare dei problemi precedenti. La richiesta di proteggere le informazioni private scambiate si basa su livelli di sicurezza. Una classica applicazione di tale proprietà è la *Multilevel Security* [90]. Un sistema che implementa la Multilevel Security deve imporre le seguenti due regole.

  - **No Read Up**: un agente può leggere solo dati di livello di sicurezza minore o uguale. Questa si trova in letteratura come *Simple Securety Property*.
  - **No Write Down**: un agente può scrivere solo su dati di livello di sicurezza maggiore o uguale. Questa si trova in letteratura come **-Property*.

- **Autenticazione**: le parti comunicanti devono poter ricevere sufficienti garanzie sull'identità del partner nelle comunicazioni, cioè mittente e destinatario di una trasmissione di dati vogliono essere certi della provenienza e della destinazione dei messaggi.

- **Non repudiabilità**: in alcuni casi deve essere garantita l'identità del mittente di un messaggio in modo inequivocabile e permanente, cioè un agente, in qualsiasi momento, può risalire al mittente di un messaggio.

Un sistema sicuro di comunicazione deve proteggere le informazioni scambiate tra gli agenti da possibili attacchi esterni. Il problema della sicurezza delle comunicazioni viene così affrontato introducendo particolari schemi di trasmissione in grado di garantire i requisiti richiesti. Gli schemi più utilizzati adottano meccanismi crittografici e vengono chiamati protocolli crittografici di comunicazione.

1.2 Crittografia

La crittografia è la scienza che si occupa di scrivere messaggi che nessuno, al di là del vero destinatario, può leggere. La crittografia studia le tecniche per mascherare l'informazione trasmessa su un canale pubblico, con lo scopo di difenderla dalle intrusioni di utenti non autorizzati a riceverla. Il *testo in chiaro* (o *plaintext*) è costituito dalle parole, dai caratteri o dalle lettere del messaggio originale in forma comprensibile. Il *testo cifrato* (o *ciphertext*) è costituito dalle parole, dai caratteri o dalle lettere della versione segreta del messaggio. La *cifratura* non è altro che il procedimento che consente di ottenere il testo cifrato, o crittogramma, partendo dal testo in chiaro. L'operazione opposta, cioè il passaggio dal testo cifrato al testo in chiaro, prende il nome di *decifratura* se eseguita dall'utente legittimo, oppure di *decriptazione* se eseguita da un utente esterno non autorizzato, utilizzando strumenti di criptoanalisi.

Le prime tecniche crittografiche basavano l'indecifrabilità di un messaggio criptato sulla segretezza degli algoritmi di codifica e decodifica utilizzati dagli agenti che volevano comunicare. In questo tipo di approccio la sicurezza dei dati trasmessi veniva a mancare una volta che venivano scoperti gli algoritmi di codifica e decodifica.
La crittografia moderna si basa su algoritmi di dominio pubblico e imposta la sicurezza delle informazioni sull'utilizzo di chiavi segrete durante le fasi di codifica e decodifica dei dati. La segretezza di un'informazione codificata si basa sullo spazio delle chiavi che un agente ostile deve provare per riuscire ad ottenere l'informazione. Un algoritmo è un procedimento di calcolo, un ben preciso schema di operazioni, non solo matematiche, ma anche logiche, da compiere per risolvere una determinata classe o tipologia di problemi. E' importante notare come l'algoritmo usato per risolvere un problema non dipende dai dati che, di volta in volta, il problema può presentare, ma solo dalla sua stessa struttura. Caratteristica altrettanto importante di un algoritmo è la terminazione, ovvero che prima o poi restituisce un risultato. Infatti tale algoritmo risulterebbe inutilizzabile se questo continuasse a calcolare in eterno. Quando si parla di sicurezza, gli algoritmi a cui ci si riferisce sono gli algoritmi di hashing e di crittografia. Una funzione hash H è una trasformazione che prende in input un testo di lunghezza arbitraria X e restituisce una stringa h di lunghezza fissata. Questa stringa rappresenta un'*impronta digitale* unica del messaggio e viene spesso definita valore di hash o checksum crittografico. Le funzioni hash con solo questa proprietà vengono utilizzate per risolvere problemi di carattere generale, ma quando vengono usate in crittografia, le funzioni hash sono solitamente scelte in modo tale da avere alcune proprietà addizionali. I prerequisiti per le funzioni hash crittografiche sono i seguenti:

- L'input può essere di qualsiasi lunghezza.
- L'output ha una lunghezza fissata.
- $H(x)$ è relativamente semplice da calcolare per ogni x dato.
- $H(x)$ è one-way.
- $H(x)$ è collision free.

Una funzione è detta essere *one-way* se è difficile da invertire, dove "difficile da invertire" significa che, dato un valore di hash h, non è computazionalmente possibile trovare qualche input x tale che $H(x) = h$. Se, dato un messaggio x, non è computazionalmente possibile trovare un messaggio y tale che $H(x) = H(y)$, allora H è detta essere una funzione hash *collision free*.
Gli algoritmi crittografici hanno il compito di rendere segreto un testo basandosi su una chiave. Le caratteristiche base per un algoritmo crittografico sono le seguenti:

- Lavora su una sequenza di dati iniziale, i dati da crittografare;
- Genera una seconda sequenza, contenente i dati crittografati;
- Basa il suo lavoro su una chiave.

La chiave viene utilizzata dall'algoritmo per crittografare (o cifrare) i dati. Gli algoritmi crittografici che fanno uso di chiavi possono essere divisi in due gruppi:

- algoritmi crittografici a chiave condivisa, o algoritmi simmetrici,
- algoritmi crittografici a chiave pubblica, o algoritmi asimmetrici.

1.2.1 Crittografia a Chiave Condivisa

Un algoritmo di crittografia si dice simmetrico se codifica e decodifica dati utilizzando la stessa chiave. Di solito sono più semplici e più veloci degli algoritmi asimmetrici, e consentono di crittografare anche sequenze di dati molto grandi. Sono loro che si occupano di crittografare le pagine web protette oppure i nostri messaggi di posta elettronica confidenziali.
Da una sequenza di dati, che rappresenta il messaggio, è possibile creare un'altra sequenza apparentemente incomprensibile da trasmettere in tutta sicurezza. Il processo di trasformazione prende il nome di crittografia e viene realizzato eseguendo un algoritmo che elabora la sequenza di dati iniziali e genera la sequenza di dati cifrati. Successivamente, il destinatario del messaggio, che conosce la chiave utilizzata nell'algoritmo crittografico, con un algoritmo opposto (di decodifica) elabora i dati cifrati riottenendo il messaggio iniziale. È fondamentale notare che la chiave utilizzata nell'algoritmo di codifica è la stessa che viene utilizzata nell'algoritmo di decodifica.

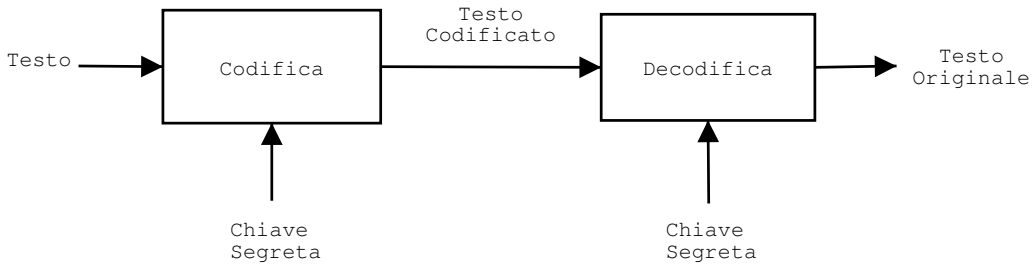


Figura 1.1. Codifica e decodifica nel caso di crittografia a chiave condivisa

A questo punto sembrerebbe che i nostri dati siano al sicuro. Il problema però si è solo spostato, infatti, se si desidera inviare dati cifrati a un amico, è necessario dirgli anche con che chiave sono stati crittografati, ovvero, è necessario inviare anche la chiave per poterli decifrare. La chiave viaggerebbe in chiaro, non crittografata, altrimenti servirebbe una seconda chiave per crittografare la prima, ma questa seconda viaggerebbe in chiaro, e così via. Basterebbe intercettare la chiave e si potrebbe decifrare tutto il messaggio. Questo è il limite fondamentale degli algoritmi simmetrici. Loro proteggono i dati, ma chi protegge la chiave? La sicurezza di questo tipo di algoritmi risiede nella segretezza della chiave utilizzata dalle parti comunicanti; divulgare la chiave segreta significa dare la possibilità a chiunque di codificare e decodificare i messaggi circolanti tra i due agenti. Affinché una data informazione

criptata rimanga segreta per un certo periodo bisogna che anche la chiave utilizzata per la codifica lo rimanga per lo stesso periodo.
Gli algoritmi simmetrici più utilizzati sono l'RC2 con chiavi da 40, 60 e 128 bit, il DES con chiavi da 56 bit e il Triple-DES, o 3DES, con chiavi da 168 bit.

1.2.2 Crittografia a Chiave Pubblica

Fino a quasi trent'anni fa erano conosciuti solo algoritmi a chiave simmetrica. Nel 1976 Diffie ed Hellman [29] presentarono un protocollo per lo scambio di una chiave segreta attraverso un canale insicuro. Tale meccanismo fu ideato essenzialmente per risolvere il problema dell'avvio di un normale sistema di cifratura a chiavi simmetriche, ma in realtà pose le basi della crittografia a chiave pubblica.
Un algoritmo di crittografia si dice asimmetrico se opera con due chiavi (una detta pubblica, l'altra privata). La chiave pubblica è una sequenza di bit utilizzata da un algoritmo asimmetrico per crittografare dei dati. La chiave pubblica, come dice il nome, può essere distribuita in chiaro, ovvero non crittografata, senza alcun problema. Esistono persino archivi online di chiavi pubbliche di migliaia di persone. Tutto ciò è possibile a causa delle caratteristiche degli algoritmi asimmetrici: la chiave pubblica non può essere utilizzata per decriptare alcunché: la sua trasmissione, anche in chiaro, non è affatto pericolosa. La chiave privata, invece, è una sequenza di bit utilizzata da un algoritmo asimmetrico per decodificare i dati crittografati con la chiave pubblica corrispondente. Al contrario della chiave pubblica gemel-

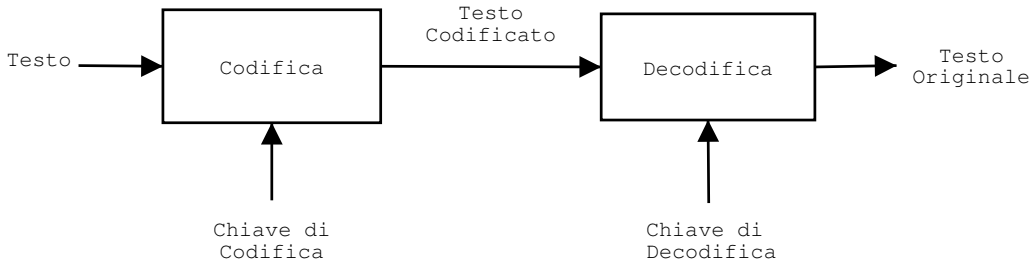


Figura 1.2. Codifica e decodifica nel caso di crittografia a chiave pubblica

la, non viene mai trasmessa a nessuno. È utilizzata, ad esempio, per decrittografare le email crittografate ricevute. Chiunque può accedere alla chiave privata di un utente è in grado di decriptare la sua posta crittografata. Naturalmente accedervi senza la sua autorizzazione è impossibile, ma può essere rischioso non proteggere la chiave privata con password o altri trucchi.
Data una sequenza di dati di partenza, questa viene crittografata utilizzando una chiave e può venir decodificata solamente attraverso l'altra chiave. Di norma gli algoritmi asimmetrici sono molto lenti, complessi, e adatti a crittografare solo sequenze di dati di dimensione finita e sono spesso utilizzati per crittografare le chiavi usate dagli algoritmi simmetrici.
A volte la chiave privata viene utilizzata per ottenere la codifica di un'informazione, ad esempio per firmare le email (ovvero per firmare gli hash delle email, che sono poi controllati

dal programma di posta elettronica del destinatario), mentre la chiave pubblica può essere utilizzata per la decodifica della stessa. Questo procedimento, chiamato *firma elettronica*, consente di garantire l'identità del mittente di un messaggio al destinatario. Inoltre l'utilizzo della firma garantisce al destinatario di un messaggio che i dati ricevuti non siano stati modificati. Chiunque conosca la chiave pubblica appropriata è in grado di verificare chi ha scritto il messaggio. Usando la chiave pubblica un agente può ricostruire il checksum del messaggio. Una volta calcolato il checksum viene confrontato con il checksum che era stato aggiunto al messaggio. Se i due checksum sono identici, sicuramente il proprietario della chiave privata ha creato questa firma e i dati non sono stati modificati. Notiamo quindi che l'utilizzo della chiave pubblica per la codifica garantisce la segretezza delle informazioni, ma non l'origine.
L'algoritmo asimmetrico più utilizzato è l'RSA, che lavora con chiavi da 512 bit e 1024 bit, nella sua versione più sicura.

1.2.3 Criptoanalisi

Lo scopo della crittografia è rendere incomprensibile l'informazione a chi non è autorizzato alla sua interpretazione. Nelle reti telematiche esistono varie entità in grado di intercettare ogni comunicazione che vi transita e che tentano di scoprire il contenuto di tali comunicazioni. Chiamiamo, in seguito, tali intercettatori intrusi o attaccanti.
La criptoanalisi è la disciplina che studia come decifrare un messaggio senza conoscere la chiave. Le principali tecniche di criptoanalisi, chiamate anche attacchi, possono essere classificate secondo il seguente schema:

- **Ciphertext-Only Attack**: l'intruso tenta di ricostruire alcune chiavi o alcuni messaggi in chiaro da un insieme di messaggi codificati a sua disposizione.
- **Known-Plaintext Attack**: l'intruso cerca di ricostruire la chiave segreta da alcuni messaggi criptati e dalle rispettive decodifiche che ha a sua disposizione.
- **Chosen-Plaintext Attack**: l'intruso non solo può disporre di messaggi criptati e le rispettive decodifiche, ma può scegliere una serie di messaggi da farsi codificare al fine di poter derivare la chiave.
- **Adaptive-Chosen-Plaintext Attack**: questo è un caso particolare dell'attacco precedente. Non solo l'intruso è in grado di scegliere i messaggi da codificare, ma può anche modificare le sue scelte in base ai risultati delle codifiche precedenti in modo da raffinare l'attacco per giungere prima al suo scopo di derivare la chiave.
- **Chosen-Ciphertext Attack**: l'intruso può scegliere una serie di ciphertext da farsi decodificare al fine di poter derivare la chiave segreta di una qualche componente del sistema. Questo tipo di attacco è usato specialmente contro sistemi che usano crittografia a chiave pubblica.

È interessante notare come in letteratura non si parli di algoritmi sicuri, ma di algoritmi che probabilmente lo sono; non si esclude la possibilità che esista una tecnica che possa consentire in futuro di rompere, con una certa semplicità, i meccanismi crittografici attualmente

utilizzati.
Un algoritmo crittografico è considerato sufficientemente sicuro se il costo necessario per romperlo è più elevato del valore dei dati codificati. Questo costo non si riferisce solamente ad un valore economico, ma anche alle risorse necessarie per effettuare un eventuale attacco. Un parametro di valutazione frequentemente utilizzato consiste nel calcolare il costo di un *brute force attack*, ovvero nel calcolare il costo necessario per rompere una codifica procedendo per tentativi: l'attaccante prova tutte le possibili chiavi, una alla volta, sino a quando non individua quella corretta.

Capitolo 2

Protocolli di Autenticazione

Un protocollo è una sequenza di passi che coinvolge due o più parti, progettata per il raggiungimento di uno specifico scopo. In un protocollo l'ordine di esecuzione e la presenza di tutti i passi fissati è fondamentale per il raggiungimento dello scopo fissato. Un protocollo di comunicazione è uno schema che stabilisce il formato e la sequenza dei messaggi che due o più parti, in accordo sul suo utilizzo, devono inviarsi. Ogni passo del protocollo viene scritto nella forma "$P \rightarrow Q \ : \ m$" che rappresenta la comunicazione del messaggio m dall'agente P all'agente Q.
Questo capitolo definisce il significato di protocollo dandone una classificazione e presenta i modi in cui un agente ostile può interagire con un protocollo allo scopo di comprometere il funzionamento del protocollo stesso. Mostra, inoltre, come differenti autori in letteratura abbiano definito il significato di autenticazione e conclude fornendo una tassonomia dei protocolli di autenticazione.

2.1 Protocolli Crittografici di Comunicazione

Un sistema crittografico utilizza un determinato tipo di algoritmo crittografico per garantire la sicurezza delle comunicazioni che i suoi passi prevedono. Nei sistemi crittografici la presenza di agenti ostili, cioè osservatori passivi o attaccanti attivi, rende necessario modellare i protocolli in modo tale che:

- Il protocollo deve essere non ambiguo: ogni passo deve essere ben definito e completo, deve quindi essere definita un'azione per ogni possibile situazione.
- Gli agenti devono conoscere solo quello che il protocollo decide che possono conoscere.
- I messaggi del protocollo devono fornire informazioni utili solo agli agenti onesti.
- I messaggi del protocollo devono essere definiti in modo tale che solo agenti onesti li possano generare.

La crittografia viene così usata per prevenire o individuare un eventuale attacco da parte di un agente ostile durante l'esecuzione del protocollo. A tale scopo sono state ideate tre tipologie principali di protocolli.

- **Arbitrated Protocol**: questo tipo di protocollo prevede la presenza nelle comunicazioni di una parte disinteressata ed onesta in grado di garantire la correttezza delle trasmissioni che avvengono tra due agenti che, in accordo, decidono di seguire quel protocollo.

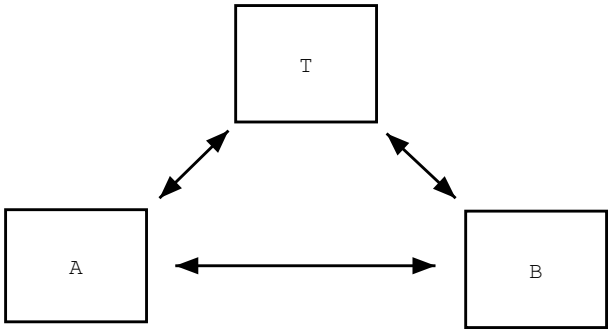


Figura 2.1. Arbitrated Protocol

Questa tipologia di protocollo presenta una serie di problemi.

1. Risulta complicato identificare una terza parte che abbia i requisiti richiesti. Due parti che ricorrono ad un protocollo di autenticazione per verificare l'identità del partner nella comunicazione, non sono disposte a fidarsi di una terza parte se questa non possiede tutte le caratteristiche desiderate.
2. Le parti devono sostenere i costi di mantenimento di una terza parte sia in termini economici che in termini di overhead delle comunicazioni.
3. La comunicazione tra le due parti nei protocolli di questo tipo ha un rallentamento dovuto alla presenza della terza parte.
4. Ogni agente che partecipa al protocollo deve assumere che la terza parte sia fidata, di conseguenza la terza parte rappresenta un punto vulnerabile nell'intero sistema.

Un esempio di questo tipo di protocollo è il Wide Mouthed Frog Protocol [1, 12, 41]. Questo protocollo è composto dai tre seguenti messaggi.

$$\begin{array}{lcl} \text{Messaggio 1} & A \rightarrow S & : \{K_{AB}\}_{K_{AS}} \\ \text{Messaggio 2} & S \rightarrow B & : \{K_{AB}\}_{K_{BS}} \\ \text{Messaggio 3} & A \rightarrow B & : \{M\}_{K_{AB}} \end{array}$$

Gli agenti A e B condividono, rispettivamente, le chiavi K_{AS} e K_{SB} con un server fidato S. Quando A e B vogliono comunicare in modo sicuro, A crea una nuova chiave

K_{AB}, la manda al server codificata con la chiave K_{AS}, e il server la inoltra a B codificata con la chiave K_{SB}. Poiché tutte le comunicazioni sono protette da crittografia le comunicazioni possono avvenire attraverso canali pubblici.

- **Adjudicated Protocol**: questo tipo di protocollo è stato progettato per contenere il costo dell'esecuzione di un protocollo arbitrario. Esso è costituito da due differenti sottoprotocolli: il primo di tipo non arbitrario e viene utilizzato nelle comunicazioni comuni senza offrire nessuna garanzia alle parti comunicanti; il secondo invece è di tipo arbitrario e viene utilizzato quando le parti comunicanti entrano in disputa tra di loro. In questo caso l'arbitro viene detto adjudicator, in quanto è in grado di valutare i messaggi passati in rete in precedenza.

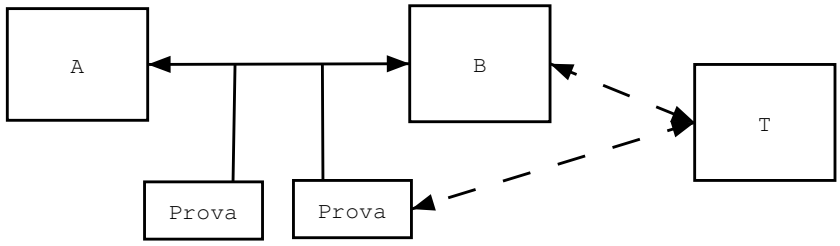


Figura 2.2. Adjudicated Protocol

Le parti A e B comunicano tra loro senza nessun intermediario T se non vi è alcun problema, altrimenti ricorrono a T per risolvere, grazie all'uso di prove che confermino la propria tesi, le eventuali dispute.
Un buon esempio è presentato da Simons [88]. Alice vuole vendere la sua macchina a Bob e Bob vuole pagare con un assegno. Alice non conosce Bob, e non si fida della copertura dell'assegno. Così non vuole dare la macchina a Bob finché la banca non ha accettato l'assegno. D'altra parte Bob non si fida di Alice, e ha paura che Alice, una volta incassato l'assegno, non gli dia le chiavi e il libretto di circolazione della macchina. Vediamo ora lo schema di un Adjudicated Protocol per risolvere questo problema.

- Alice dà le chiavi e il libretto di circolazione della macchina a Bob.
- Bob dà l'assegno ad Alice.
- Se l'assegno è scoperto, o se le chiavi e il libretto di circolazione risultano essere falsi, Bob e Alice appaiono davanti al giudice ed entrambi presentano le loro prove. Il giudice decide sulle prove e la parte che ha commesso la truffa viene condannata.

Il compito del giudice, comunque, potrebbe non essere facile in quanto la parte truffata potrebbe non essere in grado di presentare prove certe e non vedere di conseguenza riconosciuta la sua ragione. Ovviamente, è compito del protocollo produrre prove non

confutabili della truffa. Un altro aspetto dei protocolli di questo tipo è la pena che viene assegnata all'agente che ha commesso la truffa. La pena deve essere sufficiente per scoraggiare le truffe.

- **Self-Enforcing Protocol**: un protocollo di questo tipo è progettato in modo tale da rendere una truffa virtualmente impossibile. La sicurezza delle comunicazioni è garantita dalla struttura stessa dei messaggi del protocollo. Se qualche agente partecipante al protocollo compie una truffa, il truffatore è immediatamente scoperto dagli altri agenti. Le prestazioni offerte da questo tipo di protocollo sembrerebbero migliori di quelle offerte dai precedenti due tipi.

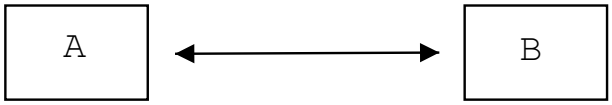


Figura 2.3. Self-Enforcing Protocol

In un mondo ideale, tutti i protocolli dovrebbero essere self-enforcing, ma sfortunatamente non tutti i problemi hanno ancora una soluzione self-enforcing. I protocolli di questo tipo sono generalmente molto onerosi per tutti gli agenti e quindi sono poco efficienti. Per questo motivo arbitrated e specialmente adjudicated protocol sono usati comunemente, perfino se, per lo stesso compito, è disponibile un protocollo self-enforcing.

2.2 Attacchi a Protocolli

Un sistema sicuro di comunicazione deve proteggere le informazioni scambiate tra gli agenti da possibili attacchi. Si possono individuare due tipologie di attacchi. La prima può essere indirizzata direttamente contro gli algoritmi crittografici, mentre l'altra contro la struttura stessa dei protocolli. Nel primo caso l'attacco viene chiamato *passivo* perché l'attaccante non modifica la struttura del protocollo. L'agente ostile utilizza la crittoanalisi per ottenere informazioni utili per rompere il protocollo. Poiché l'attacco passivo è difficile da individuare, solitamente il protocollo cerca di prevenirlo piuttosto che di individuarlo. Nel secondo caso l'attacco viene chiamato *attivo* perché l'attaccante interviene attivamente sulla struttura del protocollo allo scopo di compromettere il funzionamento del protocollo stesso. I modi di questo tipo di attacco dipendono da molti fattori tra cui la tipologia della rete, la struttura e la sequenza dei messaggi, e il numero di partecipanti alla comunicazione.
In generale gli attacchi attivi sono considerati più pericolosi di quelli passivi in quanto, oltre a compromettere la segretezza di alcune informazioni, possono consentire all'attaccante l'utilizzo di risorse riservate con la possibile modifica o distruzione dei dati appartenenti agli agenti delle comunicazioni.

2.2.1 Attacchi Passivi

Prendiamo in esame gli attacchi che dipendono solo dalle potenzialità crittografiche possedute dagli attaccanti. La crittografia è usata per rendere sicura una conversazione elettronica tra due agenti. Consideriamo, per esempio, due uomini d'affari che discutono su un possibile accordo tra due grosse compagnie. Agenti ostili potrebbero essere molto interessati alle informazioni scambiate in queste conversazioni. Questi possono provare a spiare i dettagli dell'accordo. Una possibilità è data dall'"origliare" la linea telefonica dei due uomini d'affari. Questo è considerato un attacco passivo perché l'attaccante si limita ad ascoltare senza interagire attivamente sulla struttura del protocollo.
Per anni, la crittografia a chiave condivisa era il solo modo per difendersi da questo tipo di attacco. Il punto debole era lo scambio delle chiavi per l'algoritmo di codifica e decodifica, in quanto l'attaccante leggeva sicuramente anche le chiavi ed era così in grado di decriptare la conversazione. La crittografia a chiave pubblica risolve in parte questo problema. Spesso, la sola conoscenza che una conversazione ha avuto luogo è tutto quello che un attaccante vuole sapere. Se un attaccante passivo nota che i capi di due grosse compagnie comunicano spesso tra loro, potrebbe essere in grado di indovinare che cosa sta succedendo. Questo tipo di attacco è chiamato *monitoring* o *traffic analysis* [88]. La persona che monitorizza la linea telefonica, anche se non è in grado di ricostruire il contenuto del messaggio, sa che è avvenuta una conversazione criptata.

2.2.2 Attacchi Attivi

La sicurezza è una proprietà cruciale del comportamento dei sistemi e richiede un controllo preciso sul flusso di informazione tra le parti di un sistema di comunicazione. Uno degli obiettivi principali è limitare, e possibilmente evitare, i danni prodotti da attaccanti attivi che cercano di scoprire e trasmettere informazioni sicure interagendo attivamente sulla struttura del protocollo. La maggior parte di questo tipo di attacco si può suddividere nelle seguenti categorie.

Man-in-the-middle Attack: l'agente ostile conduce sessioni simultanee con A e B in modo da convincere uno dei due partecipanti che la sessione è stata attivata con l'altro partecipante.
Vediamo un esempio di come funziona tale tipo di attacco. Mallory, un attaccante attivo, ha il controllo della linea di comunicazione tra i direttori di due compagnie, Alice e Bob. Alice manda a Bob la sua chiave pubblica per renderlo abile a risponderle con un messaggio cifrato. Mallory intercetta la chiave pubblica, ne genera una nuova e la spedisce a Bob. Bob riceve una chiave che lui pensa essere quella di Alice. Così Bob manda la sua chiave pubblica indietro sulla stessa linea. Mallory intercetta e rimpiazza anche questa chiave con una delle proprie. Ora Alice e Bob iniziano il loro scambio di messaggi. Mallory intercetta il messaggio di Alice, che è codificato con la chiave che lei assume essere quella di Bob. Dopo aver decriptato il messaggio, Mallory codifica il plaintext con la chiave pubblica di Bob e manda a Bob il messaggio codificato. Bob riceve un messaggio valido codificato con la sua chiave pubblica e assume che il trasmettitore sia Alice, e così via.

Notiamo che Alice e Bob non si accorgono se qualcuno sta leggendo i loro messaggi. Inoltre, non solo Mallory può leggere la comunicazione cifrata, ma può anche modificare o riscrivere il testo del messaggio per prevenire un accordo tra Alice e Bob.

Parallelal Session Attack: questo tipo di attacco è una generalizzazione dell'attacco precedente. Rappresenta ogni situazione in cui l'agente ostile riesce a gestire sessioni intervallate ingannando uno o più agenti legittimi.

Duplicate Session: l'agente ostile riesce a convincere un agente di essere coinvolto in una seconda esecuzione di un protocollo, sebbene l'interlocutore di quest'ultimo non sia a conoscenza della seconda esecuzione. Generalmente in questo tipo di attacco l'agente ostile riesce a ingannare un agente facendogli riaccettare un oggetto che ha accettato in precedenza.

Covert Channel Attack: questo tipo di attacco avviene solitamente contro sistemi che adottano la Multilevel Security. Ricordiamo che la Multilevel Security si basa su due regole: la No Read Up e la No Write Down. Focardi e Gorrieri [33] hanno mostrato, però, che queste regole di accesso non sono sufficienti. Potrebbe essere possibile trasmettere indirettamente informazioni usando qualche side effect del sistema. Per esempio, se due livelli di sicurezza, *high* (alto) e *low* (basso), condividono qualche risorsa di memorizzazione finita, è possibile trasmettere dati dal livello alto al livello basso di sicurezza usando il messaggio di errore "full resource". Per un trasmettitore a livello high è sufficiente riempire o svuotare alternativamente la risorsa rispettivamente per trasmettere un 1 o uno 0. Simultaneamente l'agente ricevente a livello low cerca di scrivere sulla risorsa, decodificando ogni messaggio di errore come un 1 e ogni successo di scrittura come uno 0. È chiaro che tale modo indiretto di trasmettere, chiamato *covert channel attack* [33, 41, 50, 100], non viola le due regole di accesso multilivello. Il punto debole dei modelli basati sul controllo degli accessi è la mancanza di una semantica precisa, nel senso che l'identificazione dei soggetti e degli oggetti e la loro mappa di diritti di accesso è lasciata all'implementatore. Scegliere questi elementi è un lavoro estremamente difficile se si vuole un modello che consideri tutti i possibili canali dal livello alto al livello basso di sicurezza. Normalmente l'implementatore, dopo aver mappato le primitive del modello al sistema, necessita di studiare le capacità dei canali rimanenti, usualmente chiamati *covert channel*.

Replay Attack: un rischio importante per i collegamenti è l'autenticazione falsa, cioè l'aggiramento dell'autenticazione richiesta agli utenti. Infatti un attaccante potrebbe fingere di essere un utente legale del sistema. Questo rischio è aumentato da alcune proprietà delle password. Solitamente, se si vuole trasmettere un'informazione riservata attraverso la rete, si può cifrare il dato e spedirlo in quella forma. Questa procedura non è sufficiente se le informazioni possono essere usate anche se non decriptate. Per esempio, l'utilizzo di password cifrate non garantisce la sicurezza perché un agente ostile che sta usando sniffing, può semplicemente intercettare e rispedire la password cifrata senza doverla decriptare. Nei replay attack, una trasmissione di dati valida è ripetuta, o da un partecipante o da un avversario che intercetta i dati e li ritrasmette, utilizzando

messaggi trasmessi precedentemente. Un attaccante memorizza i messaggi che transitano sulla rete e successivamente li trasmette all'agente che vuole ingannare. Nel caso in cui i messaggi vengono spediti all'agente che li ha generati, questo attacco viene chiamato *reflection attack*. Benché l'attaccante non possa decriptare il messaggio, può avvantaggiarsi ricevendo un servizio dall'agente al quale sta ripetendo il messaggio. Di conseguenza, occuparsi dell'autenticazione attraverso Internet richiede qualcosa di più complesso delle password cifrate. Bisogna avere un metodo di autenticazione dove i dati che passano attraverso la rete siano non riutilizzabili, in modo che un agente ostile non possa semplicemente bloccarli e rispedirli indietro. Il modo migliore per contrastare un replay attack è quello di utilizzare challenge per dimostrare la freschezza del messaggio. Ciò è fatto includendo un timestamp, un numero progressivo, o un numero casuale nel messaggio.

Denial of Service (DoS): l'agente ostile tenta di bloccare l'accesso ad un determinato servizio ad uno o più utenti. È importante notare che un *DoS* non è necessariamente un attacco fatto tramite una rete ed infatti il concetto si estende anche al deterioramento intenzionale delle prestazioni di una risorsa, compresa la distruzione fisica. Comunque gli attacchi *DoS* non mirano ad ottenere informazioni riservate, ma solamente a bloccare servizi di rete. I *DoS* possono presentarsi in varie forme e bloccare una grande quantità di servizi. Possiamo dividerli in tre categorie:

Consumo di risorse limitate e non rinnovabili: per far funzionare una rete di calcolatori e dare un servizio si ha bisogno di diverse risorse, come potenza di calcolo, spazio su disco, RAM e banda di rete. Questo tipo di attacco cerca di sovraccaricare o bloccare l'uso di queste risorse allo scopo di impedire il loro utilizzo. Vediamo ora i tipi di *DoS* di rete conosciuti finora.

Attacco SYN flood: l'attaccante manda costantemente richieste *SYN* all'host attaccato senza completare il *3-Way Handshake*[1], lasciando così la richiesta pendente. Nel frattempo il server alloca nella sua memoria una struttura che rimane finché non è passato un certo periodo di tempo. Se le richieste in questo tempo sono in numero sufficiente le strutture create dal kernel intaseranno completamente la memoria del sistema, bloccando così i servizi normalmente erogati.

Attacco tramite bugs di sistema: un attaccante utilizza programmi che sono in grado di bloccare un server su cui gira una particolare versione di sistema operativo o determinati programmi, sfruttando situazioni non considerate dal programmatore.

Attacco Smurf: l'attaccante manda dei *pacchetti spoofed*, cioè con il campo sender cambiato con l'indirizzo IP della vittima, con una richiesta *ICMP*

[1] Stabilire una connessione TCP richiede tipicamente lo scambio di tre pacchetti tra due macchine, chiamati *TCP 3-Way Handshake*. Un TCP client inizia una connessione con un server TCP spedendo un pacchetto "*SYN*" al server. Quando un pacchetto per la richiesta di connessione *SYN* è ricevuto su una porta di servizio TCP aperta, il server risponde con un pacchetto di accettazione della connessione "*SYN*/*ACK*". Quando il client riceve il pacchetto *SYN*/*ACK* spedito dal server per la connessione pendente, risponde con un pacchetto *ACK* e inizia la trasmissione dei dati.

echo ad un indirizzo di rete broadcast. A questo punto, la richiesta viene mandata a tutti i computer collegati alla sottorete di quel broadcast, i quali rispondono alla richiesta di *echo* utilizzando il campo sender, cioè l'IP della vittima.

Attacco Distributed Denial of Service (DDoS): un attaccante ha il controllo di più macchine configurate per attaccare la stessa rete, o macchina, contemporaneamente con attacchi *SYN flood* o *Smurf*.

Distruzione o modifica delle informazioni di configurazione: questo tipo di attacco si basa sul fatto che i servizi, che sono malconfigurati, non funzionano o funzionano male. Con questo principio un attaccante potrebbe entrare in un sistema tramite un baco di un programma del server che apre un accesso con privilegi di amministratore e cambiare i parametri di configurazione.

Distruzione fisica o alterazione dell'hardware: è doveroso considerare la distruzione o la manomissione fisica di un server come un *DoS* vero e proprio. Infatti se l'hardware della macchina subisce un deterioramento, i servizi che erogava non sono più disponibili.

Gli attacchi presentati non rappresentano tutti i possibili attacchi che un agente ostile può portare, ma raccolgono la maggior parte di attacchi studiati in letteratura. Poiché dimostrare la sicurezza di un protocollo non è un'operazione banale, diventa di fondamentale importanza l'utilizzo di tecniche formali che consentono l'analisi dei protocolli.

2.3 Definizione di Autenticazione

L'autenticazione è fondamentale per la progettazione di sistemi distribuiti sicuri e la sua importanza si rispecchia, anche, nell'enorme attenzione che ha ricevuto in letteratura. Sono centinaia i lavori che sono stati prodotti unitamente ad altrettanti protocolli proposti e implementati. La definizione di autenticazione vincola ovviamente le analisi sia formali che informali dei protocolli di autenticazione e un'attenzione particolare deve essere rivolta al significato di sicurezza in quanto è strettamente legato alla definizione che si stabilisce essere valida per i protocolli stessi.
L'autenticazione è il processo mediante il quale un agente di un sistema distribuito dimostra la propria identità. Ogni partecipante alla comunicazione, solitamente, condivide un segreto con un'entità fidata detta *authentication server* o *trust server*[2]. Dimostrando di possedere questo segreto, un'agente è in grado di provare la propria identità, si pensi, per esempio, all'impiego di password in un ambiente multi-user. Il processo di autenticazione è utilizzato in un sistema distribuito allo scopo di proteggere i partecipanti da eventuali attacchi portati da un'entità ostile che voglia impadronirsi di informazioni sensibili. In sistemi distribuiti, l'autenticazione è tipicamente garantita da protocolli, chiamati *protocolli di autenticazione*, i cui scopi sono garantire l'identità degli agenti e lo scambio di nuove chiavi segrete di sessione per future comunicazioni.

[2]Un trust server, come si intuisce dalla traduzione dall'inglese, è un server per il quale è garantita la correttezza delle operazioni effettuate; in altre parole, è una componente del sistema degna di fiducia.

Come abbiamo già osservato, esistono varie tipologie di attacco verso i protocolli crittografici di autenticazione, ma è difficile definire una classificazione precisa in quanto le modalità di attacco variano a seconda del tipo di rete, dalla struttura e dalla sequenza dei messaggi, dal numero di partecipanti alla comunicazione, dall'implementazione degli algoritmi crittografici e dalla definizione stessa di cosa si intende per autenticazione. Un attacco ad un protocollo di autenticazione deve essere considerato tale solo se viola la definizione di sicurezza che si formula durante l'ideazione del protocollo stesso. Ad esempio, se l'intento del protocollo di autenticazione è determinare l'identità di un solo partecipante alla comunicazione (*protocollo di autenticazione unilaterale*), tutti gli attacchi portati alla sicurezza dell'altro partecipante non possono essere considerati attacchi.
Le prossime sezioni presentano come diversi autori in letteratura abbiano catturato il significato di autenticazione, soffermandosi maggiormente sull'aspetto delle definizioni.

2.3.1 Doffie, Van Oorschot, Wiener: Matching Records

Doffie, Van Oorschot e Wiener [30] estendono l'idea di *matching protocol run* di Gird, Gopal e altri [10] al caso di protocolli che si basano sullo schema di codifica a chiave pubblica. Nel loro lavoro vengono trattati separatamente i concetti di autenticazione e di key exchange, per la convinzione di una differenza di intenti.
Il modello è ristretto al caso in cui la crittografia utilizza solo algoritmi asimmetrici. Assumendo che i meccanismi crittografici alla base del sistema non siano vulnerabili, si consentono attacchi solo alla struttura del protocollo stesso. L'agente ostile, inoltre, è in grado di vedere tutti i messaggi scambiati dalle parti, cancellarli, modificarli, iniziare nuove comunicazioni con altre parti e riutilizzare messaggi di sessioni passate.
Un'istanza di un protocollo di autenticazione è chiamata *run*, mentre l'insieme dei messaggi di una run è chiamato *record*. Un messaggio di un record ha un *matching* con un messaggio di un altro record se in quel record la lista dei messaggi è etichettata come *incoming*, nell'altro come *outgoing*, e tutti i campi dei messaggi rilevanti per l'autenticazione sono identici per entrambi i record.
Due record di una run hanno un *run match* se i loro messaggi sono suddivisi in un insieme di matching messages (ogni insieme contiene un messaggio per ogni record), se i messaggi originati da un partecipante appaiono nello stesso ordine in entrambi i record, e se tutti i messaggi originati da un altro partecipante appaiono nello stesso ordine in entrambi i record.
Nell'analisi viene dato un particolare accento alla definizione di *successful run*, *secure run* e *matching records of runs*. L'idea è considerare le run che hanno successo come sicure. D'altra parte è possibile che esista una run che non sia stata completata con successo senza però causare un buco nella sicurezza. In questo modo attacchi come *DoS* non sono da considerarsi attacchi in quanto non causano brecce nella sicurezza.

$$\textit{successful} \rightarrow \textit{secure}$$
$$\textit{insuccessful} \nrightarrow \textit{insecure}$$

L'idea è che se due parti accettano una l'identità dell'altra ed esiste un *matching records of runs*, allora si ha un protocollo sicuro di autenticazione.

Definizione 2.3.1 *Una particolare run di un protocollo è* insicura *se una sua parte coinvolta la esegue fedelmente, accetta l'identità di un'altra parte e al tempo dell'accettazione, l'altro record non ha un match con il record della prima parte.*

L'intento del nemico è di operare in modo che il protocollo risulti insicuro, mentre l'intento degli ideatori del protocollo è rendere l'attacco del nemico computazionalmente improbabile in tutte le istanze. Diamo ora la definizione inversa, cioè di un protocollo sicuro di mutua autenticazione.

Definizione 2.3.2 *Un protocollo è* sicuro *se all'istante in cui un agente A accetti l'identità di un agente B, il record dell'agente B ha un parziale o totale run match con il record dell'agente A.*

Le precedenti definizioni non risultano essere particolarmente utili per decidere se un protocollo sia sicuro o meno, ma se applicate, possono aiutare a determinare se un potenziale attacco sia un attacco reale.

2.3.2 Rogaway, Bellare: Matching Conversation

Rogaway e Bellare hanno per la prima volta proposto la definizione di mutua autenticazione e distribuzione delle chiavi in [7]. La loro ricerca si basa su quella di Doffie, Van Oorschot e Wiener [30] che per primi hanno introdotto il concetto di *matching records of runs* nell'ambito di un sistema a chiave pubblica. Una run è un'istanza di una sessione del protocollo; se esiste una sorta di matching tra istanze dello stesso protocollo, allora esiste una *matching protocol run*. Rogaway e Bellare hanno il merito di aver adattato l'idea di matching protocol run nell'ambito della crittografia moderna costruendo un sistema di definizioni basato sul concetto di *probable security* [11, 43]. Questa astrazione ha permesso di trattare i problemi di autenticazione e distribuzione delle chiavi al medesimo livello di primitive come lo schema di cifratura, i generatori pseudorandom e le firme digitali.
Rogaway e Bellare assumono che le comunicazioni tra le parti avvengano sotto il controllo dell'avversario e che questo sia in grado di leggere i loro messaggi, crearne propri, modificarli prima che giungano a destinazione, e replicarli. Inoltre l'avversario può iniziare nuove istanze con qualsiasi partecipante alle comunicazioni, gestendo più sessioni contemporaneamente. Ciascuna parte è modellata da una collezione infinita di oracoli con i quali l'avversario riesce ad interagire. Questi oracoli interagiscono a loro volta solamente con l'avversario e non tra di loro o con altre entità del sistema.
In presenza di un avversario così potente è difficile capire come una parte possa essere convinta di aver intrapreso una comunicazione con un preciso partner. Il protocollo è sicuro se l'unico modo, nel quale un avversario possa convincere una parte ad accettare, è consegnare i messaggi fedelmente. Per formalizzare questa semplice idea si ricorre alla nozione di *matching conversation*. Una conversazione è costituita da una sequenza di triple della forma $(\tau_i,\alpha_i,\beta_i)$ dove τ_i rappresenta l'istante di tempo, α_i il dato ricevuto e β_i il dato spedito. Si dice che una conversazione ha un matching con una conversazione di un altro oracolo se le due sequenze seguono un ordine fissato. La definizione di mutua autenticazione si basa nel mandare in esecuzione un avversario E. Una volta che E ha terminato, ogni oracolo ha avuto

una certa conversazione con E, ed è giunto a una certa decisione. La determinazione della sicurezza di un protocollo di mutua autenticazione dipende da queste conversazioni e decisioni. Le esecuzioni dell'esperimento possono essere classificate *buone* o *cattive* a seconda che l'avversario sia riuscito o meno a sovvertire una particolare run.

Definizione 2.3.3 *Un protocollo di mutua autenticazione è* sicuro *se, per ogni avversario E con complessità di calcolo polinonomiale,*

1. *due oracoli hanno matching conversation, allora entrambi accettano,*
2. *la probabilità che un oracolo accetti e nell'altro non ci sia matching conversation, è trascurabile.*

2.3.3 Lowe: Agreement

Lowe [58] suggerisce che i requisiti, appropriati per i protocolli di autenticazione, dipendono direttamente dagli scopi dei protocolli stessi. Identifica alcune possibili definizioni di autenticazione, incentrando il suo lavoro sul concetto di *agreement* e tutte le definizioni proposte sono giustificate attraverso l'utilizzo di un formalismo quale l'algebra di processo CSP.
Un protocollo di autenticazione è definito come un oggetto che permette l'autenticazione di un *responder B* a un *initiator A*, possibilmente grazie all'aiuto di una terza entità chiamata *server*. Si delineano così una serie di ruoli che un agente partecipante alla comunicazione può adottare; difatti un agente può comportarsi come un initiator di alcune run, e responder di altre. I protocolli sono costruiti in modo da terminare con successo anche in presenza di un agente ostile che ha il completo controllo delle comunicazioni e che è in grado di intercettare messaggi e introdurne di nuovi sfruttando le informazioni ottenute dai messaggi visti.
Lowe ha posto alcune domande sul significato di autenticazione, e cercando di dare una risposta, ha delineato la struttura su cui basare la costruzione del proprio sistema di definizioni.

1. Se A ha completato una run apparentemente con B, che cosa può dedurre A rispetto lo stato di B?
2. Può A dedurre che B recentemente sia vivo?
3. Può A dedurre che B abbia recentemente eseguito lo stesso protocollo di A?
4. Può A dedurre che B pensi che stia eseguendo il protocollo con A?
5. Può A assumere che ci sia una relazione uno-a-uno tra le sue run e le run di B?

Un protocollo di autenticazione deve innanzitutto assicurare ad un agente A l'identità di un agente B con il quale ha iniziato una sessione del protocollo. Inoltre, A deve essere sicuro che B pensi di aver iniziato la sessione con A. Differenti specifiche di protocolli di autenticazione possono avere validi intenti, quindi è importante formulare una definizione appropriata per ogni esigenza. Sono stati individuati quattro significati ragionevoli del concetto di autenticazione.

Definizione 2.3.4 ***(Aliveness)*** *Un protocollo garantisce all'initiator A l'esistenza di un altro agente B se, una volta che A completa una run apparentemente con B, B ha precedentemente eseguito il protocollo.*

Questa definizione è la più debole tra quelle proposte. Infatti, lo stesso autore avverte che *B* può anche non credere di avere una sessione aperta con *A* e che può anche non aver iniziato la sessione recentemente.

Definizione 2.3.5 ***(Weak Agreement)*** *Un protocollo garantisce ad un initiator A* weak agreement *con un altro agente B se, una volta che A completi una run apparentemente con B, B ha eseguito una sessione del protocollo apparentemente con A.*

Definizione 2.3.6 ***(Non-Injective Agreement)*** *Un protocollo garantisce* non-injective agreement *con un responder B su un insieme di dati d_s, se, una volta che A completi una run apparentemente con B, B ha precedentemente eseguito il protocollo con A, B è un responder in questa run e i due agenti sono d'accordo nei valori di tutte le variabili in d_s.*

Questa definizione non garantisce che ci sia una corrispondenza uno a uno tra le run di *A* e le run di *B*. L'agente *A* potrebbe credere di aver completato due run, quando *B* ha preso parte ad una sola run.

Definizione 2.3.7 ***(Injective Agreement)*** *Un protocollo garantisce ad un initiator A* injective agreement *con un responder B su un insieme di dati d_s, se, una volta che A completi una run apparentemente con B, B ha precedentemente eseguito il protocollo, apparentemente con A, e B è un responder in quella run, e i due agenti sono d'accordo sui valori di tutte le variabili in d_s e tale run di A corrisponde unicamennte alla run di B.*

L'injective agreement, o *full agreement*, viene considerata la definizione più utile, con essa infatti i due agenti sono d'accordo su tutti i principali aspetti della run del protocollo.

2.3.4 Focardi: Non Deducibility on Composition

Negli ultimi anni sono state individuate molte proprietà di sicurezza per protocolli crittografici come *secrecy* (le informazioni riservate devono essere disponibili solo agli agenti onesti della comunicazione), *authentication* (la capacità di identificare gli altri agenti che interagiscono in una comunicazione), *integrity* (assicurare l'integrità del contenuto dei messaggi), *fairness* (nessun agente può ottenere un vantaggio terminando prima il protocollo). Focardi [36, 37] propone un approccio per uniformare la definizione e l'analisi di alcune proprietà di sicurezza basandosi sull'idea che una proprietà di sicurezza dovrebbe essere soddisfatta anche in presenza di un ambiente ostile.
Focardi e altri [33, 34, 35] notano come molte proprietà di sicurezza possono essere viste come un'istanza specifica di una proprietà generale che chiama *Non Deducibility on Composition* (NDC). NDC è stata proposta come una generalizzazione dell'idea classica di *non-interference* per sistemi non deterministici. La non-interference cerca di capire se un gruppo fissato di processi *G* è in grado di "interferire" in qualche modo con un altro gruppo G',

cioè se le azioni dei processi in G hanno qualche effetto sull'esecuzione dei processi in G'. Qualche volta le proprietà di non-interference sono chiamate anche proprietà di *Information Flow*, quindi un'interazione di G con G' può essere vista come un flusso di informazioni dal primo gruppo al secondo. Una classica applicazione di queste proprietà è la *multilevel security* [90] dove H (high level) rappresenta un insieme di processi a cui è proibito spedire qualsiasi dato a L (low level), cioè interagire con L. NDC è così applicata per la verifica di protocolli di sicurezza. Richiedendo che le azioni di un attaccante non modifichino in alcun modo il comportamento osservabile del protocollo, Focardi, Gorrieri e Martinelli [34] hanno mostrato come NDC sia la proprietà piu forte definibile per protocolli crittografici.
Focardi e Martinelli [35] propongono uno schema generale per la definizione delle proprietà di sicurezza, che chiamano *Generalized NDC* (GNCD), che ha la seguente forma

$$E \text{ soddisfa } P^{\alpha}_{\triangleleft} \quad \text{sse} \quad \forall x \in Env : E||X \triangleleft \alpha(E)$$

La proprietà generale $P^{\alpha}_{\triangleleft}$ richiede che il sistema soddisfi una specifica $\alpha(E)$ quando è composto con qualsiasi ambiente X. La proprietà è parametrizzata rispetto a $\alpha(E)$ e $\triangleleft$ che sono istanziate per ottenere diverse proprietà di sicurezza. In particolare, $\alpha(E)$ è una funzione tra processi che specifica il "corretto" comportamento di E; $\triangleleft$ è una relazione tra processi che rappresenta la nozione di "osservabile". Così, con $E||X \triangleleft \alpha(E)$ si controlla se il processo E mostra un comportamento corretto anche in presenza di X. L'idea principale è la seguente: un sistema E è $\text{GNDC}^{\alpha}_{\triangleleft}$ se e solo se, per ogni ambiente X, la composizione del sistema con X soddisfa una specifica $\alpha(E)$. In pratica, $\text{GNDC}^{\alpha}_{\triangleleft}$ è soddisfatta (rispetto alla relazione $\triangleleft$) anche quando il sistema è composto con qualsiasi ambiente ostile. Allora la proprietà NDC può essere riscritta come $\text{GNDC}^{E\backslash C}_{\approx_{trace}}$ dove $\approx_{trace}$ è l'equivalenza tra tracce e $E\backslash C$ è il sistema in cui non è permessa alcuna attività ad alto livello.
Focardi e Martinelli cercano di sfruttare alcune idee fondamentali di proposte esistenti piuttosto che definire un approccio completamente nuovo. Pensano che uno schema uniforme per la definizione delle proprietà di sicurezza porti molti vantaggi e consenta di studiare la relazione tra differenti proprietà di sicurezza.
Focardi e Martinelli [35, 36] giudicano interessante la nozione di *message authentication* proposta da Abadi e Gordon [1]. L'idea base è considerare un protocollo $P(M)$, che cerca di trasmettere un messaggio M da un agente A ad un agente B. L'autenticazione del messaggio M è controllata verificando se $P(M)$ è equivalente a $P_{spec}(M)$ in cui M è sempre consegnato correttamente. In $P_{spec}(M)$ il ricevente B conosce sempre M e qualsiasi cosa succede sul canale di comunicazione, B continua esattamente la sua esecuzione come se avesse ricevuto il messaggio corretto M. In altre parole, $P_{spec}(M)$ rappresenta la situazione dove M è sempre ricevuto e nessun agente ostile è in grado di sostituirlo con un messaggio diverso. Se $P(M)$ è equivalente a $P_{spec}(M)$ allora anche $P(M)$ è chiaramente in grado di evitare qualsiasi attacco possibile. Il linguaggio utilizzato in [1] è lo spi-calculus e la *may-testing equivalence* [28] è utilizzata per controllare se $P(M)$ è equivalente a $P_{spec}(M)$ rispetto a ogni possibile interazione con l'ambiente. La definizione di autenticazione è data come segue:

Definizione 2.3.8 *$P(M)$ garantisce autenticazione se e solo se, per ogni M, abbiamo che $P(M)$ è may-testing equivalent a $P_{spec}(M)$.*

Focardi e Martinelli, riscrivendo questa definizione nel loro modello, ottengono:

Definizione 2.3.9 *Sia $P(M)$ un protocollo dove gli agenti comunicano su un insieme di canali C. $P(M)$ garantisce autenticazione se e solo se, per ogni M, abbiamo che $P(M) \in NDC$.*

Focardi e Martinelli mostrano anche come la nozione di *agreement* proposta da Lowe [58] può essere rappresentata nel loro sistema di definizioni. La proprietà di agreement stabilisce che ogni azione che rappresenta l'esecuzione del protocollo ne abbia una che che la complementa. Per esempio, si consideri un'azione $commit\_res(B,A,d)$ che rappresenta una corretta terminazione di B, come responder, che è convinto di comunicare con A e concorda sul dato d. Inoltre, esiste un'azione $running\_ini(A,B,d)$ che rappresenta il fatto che A sta eseguendo il protocollo come initiator, apparentemente con B e con in dato d. Se ci sono queste due azioni nel protocollo, la proprietà di agreement richiede che quando B esegue $commit\_res(B,A,d)$ allora A ha precedentemente eseguito $running\_ini(A,B,d)$. Questo significa che ogni volta che B completa il protocollo con A è convinto che i dati rilevanti siano quelli rappresentati da d, allora A deve aver eseguito il protocollo con B usando esattamente i dati in d.
La funzione $\alpha(E)$ può essere definita come segue:

$$\begin{aligned} E' &= Top_{trace}^{Sort(E)\setminus\{running\_ini,commit\_res\}} \\ E''(x,y) &= \textstyle\sum_{d\in D} \overline{running\_ini}(x,y,d) \,.\, \overline{commit\_res}(x,y,d) \\ \alpha_{Agree}(E) &= E' \| E''(A,B) \end{aligned}$$

Dato E, $\alpha(E)$ rappresenta il sistema più generale che soddisfa la proprietà di agreement. Se vogliamo analizzare più sessioni, è sufficiente estendere α considerando più copie del processo $E''(A,B)$ in parallelo. Per esempio, per n sessioni consideriamo

$$\alpha_{Agree}(E) = E' \| \underbrace{E''(A,B) \| \cdots \| E''(A,B)}_{n}$$

Si vuole che anche in presenza di un processo ostile, E non sia in grado di eseguire le tracce che non siano in $\alpha(E)$, cioè si richiede che $E\|_C X \leq_{trace} \alpha(E)$ dove $\leq_{trace}$ rappresenta un preordine tra tracce.

Definizione 2.3.10 *E soddisfa agreement se e solo se E è $GNDC_{\leq_{trace}}^{\alpha_{Agree}(E)}$.*

Focardi e Martinelli traducono nel loro sistema di definizioni anche la nozione di *message-oriented authentication* e quella di *non-repudiaton* proposte da Schneider [84].

2.4 Classificazione di Protocolli di Autenticazione

Ogni trasmissione che si attiva in un sistema di comunicazione ha lo scopo di portare a termine una funzione. Per la sicurezza delle comunicazioni assumono la massima importanza i seguenti obiettivi:

- **Key Exchange**: stabilire una nuova chiave segreta tra due o più parti che intendono attivare una sessione di comunicazione, cioè dare la sicurezza ad un agente che nessun agente ostile possa venire a conoscenza dei suoi segreti.

- **Authentication**: consentire ad ogni parte coinvolta nelle comunicazioni tramite il protocollo di ottenere sufficienti garanzie sull'identità delle altre parti, cioè l'agente, che si deve autenticare, deve essere sicuro di parlare con l'agente desiderato.

Protocolli crittografici sono protocolli di comunicazione che usano algoritmi crittografici il cui scopo principale è realizzare obiettivi di sicurezza. Tra tali protocolli distinguiamo quelli che si prefiggono di garantire la scambio di chiavi e l'autenticazione, chiamati *protocolli di autenticazione*. Il processo di autenticazione è, infatti, utilizzato in un sistema distribuito allo scopo di proteggere i partecipanti da eventuali attacchi portati da un'entità ostile che si voglia impadronire di informazioni sensibili. I protocolli di autenticazione vengono suddivisi, in base alle proprietà che si propongono di garantire, nelle seguenti tipologie:

- **Key Exchange Protocol**: permette a due o più parti di scambiarsi una chiave segreta in modo da attivare una nuova sessione di comunicazione.

- **Authentication Protocol**: consente ai partecipanti coinvolti nella comunicazione tramite il protocollo di ottenere sufficienti garanzie sull'identità delle altre parti.

- **Authentication and Key Exchange Protocol**: le parti coinvolte nella comunicazione attraverso il protocollo sono in grado di scambiarsi una chiave di sessione e nello stesso tempo ottenere garanzie sull'identità dei partecipanti.

La verifica della sicurezza di tali protocolli risulta fondamentale per l'analisi della correttezza di tutti i sistemi e protocolli più complessi che si basano su di essi. La necessità di definire una verifica di protocolli è dovuta al fatto che la progettazione degli stessi è normalmente incline a errori. Infatti sono molti i protocolli che, dopo essere stati pubblicati, sono risultati non corretti.

Capitolo 3

Tecniche di Analisi per Protocolli

I protocolli crittografici di comunicazione utilizzano algoritmi di codifica e di decodifica dei dati per garantire la sicurezza delle trasmissioni. L'utilizzo di un algoritmo crittografico efficiente e funzionante nei passi del protocollo non è però sufficiente a garantire la sicurezza delle comunicazioni. Infatti sono la combinazione dei dati e la sequenza dei messaggi trasmessi che esprimono le proprietà di un protocollo crittografico.
Questi motivi hanno spinto molti gruppi di ricerca a studiare il modo per individuare sessioni sicure tra i partecipanti di una comunicazione. Il fine è ideare tecniche idonee a dimostrare la sicurezza del protocollo a priori. Sebbene molte di queste tecniche possono essere applicate a tutti i protocolli crittografici, la ricerca si è soffermata in modo particolare sullo studio di protocolli che si prefiggono di garantire l'autenticazione e lo scambio di chiavi.
Questo capitolo prende in esame le tecniche presenti in letteratura finalizzate ad analizzare i protocolli crittografici, evidenziando i maggiori problemi a loro associati. Inoltre presenta un modello per formalizzare le capacità degli agenti ostili adottato da molte delle tecniche discusse, che è alla base anche della nostra analisi. Tale modello è chiamato *modello di Dolev-Yao* [31] dal nome degli autori che lo hanno definito per la prima volta.

3.1 Metodi Informali

Il problema della sicurezza dei protocolli di comunicazione è nato insieme all'utilizzo degli strumenti crittografici nella trasmissione dei dati. Esistono molti esempi di analisi informali e suggerimenti utili per la progettazione di un buon protocollo crittografico e per la verifica di quelli esistenti. Negli ultimi anni le tecniche informali sono state abbandonate a favore dei metodi di analisi formali. La progettazione di nuovi protocolli sono invece ancora affidate all'esperienza e al buon senso del progettista. Abadi e Needham [2] hanno raccolto le esperienze accumulate negli anni nel settore, fornendo una sorta di linee guida in grado di agevolare la progettazione di un sistema crittografico "sicuro". Questi principi non sono sufficienti per garantire la correttezza dei protocolli, ma seguendoli si possono evitare alcuni attacchi conosciuti e, inoltre, sono indipendenti dai metodi di analisi formali. Le regole fondamentali che devono guidare la progettazione di protocolli sicuri sono due: la prima riguarda il contenuto dei messaggi, la seconda le circostanze in cui è utilizzato.

- Ogni messaggio deve specificare esattamente il suo significato, e la sua interpretazione deve dipendere solamente dal suo contenuto.

- Le condizioni che motivano l'invio del messaggio devono essere chiare, in modo da decidere quando queste condizioni possono essere accettate oppure no.

Da questi principi fondamentali ne derivano altri a seconda del contesto di applicazione:

1. Se l'identità di un partecipante alla comunicazione è essenziale per il significato del messaggio, è prudente menzionare esplicitamente il nome del partecipante all'interno del messaggio.

2. I motivi che guidano l'utilizzo della crittografia devono essere chiari. Poiché l'operazione di codifica risulta computazionalmente gravosa, si corre il rischio di causare ridondanza. La crittografia non è sinonimo di sicurezza e, quindi, un suo utilizzo improprio può portare a gravi errori.

3. Se si utilizzano strumenti di firma elettronica, occorre tener presente che la firma di un dato già codificato non comporta la conoscenza del dato stesso da parte del firmatario. D'altra parte la codifica di un dato firmato implica la conoscenza del dato stesso.

4. Le proprietà di utilizzo dei nonce[1] devono essere chiare. Le assunzioni che garantiscono la successione temporale dei messaggi potrebbero essere non valide per garantire l'associazione tra i diversi messaggi durante una stessa sessione del protocollo.

5. Se nello scambio dei messaggi vengono utilizzate quantità prevedibili, allora queste dovrebbero essere protette in modo tale che un intruso non sia in grado di simulare la corretta successione dei messaggi.

6. Se è utilizzato un timestamp[2] per garantire la *freshness*, allora la differenza fra gli orologi locali dei vari sistemi deve essere più piccola dell'intervallo di tempo entro i quali i messaggi sono ritenuti accettabili. Inoltre il meccanismo temporale del sistema deve far parte di una componente fidata.

7. L'utilizzo recente di una chiave non è necessariamente indice di sicurezza o di freshness della chiave stessa che potrebbe essere compromessa o molto vecchia.

[1] Un nonce è un valore intero che viene introdotto in un messaggio per dimostrarne la sua freschezza. Viene generato su richiesta ed è utilizzabile una volta sola. Ad esempio, può essere generato utilizzando la data e l'ora corrente (timestamp). Un nonce viene usato anche per sconfiggere playback attack nei protocolli di autenticazione. Un agente genera a caso un nonce e lo trasmette all'altro agente. Il ricevente lo codifica usando la chiave segreta accordata e lo rimanda al mittente. Poiché il nonce è stato generato a caso dal mittente, il ricevente non può conoscere in anticipo il nonce che il mittente genererà. Questo strumento viene utilizzato, per esempio, per sconfigge attacchi di playback.

[2] Un timestamp è un dato particolare utilizzato per garantire la freshness delle informazioni; questo indica la progresssione temporale in qualche formato, ad esempio potrebbe essere formato da data-ora-minuti-secondi-millisecondi.

8. Se il significato di un messaggio dipende dallo schema utilizzato per rappresentare le informazioni, allora deve essere possibile dedurre il tipo di schema che caratterizza ogni messaggio. Poiché gli schemi dipendono dai protocolli, deve essere possibile determinare l'appartenenza di ogni messaggio ad una determinata sessione di un protocollo ed il numero d'ordine del messaggio all'interno della sessione stessa.

9. Le relazioni di fiducia accettate nel progetto del protocollo devono essere esplicitate e motivate.

Questi suggerimenti possono essere sia inseriti nelle analisi informali dei protocolli sia utilizzati per realizzarne di nuovi senza escludere l'impegno di tecniche formali.

3.2 Metodi Formali

Come è stato già detto in precedenza, i protocolli crittografici sono soggetti ad errori. Negli ultimi anni sono state sviluppate diverse tecniche per l'analisi dei protocolli crittografici. Queste sono state classificate [61] in funzione dei diversi approcci adottati dagli analisti. Associamo ad ogni approccio il tipo di strumento utilizzato per modellare l'analisi:

1. **Tecniche d'applicazione generale** sono gli approcci che adottano linguaggi di specifica e strumenti di verifica non progettati espressamente per l'analisi di protocolli crittografici.

2. **Tecniche dei sistemi esperti** si basano su sistemi esperti per l'analisi di diversi "scenari" di esecuzione di un sistema dai quali si può trarre delle conclusioni sulla sicurezza del protocollo studiato.

3. **Tecniche di logiche modali** si basano su modelli logici specifici, sviluppati appositamente per l'analisi delle conoscenze e su predicati che le parti comunicanti derivano attraverso le interazioni previste dal protocollo.

4. **Tecniche di riscrittura dei termini** si basano sullo sviluppo di un metodo formale basato sulle proprietà algebriche delle tecniche di riscrittura dei termini dei sistemi crittografici.

5. **Tecniche di algebre di processi** sono gli approcci che adottano algebre di processi per la descrizione dei sistemi paralleli.

6. **Tecniche della teoria della complessità** utilizzano strumenti probabilistici. L'intento è di provare che un sistema di comunicazione sia una soluzione *provably secure* ai problemi di sicurezza considerati.

7. **Tecniche dell'invariante** sono gli approcci che utilizzano strumenti derivanti dalle tecniche delle algebre di processo. Per la verifica della sicurezza del protocollo si esprimono le proprietà in termini di invarianti del sistema.

3.2.1 Tecniche di Applicazione Generale

Le tecniche di applicazione generale modellano e analizzano i protocolli crittografici usando linguaggi di specifica e strumenti di verifica non espressamente progettati per l'analisi di protocolli di questo tipo. Gli approcci più diffusi sono quelli che adottano macchine a stati finiti per la descrizione del protocollo da verificare e tecniche di analisi del raggiungimento degli stati [94] per la verifica del protocollo descritto. Le proprietà di sicurezza, che devono essere preservate dal protocollo, vengono rappresentate sotto forma di teoremi che esprimono l'impossibilità di transitare a determinati stati; la verifica di tali proprietà si basa sulle prove di tali teoremi, che vengono generate automaticamente dal sistema di verifica.
L'utilizzo dei metodi di applicazione generale ha il vantaggio di basarsi su tecniche ben definite e largamente utilizzate. L'analisi di un protocollo, effettuata tramite questo tipo di tecniche, è in grado di verificare se il protocollo è corretto nel rispetto della specifica fornita dall'analista; ciò nonostante un protocollo che funziona correttamente non è necessariamente sicuro [87]. Dato che le tecniche di questo tipo non utilizzano strumenti specifici per la descrizione delle proprietà di sicurezza, molte caratteristiche degli attacchi ai protocolli crittografici non possono essere descritte.
Concludiamo, quindi, che l'analisi di un protocollo crittografico tramite questa tipologia di analisi può essere utile per individuare nel protocollo gli errori di sicurezza noti, ma non riesce ad individuare eventuali nuovi tipi di attacco.

3.2.2 Tecniche dei Sistemi Esperti

L'uso di tecniche basate sui sistemi esperti si basa sullo sviluppo e sull'investigazione di diversi possibili scenari di esecuzione del protocollo. Gli approcci più diffusi sono basati su macchine a stati finiti [52]. La verifica consiste nel determinare se, a partire da uno stato considerato insicuro, è possibile individuare una sequenza di transizioni che lo collega ad uno degli stati iniziali del sistema.
Un esempio di sistema esperto applicato all'analisi della sicurezza è l'Interrogator [62]. La verifica viene effettuata fornendo in input al sistema la specifica del protocollo in analisi e un insieme di stati obiettivo che rappresentano la sua rottura. Il sistema verifica se esiste una sequenza di transizioni che porta dallo stato iniziale ad uno degli stati obiettivo e, in tal caso, fornisce la traccia dell'attacco.
Come succedeva per le tecniche di applicazione generale, gli stati obiettivo, che rappresentano la rottura del protocollo, devono essere conosciuti a priori. L'analista, quindi, deve conoscere in anticipo i potenziali attacchi da scoprire, rendendo impossibile l'individuazione di nuovi tipi di attacco.
Affermiamo perciò che i sistemi esperti non sono in grado di dimostrare la sicurezza di un protocollo, ma si limitano a verificare se e come un determinato tipo di attacco può essere eseguito con successo.

3.2.3 Tecniche delle Logiche Modali

In questo approccio l'analisi dei protocolli crittografici utilizza modelli logici formali sviluppati appositamente per l'analisi di *knowledge* e *belief* che le parti comunicanti derivano attraverso le interazioni previste dal protocollo. Con il termine knowledge si intende l'insieme dei dati e delle proprietà che un partecipante alle comunicazioni impara dallo scambio dei messaggi previsti dal protocollo, mentre con il termine belief si intende l'insieme dei dati e delle proprietà che un agente assume come veri in funzione dei messaggi scambiati. Entrambi questi concetti sono adatti per descrivere l'evoluzione degli stati dei partecipanti alle comunicazioni previste dal protocollo e la loro assiomatizzazione rappresenta il fulcro del modello di ogni logica modale.
La logica introdotta da Burrows, Abadi e Needham [22], chiamata anche *BAN logic*, è la più utilizzata per l'analisi di protocolli di autenticazione. La BAN logic assume che l'autenticazione sia una funzione dell'integrità e della freschezza della comunicazione, e impiega regole logiche per tracciare entrambi questi attributi all'interno del protocollo. L'analisi di un protocollo tramite la BAN logic prevede la traduzione del protocollo studiato in espressioni formali e la determinazione delle belief iniziali di ogni partecipante alle comunicazioni. Le regole di inferenza della BAN logic vengono applicate sulle assunzioni iniziali e sulla specifica formale del protocollo per ricavare l'evoluzione delle belief del sistema. L'analisi di un protocollo, attraverso l'applicazione della BAN logic, si può riassumere nei seguenti passi:

Idealizzazione del protocollo: la specifica informale del protocollo viene convertita in forma idealizzata e può essere manipolata dalla logica.

Assunzione dello stato iniziale: vengono stabilite le belief iniziali e le proprietà che caratterizzano lo stato iniziale del sistema.

Determinazione degli effetti di un passo del protocollo: vengono associate delle formule logiche ad ogni passo del protocollo per modellare l'evoluzione dello stato del sistema dopo ogni passo del protocollo stesso.

Applicazione dei postulati logici: i postulati logici vengono applicati alle assunzioni iniziali del sistema e alla specifica formale del protocollo al fine di derivare l'evoluzione delle belief dei partecipanti al protocollo.

La BAN logic, quindi, non fornisce una dimostrazione formale di sicurezza, ma è uno strumento per scoprire la presenza di eventuali buchi nella sicurezza del protocollo in analisi. La BAN logic presenta però punti deboli che possono essere riassunti dalle seguenti considerazioni:

- La logica non prevede un meccanismo in grado di rappresentare esplicitamente le belief non derivabili dalle parti del sistema analizzato.
- La logica è orientata all'analisi specifica delle belief e non alle proprietà di sicurezza.

Nonostante le critiche [71] e le numerose altre logiche introdotte [44, 60, 89], la BAN logic ha contribuito a scoprire buchi nella sicurezza di alcuni tra i più conosciuti protocolli di

comunicazione come nel protocollo di Needham-Schroeder [70] e CCITT X.509. Questa logica ha scoperto anche molte ridondanze in altri protocolli tra cui Yahalom, Needham-Schroeder e Kerberos.

3.2.4 Tecniche di Riscrittura dei Termini

Questo approccio si basa sulle proprietà algebriche delle tecniche di riscrittura dei termini. Gli strumenti di verifica più recenti di questo approccio forniscono metodi automatici per l'analisi formale dei protocolli e hanno permesso agli utenti di interrogare i sistemi per attacchi conosciuti.
L'analisi consiste nella verifica del raggiungimento di certi stati, cioè dimostra che nessun stato insicuro può essere raggiunto durante la computazione del sistema. In un certo senso questo tipo di verifica è molto simile alle tecniche basate sui sistemi esperti, che consistono nel dimostrare, partendo da uno stato insicuro, l'inesistenza di una sequenza di transizioni che lo lega ad uno degli stati iniziali. Elenchiamo le fasi principali che caratterizzano questo approccio:

- Definire le regole di transizione corrispondenti alle azioni degli agenti che partecipano al protocollo.
- Descrivere le operazioni che caratterizzano la codifica e la decodifica delle informazioni.
- Descrivere gli atomi utilizzati per la costruzione dei messaggi del protocollo.
- Introdurre le regole di riduzione tramite riscrittura.
- Verificare le specifiche del protocollo.

Alcune tecniche [61] si basano sulla considerazione che un protocollo crittografico può essere studiato come un insieme di regole per generare le parole di un linguaggio formale. Il sistema analizzato viene così modellato attraverso l'associazione dei passi del protocollo in analisi e degli eventi che causano la rottura delle proprietà di sicurezza. La verifica delle proprietà consiste nel dimostrare l'incapacità delle entità ostili al sistema di derivare le parole che rompono il protocollo.
Altre tecniche di questo approccio analizzano formalmente le proprietà probabilistiche dei protocolli crittografici [91]. Il modello utilizzato da queste tecniche prevede che, ad ogni transizione, venga associata una probabilità di esecuzione in modo tale che per ogni stato raggiungibile dal sistema possa essere calcolata la corrispondente proprietà di esecuzione.
La verifica di protocolli crittografici richiede la descrizione da parte dell'analista degli stati del sistema ritenuti insicuri. Concludiamo affermando che tali tecniche non sono in grado di garantire la sicurezza dei sistemi analizzati contro attacchi che non sono stati modellati durante la fase di progettazione dell'analisi. Inoltre le tecniche che si basano sulla riscrittura dei termini hanno una complessità di computazione elevata rendendo scarso il loro utilizzo pratico.

3.2.5 Tecniche delle Algebre di Processo

L'utilizzo delle algebre di processo permette di modellare il protocollo in analisi attraverso algebre volte alla descrizione dei sistemi paralleli. Lo scopo principale di questo tipo di analisi consiste nel verificare le proprietà dei protocolli in esame, accertando la corrispondenza fra descrizione e specifica del sistema. La strategia di verifica consiste nel confrontare la descrizione algebrica delle proprietà insite nella specifica informale del protocollo con la descrizione algebrica dell'implementazione reale del sistema in analisi. Il confronto viene effettuato attraverso le equivalenze definite sui processi dall'algebra adottata.
Alcune applicazioni di questo tipo di analisi fanno uso di un'algebra di processo conosciuta con il nome di π-calculus [65, 66, 67, 82]. Questa consente di descrivere processi concorrenti focalizzando l'attenzione sulla comunicazione attraverso canali. Vengono sfruttati operatori di restrizione e di estrusione dello scoping come modello formale per indicare la condivisione dei segreti, come le chiavi crittografiche. Le regole di scoping garantiscono in tal modo che un agente ostile non possa accedere ad un canale che non gli sia reso esplicitamente disponibile.
Abadi e Gordon [1] hanno introdotto un'estensione del π-calculus progettata per la descrizione e l'analisi di protocolli crittografici. Questa estensione, chiamata spi-calculus, oltre alle primitive del π-calculus, prende in considerazione primitive specifiche per la descrizione delle operazioni crittografiche di codifica e decodifica dei dati.
L'analisi di un protocollo tramite π-calculus o spi-calculus prevede la traduzione dei passi previsti dal protocollo in forma algebrica e la verifica di alcune proprietà tradotte dalle proprietà di sicurezza fissate per quel protocollo.
Un esempio di analisi basata su questo approccio è la Control Flow Analysis (CFA) [12, 13, 14]. Questa determina approssimazioni computabili e sicure di un insieme di valori che gli oggetti di un programma possono assumere dinamicamente durante l'esecuzione del programma stesso. La CFA è basata su Flow Logic e usa il π-calculus per modellare i protocolli come processi concorrenti. Una volta che i sistemi sono specificati come espressioni del calcolo, l'analisi calcola un'approssimazione del flusso delle informazioni, che viene chiamata soluzione. Per studiare la sicurezza dei sistemi i canali sono divisi in segreti e pubblici e la sicurezza è rotta se informazioni segrete sono comunicate su canali pubblici. In sintesi viene fornita una verifica statica per controllare se un processo non abbia buchi. La verifica delle proprietà di sicurezza, su un comportamento dinamico del processo avviene stabilendo se la soluzione trovata possa essere utilizzata per verificare la corrispondente proprietà statica.
Ci sono molti lavori, come quello di Zdancewic e Mayers [100], di Pottier e Simonet [75] e di altri [12, 33, 41], basati su questo tipo di tecniche, in cui la proprietà di sicurezza è la *non-interference*, che stabilisce che informazioni ad alto livello non devono essere osservabili da computazioni a livello inferiore di sicurezza.
Un esempio di analisi che si propone di verificare quest'ultima proprietà è quella di Honda e Yoshida [50, 51]. Questi utilizzano una versione asincrona del π-calculus tipato per rappresentare il flusso di informazioni, e basano il loro lavoro sulla teoria dei giochi introdotta da Hyland e Ong [54]. L'analisi verte su una nozione comportamentale di tipi che cattura la causalità delle azioni per assicurare la sicurezza del flusso di informazioni in diversi comportamenti interativi. Vengono presentate due discipline di tipi per il π-calculus

che caratterizzano precisamente due classi di comportamenti funzionali di ordine superiore sequenziale, che sono chiamate *affine* [9] e *linear* [98, 99]. Con il termine affinity viene denotata la possibilità di comportamenti divergenti in cui ad una domanda viene data al massimo una risposta, mentre linearity denota il comportamento in cui ad una domanda viene sempre data esattamente una risposta. Sono utilizzati sistemi di tipi per verificare la sicurezza del flusso di informazione in un programma, e quindi che le informazioni ad alto livello non possano essere trasmesse su canali di livello inferiore.
Uno dei punti deboli di questo approccio è la dimensione limitata per rappresentare i sistemi. Infatti lo spazio degli stati può crescere esponenzialmente nella dimensione della sua descrizione a causa della composizione parallela e degli operatori di scoping, generando l'esplosione degli stati. Per risolvere questo problema sono state introdotte analisi simboliche [4, 17, 26, 32, 53, 63] che consentono di ottenere modelli finiti e quindi adatti per una verifica automatica delle proprietà di sicurezza.

3.2.6 Tecniche delle Teorie della Complessità

I primi lavori in questa direzione furono presentati negli anni '80 da Goldwasser e Micali [43], seguiti da Blum e Micali [11] e Yao [97]. Questa teoria afferma che la sicurezza può essere provata basandosi su assunzioni della teoria della complessità. La tecnica di base della verifica di questo tipo di approccio utilizza strumenti probabilistici con lo scopo di provare che un determinato sistema di comunicazione sia una soluzione probabilmente buona ai problemi di sicurezza trattati, tale metodologia di analisi prende il nome di *provably secure*. Se si vuol dimostrare che un dato problema sia provably secure, occorre definire gli scopi, un protocollo e una dimostrazione che il protocollo stesso soddisfi gli intenti proposti, assumendo che le proprietà basate sulla teoria della complessità siano verificate.
Bellare e Rogaway [8] sono stati i precursori di questo approccio e hanno utilizzato funzioni pseudorandom [11, 97] per analizzare il problema di distribuzione delle chiavi sicure in un sistema insicuro. La loro analisi ha fornito una stima della probabilità con la quale la sicurezza del protocollo analizzato può essere compromessa. Il modello del sistema in analisi prevede il controllo delle comunicazioni da parte di un'entità ostile che ha lo scopo di compromettere la sicurezza delle informazioni scambiate nelle comunicazioni del sistema. La verifica consiste nel dimostrare che ogni possibilità di compromettere il protocollo comporta l'inversione delle funzioni pseudorandom e, poichè il calcolo dell'inversa di queste funzioni ha complessità di calcolo esponenziale [42], un protocollo, che può essere attaccato solo tramite tale operazione, può essere considerato sufficientemente sicuro. Viceversa, se esiste un procedimento di attacco con complessità di computazione polinomiale, allora esiste un buco nella sicurezza del protocollo analizzato.

3.2.7 Tecniche dell'Invariante

L'approccio basato su tecniche dell'invariante utilizza alcune delle tecniche descritte precedentemente, in modo particolare quella di applicazione generale e quella delle algebre di processo, per verificare le proprietà di sicurezza di un protocollo espresse sotto forma di invarianti del sistema. Il sistema viene solitamente modellato operando una separazione fra le

descrizione dell'agente ostile e la descrizione degli altri agenti che comunicano nel rispetto del protocollo.
Un approccio di questo tipo per l'analisi dei protocolli di autenticazione è stato introdotto da Bolignano [15, 16], il quale ha utilizzato il "Chemical Abstract Machine Paradigm" [6] per la verifica formale dei protocolli analizzati. Il modello viene descritto attraverso l'insieme delle sequenze di azioni atomiche che possono essere eseguite dal sistema ogniqualvolta le precondizioni di transito siano verificate. La verifica è effettuata tramite dimostrazioni che provano le proprietà di sicurezza espresse in forma di invarianti e i risultati ottenuti possono essere riutilizzati per molti altri protocolli, semplificando così le analisi successive. La tecnica di Bolignano è infatti orientata più alla riutilizzazione delle dimostrazioni che all'automazione della verifica.
Un altro approccio che si basa sulla tecnica dell'invariante è stato presentato da Schneider [83] che utilizza CSP [49] come linguaggio formale di specifica. L'utilizzo di CSP forza la separazione delle proprietà di sicurezza dai protocolli permettendo così una verifica più approfondita del sistema in analisi. Inoltre, poiché ogni descrizione CSP ha una precisa semantica, verificare se il protocollo rispetta o meno le proprietà di sicurezza è un problema puramente matematico. Le analisi di questo tipo sono caratterizzate dalla descrizione, attraverso l'introduzione di un sistema deduttivo, dell'insieme delle informazioni che l'intruso riesce a ricavare dai messaggi circolanti nella rete. Questo metodo può comportare un'incompletezza dell'analisi delle proprietà del protocollo studiato se la descrizione dell'insieme delle informazioni deducibili dall'intruso non è completa.

3.3 Problemi delle Analisi Formali e Informali

Nelle analisi sia formali che informali dei protocolli di autenticazione si possono individuare due problemi: la complessità del sistema o delle componenti studiate e l'ambiguità delle loro descrizioni. Perciò bisogna porre molta attenzione nelle definizioni delle proprietà di sicurezza e della base sicura di computazione, la quale deve essere mantenuta il più semplice e piccola possibile. Infatti l'ambiguità di alcune assunzioni sui sistemi che adottano protocolli crittografici si può ripercuotere sui metodi formali per analizzarli.
Un altro problema ricorrente nei metodi di analisi presenti in letteratura è causato dalle eccessive semplificazioni apportate al modello del protocollo studiato. In molti casi le verifiche estendono informalmente i risultati ottenuti dall'analisi di una singola sessione di comunicazione al caso multisessione, non rilevando così i problemi che si possono verificare dall'esecuzione contemporanea di più sessioni. Di conseguenza, oltre alla presenza di assunzioni ambigue, un'analisi formale può essere falsata dalla presenza di assunzioni implicite, cioè proprietà assunte, ma non esplicitate. Un esempio di modellazione che comporta l'assunzione di proprietà in modo implicito è data dalla descrizione diretta delle capacità di manipolazione delle informazioni attribuite all'intruso. Tale descrizione assume implicitamente che gli algoritmi crittografici sono in grado di impedire ogni altra derivazione da parte degli attaccanti, cosa non sempre verificata. La composizione dei risultati dell'analisi delle componenti di un sistema non è sempre sufficiente per verificare le proprietà di sicurezza nel suo complesso.

Infine, molte delle tecniche utilizzate per la verifica automatica delle proprietà dei protocolli hanno il problema dell'esplosione degli stati del modello adottato. Rusinowich e Turuani [81] dimostrano, in un approccio basato sulle algebre di processo, che il problema dell'insicurezza dei protocolli per un numero finito di sessioni è NP-completo. L'analisi genera un'esecuzione del protocollo e una possibile sostituzione ground e verifica in tempo polinomiale la loro correttezza. Inoltre, viene provato che nel costruire un attacco, con un numero fissato di sessioni, l'attaccante ha bisogno solo di un messaggio di dimensione polinomiale.

3.4 Modello di Dolev-Yao

Un modello comunemente utilizzato per formalizzare le capacità degli attaccanti è il modello proposto da Dolev e Yao [31]. In questo modello si assume che l'attaccante possa osservare, eliminare e selezionare arbitrariamente i messaggi spediti sui canali pubblici di comunicazione. L'agente ostile può, inoltre, creare nuovi messaggi da quelli osservati precedentemente e introdurli nei canali. L'attaccante può dividere i messaggi non codificati nelle loro parti e decriptare i messaggi se conosce la chiave di codifica corretta. Si assume, inoltre, che i messaggi contengano abbastanza ridondanza così che il ricevente possa sempre determinare se la codifica ha successo. Per esempio, quando un agente ostile decripta un nonce codificato $\{N\}_K$ con una chiave K', può sempre dire se $K = K'$. Nel modello di Dolev-Yao l'agente ostile può decidere di intercettare qualsiasi messaggio trasmesso su un canale pubblico e sostituirlo con un messaggio costruito dalla sua conoscenza iniziale e dai messaggi spediti precedentemente dagli agenti o di un'altra sessione del protocollo. Riassumiamo ora le assunzioni fondamentali introdotte dal modello.

Avversario Nondeterministico : un avversario potrebbe tentare ogni possibile attacco: egli controlla la rete di comunicazione e può osservare, eliminare o modificare i messaggi che viaggiano nella rete.

Crittografia Perfetta : i meccanismi crittografici non sono vulnerabili, consentendo attacchi solo alla struttura del protocollo. Per esempio, un attaccante che non ha accesso ad una chiave non può ottenere alcuna informazione dal ciphertext.

I passi ottenuti dagli agenti onesti che seguono le specifiche del protocollo e le azioni non deterministiche dell'attaccante danno origine al modello astratto del protocollo come un sistema di transizione degli stati.
Le assunzioni, sopra presentate, semplificano in modo considerevole l'analisi dei protocolli, si che il modello di Dolev-Yao è tuttora largamente utilizzato per astrarre il comportamento di sistemi crittografici. L'approccio generale seguito in un'analisi formale dei protocolli di sicurezza è analizzare tutte le tracce probabili del sistema di transizione degli stati e determinare per ogni traccia se le proprietà di sicurezza dediderate sono preservate, cioè un protocollo è considerato sicuro se non sono possibili azioni che rompono la sicurezza. Questo compito è complicato dalle seguenti considerazioni:

- Ci può essere un numero arbitrario di sessioni del protocollo che possono essere intrecciate in modo arbitrario.
- Un agente può partecipare ad arbitrarie sessioni del protocollo allo stesso tempo. La memoria di ogni agente è quindi illimitata.
- L'attaccante può generare un numero illimitato di messaggi.
- I nonce hanno visibilità limitata: gli agenti onesti dimenticano i nonce non appena la corrispondente sessione del protocollo è completata.

Il modello di Dolev-Yao, comunque, non risolve il problema dell'indecidibilità della verifica causato dai comportamenti ricorsivi dei sistemi crittografici e dall'esplosione degli stati. Amadio e Charatonik [3] hanno studiato il problema della raggiungibilità del controllo per protocolli crittografici ricorsivi utilizzando il modello di Dolev-Yao, basandosi su metodi completamente automatici. I protocolli crittografici sono programmi di lunghezza finita e senza cicli, e il problema della raggiungibilità del controllo su di essi può essere risolto in NPTIME [5]. I protocolli possono essere eseguiti più volte dagli agenti istanziando così diverse sessioni e, in ognuna di queste, un agente potrebbe generare nomi "fresh", che rappresentano, per esempio, nonces o key. L'analisi è così in grado di individuare attacchi che utilizzano messaggi scambiati in sessioni precedenti o parallele.
Alcuni autori hanno cercato di definire analisi automatiche basate sulle assunzioni fatte nel modello di Dolev-Yao. Monniaux [68] introduce *tree automata* per rappresentare la conoscenza dell'avversario e, nei suoi studi, restringe l'attenzione a protocolli senza ricorsione. Più tardi, Genet e Klay [39], e Goubault [45] utilizzano tree automata per produrre un'approssimazione di protocolli crittografici. Genet e Klay ottengono approssimazioni piuttosto grossolane, in cui ogni azione può essere sempre eseguita e concludono affermando la necessità di un'analisi raffinata del controllo. Goubault, invece, propone di approssimare un generatore di nomi da un superset di tutti i valori che possono essere generati in un'esecuzione. In questi due lavori, però, non viene data alcuna analisi della complessità e della precisione dello schema approssimato.
Infine, Common [26] introduce una classe di tree automata con memoria che permette di decidere in tempo DEXPTIME proprietà di sicurezza per una classe di protocolli crittografici che contiene strettamente i ping-pong protocols. In questo tipo di protocollo gli agenti, che si inviano messaggi, devono generare qualche tipo di conferma quando le informazioni sono ricevute correttamente. Per l'agente ricevente, il modo più semplice di gestire questa conferma consiste nell'inviare al mittente un messaggio quando la trasmissione ha avuto successo per ciascun pacchetto dati inviato. Il mittente attende quindi questo pacchetto di conferma prima di trasmettere il pacchetto di dati successivo. Se la conferma non torna dopo un certo periodo di tempo, o se il messaggio ritornato indica un errore di trasmissione, il mittente trasmette di nuovo il pacchetto.

Capitolo 4

Algebre di Processo

I sistemi distribuiti sono da tempo oggetto di studio e un'attenzione particolare è stata loro dedicata negli ultimi anni soprattutto per il crescente sviluppo di quella che può essere definita la rete geografica per eccellenza, Internet. Con sistemi distribuiti intendiamo una collezione di calcolatori autonomi, collegati attraverso una rete di interconnessione e dotati di software, che hanno lo scopo di creare servizi integrati. In questo ambito assume molta importanza la sicurezza. La possibilità di trasmettere le informazioni su tutto il sistema consente ad ogni utente di reperire qualsiasi tipo di dato, anche quello che non si vorrebbe diffondere. Quindi, le informazioni private, inserite in un sistema distribuito, devono essere protette da accessi indesiderati, per evitare di farle consultare o modificare. Uno degli approcci più utilizzati per lo studio di sistemi distribuiti si basa sulle algebre di processo. Queste sono uno strumento che si è dimostrato utile per modellare sistemi distribuiti e studiare le loro proprietà di sicurezza.
Presentiamo ora le idee che hanno spinto vari ricercatori a introdurre le algebre di processo. Definiamo la nozione di algebra come struttura matematica, rivolgendo un'attenzione particolare alle algebre di processo. Concludiamo presentando il π-calculus [65, 66, 67, 82] e lo spi-calculus [1], due delle algebre più utilizzate per modellare sistemi distribuiti.

4.1 Algebra

Un'algebra è una struttura matematica largamente usata in informatica per modellare le proprietà dei programmi. Presentiamo le nozioni basi per capire tale struttura riprendendo il lavoro di Hennessy [48].
Una *signature* è un insieme (finito o infinito) di simboli di funzione. I quattro simboli di funzione {*Zero*,*Succ*,*Pred*,*Plus*} sono, per esempio, una signature adeguata per i numeri naturali. Si utilizza Σ per denotare una signature.
Ad ogni simbolo di funzione è associata un'arietà che restituisce il numero di argomenti della funzione che rappresenta. Per esempio, l'arietà di *Plus* è 2, di *Succ* e *Pred* è 1, e di *Zero* è 0. *Zero* è infatti una costante che, per amor di uniformità, è chiamata simbolo di funzione di arietà 0. Formalmente l'arietà di una signature è una mappa, $ar : \Sigma \to \mathbb{N}$. Ad ogni simbolo f in Σ è associata la sua arietà, $ar(f)$, che è un numero naturale. Si utilizza Σ_n per denotare

l'insieme dei simboli di funzione in Σ con arietà n.

Definizione 4.1.1 *Sia Σ una signature. Una Σ-*algebra *è una coppia $\langle A,\Sigma_A\rangle$ dove*

(i) A è un insieme chiamato carrier

(ii) Σ_A è un insieme di funzioni $\{f_A : A^n \to A \mid f \in \Sigma \wedge ar(f) = n\}$

Così una Σ-algebra è semplicemente un'interpretazione di una signature Σ e consiste di un insieme A e di un'interpretazione su A di ogni simbolo di funzione in Σ. Una Σ- algebra può essere vista, inoltre, come un insieme con struttura. Il carrier A ha una struttura imposta dalle funzioni Σ_A nel senso che gli elementi di A possono essere manipolati usando solo queste funzioni. Naturalmente una data signature può avere diverse interpretazioni, perfino sullo stesso carrier. Comunque, spesso $\langle A,\Sigma_A\rangle$ è abbreviato con A, quando le funzioni Σ_A sono evidenti dal contesto.
Per ogni signature Σ esiste una particolare Σ-algebra chiamata *algebra dei termini*, o *free algebre*, per Σ. Queste algebre sono costituite da oggetti formali. Il carrier è costituito da sequenze di simboli o stringhe, chiamate *termini*, che sono costruite usando i simboli di funzione in Σ. Le funzioni delle algebre dei termini manipolano solo questi termini.

Definizione 4.1.2 *L'*insieme dei termini *su Σ, T_Σ, è il più piccolo insieme di stringhe che soddisfa le seguenti condizioni:*

(i) se $f \in \Sigma$ ha arietà 0 allora la stringa consistente del simbolo f è in T_Σ

(ii) se $f \in \Sigma$ ha arietà $k > 0$ allora la stringa della forma $f(t_1,...,t_k)$ è in T_Σ, ogni qual volta $t_1,...,t_k$ sono stringhe in T_Σ.

Gli elementi di T_Σ sono, quindi, stringhe di simboli "(", ")", insieme con i simboli di Σ, costruiti utilizzando le regole (i) e (ii) mostrate sopra. Notiamo che se Σ non contiene simboli di funzione di arietà 0, cioè non contiene costanti, allora T_Σ è vuoto. Per f in Σ di arietà k, $f_{T_\Sigma} : T_\Sigma^k \to T_\Sigma$ è la funzione che mappa una tupla di termini $\langle t_1,...,t_k\rangle$ nel termine $f(t_1,...,t_k)$. Se f ha arietà 0, allora f_{T_Σ} è semplicemente la costante in T_Σ consistente della stringa "f".
Il carrier di una algebra dei termini T_Σ è definita induttivamente. Tale algebra è almeno l'insieme di stringhe che contiene i simboli di costante e che è chiuso per le operazioni dell'algebra di termini, cioè per le operazioni f_Σ per ogni f in Σ. La natura induttiva di T_Σ dà un metodo di dimostrazione molto potente per derivare le proprietà dei termini. Per mostrare che la proprietà P vale per tutti i termini in T_Σ è sufficiente

(i) provare che P vale per ogni simbolo di costante in Σ e,

(ii) assumendo che P vale per i termini $t_1,...,t_k$, provare che P vale per il termine $f(t_1,...,t_k)$ per ogni f in Σ di arietà k, $k > 0$.

Questa è chiamata *induzione strutturale* e come l'induzione si basa sulla struttura sintattica dei termini. L'induzione strutturale può anche essere usata per definire una funzione g su T_Σ.

(i) Definire il risultato dell'applicazione di g ai simboli di costante.

(ii) Definire il risultato dell'applicazione di g a $f(t_1,...,t_k)$ in termini di $g(t_1),...,g(t_k)$ per ogni f in Σ di arietà k, con $k > 0$.

Le algebre di termini sono di natura sintattica e definiscono la sintassi del linguaggio. Infatti, molti linguaggi di programmazione o linguaggi formali sono costruiti in modo tale che la loro sintassi sia una algebra di termini per qualche particolare signature Σ. La loro semantica, d'altra parte, può essere vista come una Σ-algebra. Infatti, definire la semantica (denotazionale) di un linguaggio equivale a costruire una particolare Σ-algebra che soddisfa alcuni vincoli particolari.
Per spiegare il significato matematico di algebra di termini bisogna introdurre i *Σ-omomorfismi*, che sono semplicemente funzioni tra carrier di Σ-algebre che preservano la struttura imposta dalle Σ-algebre.

Definizione 4.1.3 *Siano $\langle A,\Sigma_A\rangle$ e $\langle B,\Sigma_B\rangle$ due Σ-algebre e h una funzione da A a B. Allora h è un Σ-omomorfismo se, per ogni f in Σ di arietà k, $h(f_A(a_1,...,a_k)) = f_B(h(a_1),...,h(a_k))$.*

Gli omomorfismi sebbene preservano la struttura dell'algebra, possono collassare il loro dominio nel senso che molti elementi del dominio possono essere mappati nello stesso elemento. In generale un Σ-omomorfismo si comporta nello stesso modo delle funzioni ordinarie sull'insieme. Per esempio, se $f : A \to B$, $g : B \to C$ sono Σ-omomorfismi, allora lo è anche la loro composizione $g \circ f : A \to C$. Inoltre, la funzione identità su A è un Σ-omomorfismo su ogni Σ-algebra $\langle A,\Sigma_A\rangle$.

Teorema 4.1.4 *Per ogni Σ-algebra $\langle A,\Sigma_A\rangle$ esiste un unico Σ-omomorfismo $i_A : T_\Sigma \to A$.*

Se vediamo T_Σ come la sintassi del linguaggio e la Σ-algebra come il dominio semantico o l'interpretazione, allora il teorema afferma che ogni espressione o termine nel linguaggio ha un unico significato in $\langle A,\Sigma_A\rangle$, cioè esiste solo un modo per interpretare il linguaggio nel dominio semantico.

4.2 Algebra di Processo

Il termine "algebra di processo" è stato introdotto nel 1982 da Bergstra e Klop per indicare un'algebra universale che soddisfa un particolare insieme di assiomi dove processo è inteso come un'astrazione matematica delle interazioni tra un sistema e il suo ambiente. Un'algebra di processo è un approccio algebrico che consente di studiare processi distribuiti e i cui strumenti sono linguaggi algebrici volti a descrivere processi, e formulare e verificare dichiarazioni su di essi.
L'obiettivo di tali algebre è catturare la nozione di comunicazione e di concorrenza, entrambe essenziali per la comprensione di sistemi dinamici complessi. Da un lato tali sistemi sono composti da più parti che agiscono concorrentemente e indipendentemente dalle altre; dall'altro lato un sistema complesso è unico in quanto le sue parti sono unite attraverso la comunicazione. Fondamentale, per entrambe queste nozioni, è l'assunzione che ognuna delle

parti, chiamate *agenti*, abbia una propria identità. Senza questa assunzione sarebbe difficile distinguere, tra gli eventi, i comportamenti del sistema.
Una parte essenziale della teoria dei sistemi complessi è una nozione precisa e utilizzabile di comportamento. Il comportamento è il modo in cui un agente interagisce con il resto del sistema ed è ragionevole definirlo come le capacità di comunicazione del sistema stesso. In altre parole, il comportameno di un sistema è esattamente ciò che è osservabile, e osservare un sistema equivale a comunicare con esso. Le algebre di processo, a differenza di altri approcci presentati precedentemente, si focalizzano sulle transizioni osservabili piuttosto che sugli stati raggiungibili.
Nelle algebre di processo un agente è identificato con le azioni che descrivono il suo comportamento. Ogni sua azione o è un'interazione con gli altri agenti del sistema, e allora è una comunicazione, o agisce indipendentemente da loro, e allora è in concorrenza. Solitamente ci sono due tipi di azioni: azioni di input e azioni di output. La comunicazione è vista come l'occorrenza simultanea di due azioni: una di input e l'altra di output. In un sottoprocesso un canale può essere utilizzato sia per comunicare con un altro sottoprocesso sia con l'ambiente. Per descrivere un processo in cui i canali sono utilizzati solo per comunicazioni interne, si introduce la nozione di restrizione. La restrizione è molto potente e consente di astrarsi dai dettagli di funzionamento interno dei processi.
Un'attenzione particolare del formalismo, basato sulle algebre di processo, è rivolta agli operatori che costruiscono processi complessi derivandoli da quelli più semplici. Solitamente, nelle agebre di processo c'è un operatore binario di "parallelo", scritto |, tale che se P e Q sono processi allora $P|Q$ è un processo che stabilisce il comportamento di P e di Q eseguiti parallelamente. Un'algebra di processo ha una piccola collezione di tali operatori attraverso i quali possono essere costruiti gerarchicamente processi di complessità arbitraria. Questo metodo conduce ad un formalismo specifico che è di base, ma molto potente.
Le prime algebre di processo definite in letteratura sono CSP di Hoare [49] e CCS di Milner [64]. Hoare mostra come il comportamento di un processo può essere memorizzato come una traccia della sequenza di azioni. Una traccia del comportamento di un processo è una sequenza finita di simboli che memorizzano ordinatamente gli eventi che il processo ha eseguito. Hoare immagina che ci sia un osservatore che annota il nome di ogni evento che occorre. L'osservatore non tiene conto della possibilità che due eventi occorrono simultaneamente e memorizza prima uno e poi l'altro senza dare importanza all'ordine in cui li memorizza. Hoare rappresenta le tracce di un processo come un albero e, per ogni suo nodo, la traccia del comportamento è la sequenza delle etichette incontrate sul cammino che porta dalla radice dell'albero a quel nodo. Hoare permette, inoltre, di specificare i processi in anticipo all'implementazione attraverso la descrizione delle proprietà delle sue tracce e fornisce delle regole per favorire l'implementazione del processo. Milner, invece, assegna a ogni costrutto sintattico del calcolo una semantica operazionale, introducendo le nozioni di derivazione e di albero di derivazione. Definisce una nozione di equivalenza tra agenti basata intuitivamente sull'idea che due agenti sono distinguibili se uno esterno, interagendo con loro, è in grado di distinguerli. A partire da questa idea, introduce le relazioni di *observation equivalence* e quella di più forte di *strong equivalence*. In particolare mostra che l'ultima è una relazione di congruenza, cioè che è preservata da tutti i contesti algebrici. Milner utilizza le nozioni appena introdotte per costruire un modello matematico che consente di analizzare

il comportamento dei sistemi.

4.3 π-calculus

Il π-calculus è stato introdotto da Milner [66] e unisce l'idea del λ-calculus e il calcolo dei processi. Il π-calculus punta all'universalità per la computazione concorrente, come il λ-calculus è universale per la computazione funzionale. Il π-calculus fornisce un ambiente di lavoro per studiare la mobilità, e gli strumenti matematici per rappresentare sistemi mobili e per ragionare sul loro comportamento.
Il π-calculus è un'algebra di processo che consente di descrivere processi concorrenti, focalizzando l'attenzione sulla comunicazione attraverso canali. La potenza del calcolo è basata sulla nozione di *passaggio dei nomi*, cioè si consente ai dati trasmessi attraverso i canali di comunicazione di diventare a loro volta canali, introducendo notevoli semplificazioni e generalizzazioni rispetto ai modelli di concorrenza già esistenti: reti di Petri [79], CSP di Hoare [49], CCS di Milner [64].
Il π-calculus è un linguaggio piccolo ma estremamente espressivo. I programmi, in questo calcolo, sono sistemi di processi indipendenti e paralleli che sincronizzano, attraverso il passaggio di messaggi, comunicazioni su canali. I canali, che un processo conosce, determinano le sue possibili comunicazioni e possono essere ristretti, in modo che solo certi processi comunicano su di loro. In questo aspetto il π-calculus è simile ai calcoli di processi precedenti come CSP e CCS. Il π-calculus si distingue da quest'ultimi in quanto lo scoping della restrizione, cioè il testo del programma, in cui un canale può essere usato, potrebbe cambiare durante la computazione. Quando un processo spedisce un canale ristretto come messaggio ad un processo esterno allo scope della restrizione, lo scope è detto *extrude*, cioè si allarga per inglobare il processo che riceve il canale. I processi in π-calculus sono mobili nel senso che, conoscendo nomi di nuovi canali attraverso l'estrusione, le capacità di comunicazione dei processi possono cambiare nel tempo.
Il calcolo è utile per rappresentare sistemi comunicanti in cui si possono esprimere processi che hanno una struttura dinamica. Infatti non solo gli agenti del sistema possono essere collegati, ma anche la comunicazione tra sistemi vicini potrebbe portare informazioni che cambiano quei collegamenti.
Diamo ora la definizione formale del π-calculus riprendendo quella di Sangiorgi e Walker [82]. Due sono i tipi base di entità nel calcolo: nomi e processi. I nomi sono i nomi dei collegamenti, i processi interagiscono usando i nomi che condividono. I processi, inoltre, si evolvono a causa delle esecuzioni di azioni, e le capacità per le azioni sono espresse attraverso i prefissi.

Definizione 4.3.1 *I* prefissi *di un processo sono definiti come segue.*

$$
\begin{array}{lcll}
\pi & ::= & \overline{x}y & \textit{azione di output} \\
 & & x(y) & \textit{azione di input} \\
 & & \tau & \textit{silent move} \\
 & & [x = y]\pi & \textit{matching}
\end{array}
$$

Vediamo ora in dettaglio il significato delle azioni.

- L'azione di input $\overline{x}y$ è la capacità di spedire il nome y attraverso il nome x.
- L'azione di input $x(y)$ è la capacità di ricevere qualsiasi nome attraverso il nome x.
- La silent move τ è una capacità per azioni non osservabili, cioè azioni interne al processo.
- Il matching $[x = y]\pi$ è la capacità condizionale: l'azione π può essere eseguita se x e y sono lo stesso nome.

Definiamo ora gli operatori necessari per costruire i processi.

Definizione 4.3.2 *I* processi *del π-calculus sono dati dalla seguente grammatica.*

P,Q	::=	$\boldsymbol{0}$	*nil*
		$\pi.P$	*prefisso*
		$(\nu z)P$	*restrizione*
		$P\|Q$	*composizione parallela*
		$P + Q$	*scelta non deterministica*
		$!P$	*replicazione*

Diamo una descrizione dell'interpretazione dei processi.

- **0** è il processo che non fa niente.
- Il prefisso $\pi.P$ ha una singola capacità, espressa da π. Il processo P non può essere eseguito fino a che l'azione π non è stata eseguita.
  - Il prefisso di output $\overline{x}y.P$ può mandare il nome y attraverso il nome x e continuare come P.
  - Il prefisso di input $x(z).P$ può ricevere ogni nome attraverso x e continuare come P, dove il nome ricevuto è sostituito a z in P.
  - Il prefisso inosservabile $\tau.P$ evolve in modo invisibile a P.
  - Il prefisso di match $[x = y]\pi.P$ può evolvere come $\pi.P$ se x e y sono lo stesso nome, altrimenti non fa niente.
- Nella restrizione $(\nu z)P$, lo scope del nome z è ristretto a P. I sottoprocessi di P possono usare z per interagire con altri sottoprocessi di P, ma non con altri processi.
- Nella composizione parallela $P|Q$, le componenti P e Q possono procedere indipendentemente o interagire attraverso nomi condivisi.
- Nella scelta non deterministica $P + Q$ può venire eseguita o la componente P o la componente Q. Quando un ramo del processo viene eseguito, l'altro viene scartato.

- La replicazione $!P$ può essere pensata come la composizione infinita $P|P|...$, o equivalentemente come un processo che soddisfa l'equazione $!P = P|!P$. La replicazione è l'operatore che rende possibile esprimere comportamenti infiniti.

I dati che i processi comunicano durante una interazione possono essere usati dal processo ricevente per partecipare in un'altra interazione. Ricevuto un nome, un processo acquista la capacità di interagire con processi che lo conoscono a sua volta. La struttura di un sistema, cioè la connessione tra i suoi processi componenti, può così cambiare dinamicamente in modo arbitrario. La sorgente della potenza del π-calculus è data dallo scoping dei nomi e l'estrusione dei nomi dal loro scoping.

4.4 spi-calculus

Il π-calculus è in grado di descrivere i protocolli di sicurezza, anche ad un livello astratto ed è in grado di rappresentare perfino canali asimmetrici, cioè quelli necessari per implementare crittografia a chiave pubblica. Estendendo quest'algebra, però, tale rappresentazione potrebbe risultare più semplice e più efficiente. Comunque, l'operatore di restrizione e l'estrusione risultano fondamentali per descrivere protocolli di sicurezza anche in un'estensione del π-calculus.
Lo spi-calculus è stato definito da Abadi e Gordon [1], estendendo il π-calculus con primitive di crittografia. Questo calcolo è progettato per descrivere protocolli di sicurezza per l'autenticazione e il commercio elettronico. Lo spi-calculus è in grado di fornire descrizioni di protocolli di sicurezza più dettagliate di quelle del π-calculus, mentre il π-calculus rappresenta i canali, lo spi-calculus implementa i canali in termini di crittografia. Come nel calcolo introdotto da Milner, lo scoping è alla base della sicurezza nello spi-calculus, infatti la restrizione può essere usata per modellare la creazione di chiavi crittografiche fresh e non indovinabili. Abadi e Gordon, definendo il calcolo, fanno assunzioni significative sulla crittografia.

- Il solo modo per decodificare un messaggio criptato è conoscere la chiave corrispondente.

- Un messaggio criptato non rivela la chiave che è stata usata per codificarlo.

- C'è sufficente ridondanza nei messaggi così che l'algoritmo di decodifica può determinare se un ciphertext è codificato con la chiave attesa.

La definizione formale dello spi-calculus, di seguito presentata, riprende quella di Abadi e Gordon. Si assume un insieme infinito di nomi, che sono usati per i canali di comunicazione e un insieme infinito di variabili.

Definizione 4.4.1 *L'insieme dei* termini *è definito dalla seguente grammatica.*

M,N	::=	x	*variabili*
		n	*nomi*
		0	*zero*
		$suc(M)$	*successore*
		M^+	*chiave pubblica*
		M^-	*chiave privata*
		(M,N)	*coppia*
		$H(M)$	*funzione hash*
		$\{M\}_N$	*codifica a chiave condivisa*
		$\{\!\|M\|\!\}_N$	*codifica a chiave pubblica*
		$[\![M]\!]_N$	*firma a chiave privata*

Nel π-calculus i termini sono solo nomi. Per convenienza sono stati aggiunti i costrutti per l'accoppiamento, per l'hashing, per la codifica a chiave pubblica e a chiave condivisa e per la firma a chiave privata. Inoltre sono state distinte le variabili dai nomi. Vediamo ora in dettaglio le operazioni crittografiche da loro introdotte e le assunzioni che esse comportano.

- Il messaggio $H(M)$ rappresenta la codifica attraverso la funzione hash H del messaggio M. Le funzioni hash devono essere one-way e collision free, cioè recuperare un input dalla loro immagine e trovare due input con la stessa immagine risulta computazionalmente espansivo. Quando una funzione hash viene rappresentata nello spi-calculus, si assume che queste operazioni sono impossibili.

- Il messaggio $\{M\}_N$ rappresenta il testo cifrato ottenuto codificando il messaggio M con la chiave N. È da mettere in evidenza che si consente l'uso di messaggi arbitrari come chiavi condivise e questo rende possibile, per esempio, modellare un protocollo dove una chiave è generata usando la funzione hash su un nuovo segreto condiviso.

- Il messaggio $\{\!\|M\|\!\}_N$ dove N è una chiave pubblica, rappresenta il risultato della codifica a chiave pubblica di M con N. I tradizionali sistemi crittografici a chiave pubblica sono basati su coppie di chiavi. Normalmente, una delle chiavi in ogni coppia è privata, cioè conosciuta da un solo agente, mentre l'altra chiave è pubblica. Ogni agente può crittografare un messaggio usando la chiave pubblica, mentre solo un agente che conosce la chiave privata può decriptare il messaggio. Nello spi-calculus viene assunto che non è possibile ricavare una chiave dall'altra.

- Il messaggio $[\![M]\!]_N$ dove N è una chiave privata, rappresenta la firma di M con N. Anche per la firma digitale vengono usate una coppia di chiavi. Le chiavi private sono usate per firmare, mentre le chiavi pubbliche sono usate per controllare la firma. Le assunzioni fatte sono le stesse che per la crittografia a chiave pubblica.

Definizione 4.4.2 *L'insieme dei* processi *è definito dalla seguente grammatica.*

P,Q	::=	$\overline{M}\langle N\rangle.P$	*processo di output*
		$M(x).P$	*processo di input*
		$P\|Q$	*composizione parallela*
		$(\nu n)P$	*restrizione*
		$!P$	*replicazione*
		$[M \textit{ is } N]P$	*matching*
		$\mathbf{0}$	*nil*
		let $M = (x,y)$ *in* P	*pair splitting*
		case M *of* $0 : P$ $suc(x) : Q$	*case intero*
		case L *of* $\{x\}_N$ *in* P	*decodifica con chiave condivisa*
		case L *of* $\{\!\{x\}\!\}_N$ *in* P	*decodifica con chiave privata*
		case L *of* $[\![x]\!]_N$ *in* P	*controllo della firma*

Vediamo in dettaglio i costrutti che costituiscono i processi. Viene utilizzata la notazione $P[x \mapsto M]$ per indicare il rimpiazzamento di ogni occorrenza libera di x nel processo P con il messaggio M.

- Il passo base di computazione e il meccanismo di sincronizzazione è l'interazione, in cui un termine N è comunicato da un processo di output a un processo di input attraverso il canale conosciuto M.
  - Un processo di output $\overline{M}\langle N\rangle.P$ è pronto per trasmettere sul canale M. Se avviene un'interazione N è comunicato su M e poi si esegue il processo P.
  - Un processo di input $M(x).P$ è pronto per ricevere dal canale M. Se avviene un'interazione in cui N è comunicato su M, allora si esegue il processo $P[x \mapsto N]$. In $M(x).P$ la variabile x è legata in P.
- Il processo $P|Q$, ottenuto per composizione parallela dei processi P e Q, si comporta come i processi P e Q eseguiti in parallelo. Ognuno può interagire con l'altro sui canali conosciuti da entrambi, o con il mondo esterno, indipendentemente dall'altro.
- Il processo $(\nu n)P$, ottenuto applicando l'operatore di restrizione al processo P, è un processo che rende un nome nuovo n privato e si comporta come P. In $(\nu n)P$ il nome n è legato in P.
- Il processo $!P$ si comporta come un numero infinito di copie di P eseguite in parallelo.
- Il processo $[M \textit{ is } N]P$ si comporta come P purché i termini M e N sono gli stessi, altrimenti è bloccato, cioè non fa niente.
- Il processo nil $\mathbf{0}$ non fa niente.
- Il processo *let* $M = (x,y)$ *in* P si comporta come $P[x \mapsto N][y \mapsto L]$ se il termine M è la coppia (N,M). Altrimenti il processo si blocca. Le variabili x e y sono legate in P.
- Il processo *case* M *of* $0 : P$ $suc(x) : Q$ si comporta come P se il termine M è 0, come $Q[x \mapsto N]$ se M è $suc(N)$. Altrimenti il processo è bloccato. La variabile x è legata nel secondo ramo, Q.

- Il processo *case* L *of* $\{x\}_N$ *in* P cerca di decriptare il termine L con la chiave N. Se L è un ciphertext della forma $\{M\}_N$ allora il processo si comporta come $P[x \mapsto M]$. Altrimenti il processo si blocca. La variabile x è legata in P.

- Il processo *case* L *of* $\{\!|x|\!\}_N$ *in* P è utile quando N è una chiave privata K^-. Questo costrutto cerca di decriptare il termine L con la chiave privata K^-. Se L è un ciphertext della forma $\{\!|x|\!\}_{K^+}$ allora il processo si comporta come $P[x \mapsto M]$. Altrimenti il processo si blocca. La variabile x è legata in P.

- Il processo *case* L *of* $[\![x]\!]_N$ *in* P è utile quando N è una chiave pubblica K^+. Questo costrutto cerca di verificare la firma del termine L con la chiave pubblica K^+. Se L è della forma $[\![x]\!]_{K^-}$ allora il processo si comporta come $P[x \mapsto M]$. Altrimenti il processo si blocca. La variabile x è legata in P.

L'assenza di un costrutto per ricavare il termine M da $H(M)$ corrisponde all'assunzione che H non può essere invertita. La mancanza di un'equazione della forma $H(M) = H(M')$ corrisponde, invece, all'assunzione che H è collision free.

Capitolo 5

Analisi Simboliche

I sistemi crittografici sono necessari in molte applicazioni che richiedono la protezione dei dati trasmessi e l'autenticazione degli agenti. Detti sistemi stabiliscono il formato e la sequenza dei messaggi che, due o più agenti in accordo sul loro impiego, devono inviarsi, e utilizzano algoritmi crittografici per rendere le informazioni incomprensibili e difficilmente ricostruibili da eventuali intercettori. Per decenni i protocolli sono stati costruiti basandosi su prove ed errori, infatti una volta proposto uno schema si tentava di trovare l'errore e se non veniva trovato, allora lo schema era considerato corretto. La storia ha dimostrato che questo modo di procedere non è sempre soddisfacente, ed infatti molti protocolli presentati in letteratura come sicuri, si sono, in seguito, dimostrati inadeguati, rilevando la necessità di qualche nuovo strumento in grado di verificare le proprietà di sicurezza. Per tale motivo molte delle ricerche effettuate nel settore della sicurezza sono orientate allo studio e allo sviluppo di tecniche formali che si prefiggono di provare la sicurezza dei protocolli.
La verifica di protocolli di autenticazione utilizza un modello formale, che descrive il flusso di informazioni e l'interazione tra agenti in un ambiente ostile, e serve per controllare se un protocollo soddisfa determinate proprietà di sicurezza. Nel caso in cui quest'ultime non siano soddisfatte, la verifica dove stabilire come gli agenti ostili possano interagire con i protocolli causando il loro malfunzionamento.
Uno degli approcci più utilizzati per modellare protocolli di comunicazione si basa sul lavoro di Dolev e Yao [31] i quali definiscono un modello in cui si assume che i meccanismi crittografici, alla base del sistema, non siano vulnerabili, consentendo attacchi solo alla struttura del protocollo stesso, e che l'agente ostile controlli la rete di comunicazione. Le abilità di computazione assunte per gli attaccanti e le relazioni tra gli attaccanti e gli agenti onesti[1] del protocollo costituiscono il *threat model* [53]. In questo modello un insieme di agenti onesti comunica attraverso l'ambiente che è totalmente controllato da un attaccante il quale può così intercettare messaggi e introdurne di nuovi utilizzando le informazioni deducibili dai messaggi visti precedentemente. Si identifica così l'ambiente con l'attaccante. Il threat model è veramente generale e non è necessario assumere l'esistenza di attaccanti multipli, poiché questi non hanno più abilità dell'ambiente. I protocolli di sicurezza sono usualmente

[1] Qui "onesto" significa che l'agente procede seguendo precisamente il protocollo senza discostarsi dalle sue specifiche in nessun momento.

analizzati assumendo che la sola minaccia possa avvenire da attaccanti esterni e non da agenti legali del protocollo. Infatti, gli agenti che non seguono correttamente i passi del protocollo vengono considerati agenti ostili e il loro comportamento può essere assimilato all'ambiente. Infine, vengono analizzate solo situazioni in cui un attaccante ha tutti i poteri descritti nel modello in quanto i protocolli di sicurezza devono lavorare correttamente anche nelle situazioni peggiori.
Uno degli strumenti più utilizzati per modellare i protocolli di comunicazione sono le algebre di processo in cui un protocollo è descritto come un processo concorrente. L'analisi delle tracce generate da questo processo può essere usata per la verifica di proprietà di autenticazione e segretezza del protocollo. Comunque, l'insieme di queste tracce è tipicamente infinito. Sono due i fattori che rendono lo spazio degli stati, generato nell'analisi di protocolli crittografici, infinito. Il primo è dato da un numero infinito di messaggi diversi che l'ambiente può spedire agli agenti: anche quando restringiamo l'attenzione a esecuzioni finite di protocolli tra un numero finito di agenti, gli attaccanti possono mandare un qualsiasi messaggio ottenuto dalla manipolazione dei messaggi che sono conosciuti dall'ambiente. Il secondo fattore è dovuto allo studio di protocolli in cui il numero di agenti non è fissato a priori: un numero illimitato di agenti porta necessariamente a uno spazio infinito di stati. Mentre il primo tipo di infinito può essere affrontato, il secondo può portare a problemi di indecibilità.
Il numero infinito di messaggi conosciuti dall'ambiente conduce ad un'esplosione degli stati che rende il modello del protocollo infinito. Gli approcci, basati sul model checking, riducono il modello a una dimensione finita imponendo un limite al numero di parametri critici (numero di chiavi, numero di coppie e codifiche nei messaggi). Una esplorazione esaustiva dello spazio degli stati è così possibile con tecniche standard, ma questo approccio rende molto difficile la dimostrazione della completezza e della decidibilità dell'analisi.
Tecniche simboliche si basano sul lavoro di Dolev e Yao e si sono dimostrate utili nello studio di sistemi a stati infiniti. Tali tecniche sono state studiate per astrarre il flusso di informazione in sistemi concorrenti, ottenendo modelli finiti, e il loro utilizzo risulta così attraente nell'analisi di protocolli crittografici, in quanto per eseguire verifiche automatiche bisogna rappresentare le informazioni in una struttura finita. Solitamente le tecniche simboliche hanno due componenti fondamentali:

Riduzione simbolica: L'idea base dietro la riduzione simbolica è evitare l'istanziazione delle variabili nella specifica del protocollo a meno che non sia necessaria. A questo risultato si giunge definendo una relazione di transizione tra stati simbolici che dà origine ad uno spazio finito di stati simbolici. Ogni stato simbolico rappresenta un numero infinito di stati concreti che possono essere ottenuti istanziando le variabili in una rappresentazione degli stati simbolici. Le condizione di correttezza dei protocolli sono rappresentate da vincoli. Un vincolo tipico è la richiesta che ogni messaggio ricevuto dagli agenti onesti sia derivabile dalla conoscenza iniziale dell'ambiente insieme con i messaggi spediti dagli agenti su canali pubblici fino a quel punto.

Analisi della conoscenza: Ogni tecnica definisce un sistema deduttivo per determinare se un particolare messaggio può essere derivato da un dato insieme di messaggi. Ovviamente il sistema deduttivo dipende dal modello dell'attaccante scelto. Nel modello di

Dolev-Yao, perfino l'insieme dei messaggi, costruito da una conoscenza iniziale finita dell'attaccante, è infinito.

L'analisi dei protocolli consiste allora nel determinare se un attaccante può generare una traccia del protocollo che viola una delle condizioni di correttezza del protocollo, cioè se esiste un'istanziazione di variabili, che soddisfa i vincoli imposti da una traccia di fallimento delle proprietà di sicurezza.
L'uso di tecniche simboliche per l'analisi di protocolli non è nuovo e le prossime sezioni presentano come diversi autori abbiano affrontato tale problematica.

5.1 Huima

Oggigiorno esistono molti metodi e formalismi per analizzare protocolli crittografici e il loro sviluppo è una conseguenza di due fattori: i protocolli crittografici hanno elevate richieste di sicurezza a causa della loro area di applicazione e la progettazione di protocolli crittografici è incline a errori.
Per analizzare matematicamente un sistema è necessario un modello formale del sistema stesso. Usualmente il sistema non può essere analizzato in quanto è troppo complesso. Così il modello utilizzato nell'analisi è spesso un'approssimazione del sistema reale e cattura solo alcune sue proprietà. Un generico modello per protocolli di sicurezza consiste usualmente in tre parti:

1. Il *modello degli agenti onesti* definisce come i partecipanti al protocollo possono lavorare, cioè le capacità computazionali che hanno e cosa possono fare in generale.

2. Il *threat model* definisce le capacità assunte per gli attaccanti, cioè se possono o meno spedire messaggi falsi e quali sono le loro capacità computazionali.

3. Il *modello di crittografia* definisce le proprietà delle funzioni crittografiche.

Per Huima [53] il modello crittografico e il corrispondente threat model sono strettamente collegati. Un compito del threat model è definire quali conoscenze può ottenere un attaccante e che tipo di informazioni è in grado di derivare dalla sua conoscenza. Il modello di crittografia più utilizzato è il modello algebrico che è alla base delle logiche di autenticazione e dei metodi di analisi dello spazio degli stati.
Il modello algebrico è un'approssimazione della situazione del mondo reale ed è usato con successo in vari metodi per individuare errori in protocolli conosciuti. Comunque tale modello contiene alcuni difetti. Il primo problema è che nasconde proprietà quantitative. Ad esempio, non distingue un criptosistema basato su chiavi di 1 bit da uno basato su chiavi di 1024 bit. Il secondo problema è che il modello non individua gli errori dovuti alle informazioni indirette. Il terzo problema è che il modello ha fondamenti semantici deboli e così non è chiaro cosa significa "conoscere" o "possedere" un dato.
Huima definisce un modello completo della conoscenza degli attaccanti, che chiama *possible-worlds model*, estendendo quello algebrico per risolvere i problemi individuati, in particolare,

considerando informazioni quantitative. Tale modello risulta, però, troppo complesso. Allora definisce un nuovo modello computazionale in grado di approssimare il sistema reale, considerando solo alcune sue proprietà. Huima presenta una semantica simbolica, dalla quale l'esecuzione di un protocollo genera un insieme di vincoli in forma di equazione. Allora l'analisi simbolica si occupa di definire una relazione per stati simbolici globali che genera uno spazio finito di stati simbolici associato a vincoli. L'autore presenta, inoltre, un metodo di analisi che si basa sulla ricerca nello spazio degli stati simbolici. Questa ricerca è una forma di enumerazione dello spazio degli stati. La ricerca convenzionale si focalizza su stati individuali e li enumera, mentre quella simbolica eleva il livello di astrazione, infatti, invece di enumerare stati singoli, si occupa di insiemi di stati che sono rappresentati in qualche forma simbolica. In altre parole, gli insiemi non sono rappresentati come una collezione di stati individuali, ma come un predicato che risulta essere vero esattamente per gli stati contenuti nell'insieme. In questo modo si ottiene una rappresentazione finita di insiemi infiniti utilizzabile per verificare le proprietà di sicurezza dei protocolli. Le condizioni di correttezza del protocollo sono formulate come vincoli e l'insieme di questi è unito con quelli ottenuti durante l'analisi. Infine, ogni stato simbolico terminale è trasformato in modo da decidere se esiste un'istanziazione delle variabili che soddisfa tutti i vincoli simultaneamente. Decidere l'esistenza di un'istanziazione che soddisfa un vincolo richiede di risolvere il problema dell'analisi della conoscenza. Huima non fornisce, però, i dettagli dell'algoritmo usato per risolvere il problema del soddisfacimento dei vincoli e non dimostra se il metodo introdotto sia corretto e completo.

5.2 Amadio e Lugiez

Amadio e Lugiez [4] studiano la verifica delle proprietà di segretezza e autenticazione per protocolli che utilizzano algoritmi crittografici simmetrici con chiavi atomiche. Questi utilizzano un'algebra di processi simile allo spi-calculus per modellare i protocolli e, basandosi sul lavoro di Huima [53], hanno come obiettivo fornire una dimostrazione diretta della decidibilità per il problema della raggiungibilità in protocolli in cui il numero di agenti è finito. La verifica è eseguita controllando se un processo parallelo, che modella un protocollo, e la sua rappresentazione possono raggiungere uno stato errato interagendo con l'ambiente. Nella loro analisi Amadio e Lugiez presentano una procedura decisionale per la raggiungibilità che è basata su un sistema di riduzioni simboliche. In questo approccio la riduzione simbolica è combinata con l'analisi della conoscenza. Uno stato simbolico del protocollo è una tripla (P,T,E) dove P è lo stato del processo che rappresenta gli agenti onesti, T è l'insieme finito dei termini che rappresenta la conoscenza dell'attaccante, ed E è una lista ordinata di vincoli $x_1 : T_1,...,x_n : T_n$ tale che $T_1 \subseteq ... \subseteq T_n$. Ogni vincolo $x_i : T_i$ corrisponde ad un punto dell'esecuzione del protocollo dove la conoscenza dell'ambiente è data dall'insieme T_i. I valori di x_i sono i termini spediti agli agenti onesti del protocollo nella traccia. Per ogni passo di riduzione simbolica, l'algoritmo controlla se la sostituzione richiesta per il passo è compatibile con le sostituzioni calcolate precedentemente. L'algoritmo decide così se esiste una singola sostituzione che risolve tutti i vincoli $x_i : T_i$ simultaneamente.
Similmente a Huima [53] la loro semantica simbolica genera vincoli in forma di equazione

piuttosto che in forma di unifier. La procedura per la risoluzione dei vincoli è inserita nell'esecuzione simbolica e usa un metodo brute-force per istanziare le variabili in posizione di chiavi condivise. Questi fattori hanno un impatto rilevante sulla dimensione del modello ottenuto e rendono l'algoritmo decisionale NP-hard.
Amadio e Lugiez osservano che Huima [53] considera una signature con costruttori, come codifica e accoppiamento, e distruttori, come decodifica e proiezione, e nel suo approccio i termini sono considerati al di sopra delle uguaglianze indotte da un sistema di riscrittura di termini canonico. Nel loro approccio, invece, le funzioni di decodifica e di proiezione sono eseguite implicitamente: gli agenti possono decodificare e fare proiezioni usando operatori di *case* e l'ambiente può decriptare e fare proiezioni in accordo alla definizione data per gli operatori. Seguendo l'approccio di Monniaux [68], Amadio e Lugiez utilizzano tree automata per rappresentare l'insieme dei messaggi che possono essere generati dall'ambiente e per astrarre i valori che le variabili di input possono assumere.
Il problema della raggiungibilità corrisponde a determinare se una configurazione può raggiungere uno stato di errore. Il metodo, utilizzato per specificare una particolare proprietà, è introdurre un processo osservatore che raggiunge lo stato di errore quando la proprietà è violata.

5.3 Boreale

Un protocollo crittografico può essere descritto da un sistema di processi concorrenti e l'analisi delle tracce generate da questi sistemi possono essere utilizzate per verificare le proprietà di autenticazione e segretezza dei protocolli. Comunque questo approccio è affetto dal problema dell'esplosione degli stati che produce un insieme di stati e tracce che è tipicamente infinito. Boreale [17, 18] formalizza un modello astratto per rappresentare protocolli che supportano solo crittografia simmetrica e in cui solo variabili e termini atomici possono trovarsi in posizione di chiavi condivise. Propone, inoltre, una semantica simbolica per simulare il comportamento dei protocolli e basa il suo lavoro sull'unificazione producendo modelli compatti di protocolli. Come in altri metodi, l'enumerazione esaustiva di tutte le tracce simboliche produce uno spazio finito degli stati simbolici. Rappresenta uno stato del protocollo come una coppia (σ,Q), chiamata configurazione, dove la traccia σ è una sequenza di azioni di input/output e rappresenta la conoscenza corrente dell'avversario e Q è un processo che descrive il futuro comportamento degli agenti onesti. La semantica simbolica risulta così una relazione tra configurazioni.
Come linguaggio base, Boreale considera una variante dello spi-calculus, ma ritiene che questa scelta non sia vincolante per lo sviluppo della teoria. Il linguaggio non contiene l'operatore di replicazione che renderebbe l'analisi delle tracce indecidibile.
Nel definire il suo modello, assume che sia gli agenti onesti che gli attaccanti seguono le regole di perfect encryption e che la rete di comunicazione è totalmente sotto il controllo dell'ambiente. L'ambiente, quindi, può memorizzare, duplicare, nascondere o rimpiazzare messaggi trasmessi sulla rete. Inoltre può operare in accordo con le regole seguite dagli agenti onesti, creare nuovi nonce, nuove chiavi e nuovi messaggi per accoppiamento, decriptazione, criptazione, o creare messaggi da arbitrarie combinazioni di queste operazioni.

Queste assunzioni rendono il modello del protocollo, che è un grafo di transizione degli stati, infinito. L'idea di Boreale è di rimpiazzare le infinite transizioni risultanti da un'azione di input da una singola transizione simbolica, e di rappresentare il messaggio ricevuto con una variabile. I vincoli sulla variabile sono posti durante l'esecuzione della computazione e prendono la forma di most general unifier. L'analisi dei protocolli è allora equivalente a decidere, per ogni traccia simbolica, se questa può essere risolta, cioè se esiste un'istanziazione delle variabili tale che nella traccia concreta risultante ogni termine ground spedito dall'ambiente è derivabile dalla conoscenza dell'ambiente corrente, usando un sistema deduttivo. Questo è lo stesso che decidere la soddisfacibilità dei vincoli di Huima [53] per una particolare traccia simbolica. Inoltre, l'autore osserva, attraverso alcuni esempi, che si può controllare la consistenza di una traccia simbolica istanziandola gradualmente fino ad ottenere una traccia, e chiama questo processo *raffinamento*. In [18] è data una procedura decisionale per il raffinamento che funziona istanziando progressivamente le variabili della traccia simbolica fino ad ottenere una *forma risolta* in cui ogni termine ricevuto dal processo è derivabile dall'ambiente. L'idea base è che ogni termine simbolico può essere decomposto in un numero finito di componenti irriducibili, dividendo le coppie e decriptando i messaggi se la chiave corretta è conosciuta dall'ambiente. Per ogni messaggio spedito dall'ambiente è possibile

(i) dividere il termine spedito nei suoi termini irriducibili,

(ii) dividere tutti i termini simbolici conosciuti dall'ambiente nei suoi termini irriducibili.

Poiché entrambi gli insiemi sono finiti, il problema dell'analisi della conoscenza simbolica può essere deciso verificando se l'insieme dei termini è incluso, attraverso l'unificazione, nell'insieme conosciuto dall'ambiente. Il processo di raffinamento è non deterministico e potrebbe condurre a molte forme risolte diverse per la stessa traccia simbolica.
Boreale dimostra che la semantica simbolica e quella convenzionale sono in pieno accordo, e presenta un metodo dal quale l'analisi delle tracce può essere portata avanti direttamente su modelli simbolici. Inoltre mostra che questo metodo è completo per una larga classe di protocolli e proprietà ed è ben disposto a verifiche automatiche. In particolare la completezza è dimostrata provando che ogni soluzione delle traccia simbolica è una soluzione di almeno una delle forme risolte prodotte dall'algoritmo.
In un'implementazione pratica, piuttosto che generare l'intero insieme di tracce simboliche di una configurazione e poi controllare le proprietà di sicurezza, Boreale reputa più conveniente unire la fase di generazione delle tracce con quella della verifica delle proprietà. In [19] Boreale e Buscemi presentano STA (Symbolic Trace Analyzer), un ML-tool per l'analisi di protocolli di sicurezza basata sull'analisi presentata. Ciò che distingue STA dai model-checker a stati finiti è il suo uso di tecniche simboliche che evitano di costruire esplicitamente l'intero spazio degli stati del protocollo. Confrontando STA con tecniche a stati finiti, questo analizzatore restituisce:

- modelli di protocolli più accurati,
- incremento di efficienza, sia in termini di memoria occupata che in tempo di esecuzione,

- formalizzazione più diretta del protocollo.

5.4 Fiore e Abadi

Fiore e Abadi [32] sono partiti dal lavoro di Boreale [17] per definire una nuova analisi per la verifica dei protocolli di autenticazione basati su algoritmi crittografici simmetrici. Rappresentano i protocolli come processi espressi in spi-calculus e le proprietà di un protocollo vengono determinate considerando le tracce di computazione del processo che descrive il protocollo. Il metodo di analisi supporta chiavi complesse, ma la correttezza è provata solo per chiavi atomiche. Il loro modello assume che la comunicazione sia globale, cioè che tutti i messaggi siano trasmessi e ricevuti attraverso l'ambiente. L'ambiente, inoltre, è in grado di introdurre nuovi messaggi sfruttando le informazzioni ottenute dai messaggi visti. In questo modo i messaggi che l'ambiente può generare, sono infiniti. Per tale comportamento, l'ambiente è stato identificato con l'agente ostile. Poiché i messaggi conosciuti dall'ambiente sono infiniti, anche l'insieme delle tracce di computazione è infinito, e quindi, per eseguire una verifica automatica, è necessaria una rappresentazione finita di questo insieme.
L'idea base del loro lavoro è considerare le transizioni di input simbolicamente per evitare una struttura con un numero infinito di rami. Definiscono il modello, utilizzato per la verifica della correttezza dei protocolli, attraverso tecniche simboliche in due passi. Nel primo viene considerata una riduzione simbolica di processi, in cui l'input è valutato formalmente, cioè rappresenta il messaggio ricevuto con una variabile, ottenendo così un insieme finito di esecuzioni simboliche. Tuttavia, non tutte le esecuzioni simboliche rappresentano una storia valida del sistema a causa della valutazione simbolica dei messaggi ricevuti. Nel secondo passo si utilizza una procedura simbolica per analizzare la conoscenza dell'ambiente, cioè per determinare quali messaggi possono essere dedotti da messaggi spediti precedentemente all'ambiente. La soluzione della procedura sono vincoli, dati in forma di sostituzioni, tali da rendere ogni termine ricevuto dal processo derivabile dalla conoscenza corrente dell'ambiente. Combinando questi due passi vengono costruiti modelli simbolici finiti costituiti da tutte le esecuzioni simboliche computazionalmente valide del protocollo. Tali modelli finiti possono essere analizzati per la verifica della correttezza dei protocolli.
Fiore e Abadi si differenziano da Boreale per la tecnica utilizzata per raffinare il modello ottenuto applicando la semantica simbolica e per aver distinto la fase di verifica delle proprietà da quella della costruzione del modello. Questa scelta risulta conveniente se, per qualche particolare applicazione, è necessario definire altre proprietà da analizzare. Infatti la nuova proprietà introdotta viene verificata direttamente sul modello senza doverlo ricalcolare.

Parte II

Modelli di Protocolli di Autenticazione

Introduzione

L'autenticazione è fondamentale nella progettazione dei sistemi distribuiti sicuri. L'autenticazione è il processo mediante il quale un agente di un sistema distribuito dimostra la propria identità ed è utilizzata per proteggere i partecipanti di una comunicazione da eventuali attacchi di agenti esterni che vogliono impadronirsi di informazioni riservate. In sistemi distribuiti, l'autenticazione è tipicamente garantita da protocolli, chiamati *protocolli di autenticazione*, la cui analisi è l'obiettivo del nostro lavoro. Gli scopi principali di un protocollo di autenticazione sono garantire l'identità degli agenti e lo scambio di nuove chiavi di sessione segrete per future comunicazioni. Infatti, un protocollo di autenticazione, per terminare con successo, deve garantire i seguenti obiettivi:

Autenticazione: un agente, che si deve autenticare, deve essere sicuro di parlare con l'agente desiderato, cioè bisogna consentire agli agenti coinvolti nella comunicazione di ottenere sufficienti garanzie sull'identità degli altri agenti che partecipano alla comunicazione.

Distribuzione delle chiavi: un agente vuole avere la sicurezza che nessun attaccante possa venire a conoscenza dei suoi segreti.

Per analizzare i protocolli di autenticazione, dovremo tradurre questi obiettivi di alto livello in proprietà di correttezza primitive. Woo e Lam [96] propongono una formalizzazione della correttezza basata su due proprietà primitive di sicurezza, chiamate *correspondence* e *secrecy*. Con correspondence si intende che la comunicazione tra gli agenti in un protocollo di autenticazione deve avvenire seguendo una sequenza di passi fissati, mentre con secrecy si intende che informazioni riservate non devono essere accessibili ad un agente ostile. La proprietà di correspondence è così indirizzata a problemi di autenticazione, mentre quella di secrecy è rivolta a problemi di distribuzioni delle chiavi.
Definiamo ora un modello per rappresentare i protocolli di autenticazione, mettendo in luce le assunzioni che esso comporta e il modo in cui vengono verificate le proprietà di sicurezza. Basandoci sul modello di Dolev-Yao [31] e considerando i protocolli di autenticazione in forma idealizzata, cioè ignorando la maggior parte dei dettagli degli attuali criptosistemi, assumiamo crittografia perfetta. La codifica, la decodifica e altre operazioni sono modellate usando solo le regole algebriche a cui obbediscono. Assumiamo, inoltre, che l'agente ostile controlli la rete di comunicazione e quindi l'ambiente è identificato con l'attaccante. L'insieme dei messaggi conosciuti dall'ambiente è infinito, in quanto l'ambiente conosce i

messaggi che appartengono alla sua conoscenza iniziale, quelli spediti dal processo e i messaggi generati a partire da questi due insiemi. Il numero infinito di messaggi, conosciuti dall'ambiente, porta all'esplosione degli stati del modello concreto, cioè il numero delle storie valide del sistema è infinito. Infatti, ad ogni azione di input, il processo può ricevere qualsiasi messaggio appartenente alla conoscenza dell'ambiente. Per eseguire verifiche automatiche, dobbiamo catturare le informazioni di questa struttura infinita in una finita. A tal fine definiamo un modello finito che rappresenti esattamente le esecuzioni computazionalmente valide del protocollo in due passi, seguendo l'approccio proposto da Fiore e Abadi [32]. Prima consideriamo una riduzione simbolica dei processi, in cui le azioni di input sono valutate formalmente, cioè si introduce una variabile nuova ad ogni azione di input. Il modello ottenuto è si finito, ma contiene alcune esecuzioni che non sono computazionalmente valide. Successivamente forniamo una procedura per stabilire la conoscenza dell'ambiente, cioè determiniamo quali messaggi possono essere derivati dalla conoscenza iniziale dell'ambiente e dai messaggi spediti precedentemente. Abbiamo così gli strumenti per raffinare il modello, ottenuto al passo precedente, cioè siamo in grado di eliminare quei comportamenti che non rappresentano una storia valida del sistema. Combinando questi passi otteniamo così il modello cercato.
Modelliamo i protocolli di autenticazione come processi espressi in un dialetto di spi-calculus. Un processo è definito come la composizione parallela di sottoprocessi sequenziali che descrivono il comportamento di ogni agente coinvolto nel protocollo. Vediamo un esempio di processo considerando il *Wide Mouthed Frog Protocol* [1, 12, 41] nel caso di una singola sessione. Il protocollo è composto dai tre seguenti messaggi:

$$\begin{array}{lcl} \text{Messaggio 1} \quad A \to S & : & \{K_{AB}\}_{K_{AS}} \\ \text{Messaggio 2} \quad S \to B & : & \{K_{AB}\}_{K_{BS}} \\ \text{Messaggio 3} \quad A \to B & : & \{M\}_{K_{AB}} \end{array}$$

L'agente A manda la chiave K_{AB} al server fidato S codificata con la chiave condivisa K_{AS}, e, dopo aver ricevuto e decriptato il messaggio, S spedisce a B la chiave K_{AB} codificata con la chiave condivisa K_{BS}. A questo punto B può ricevere e decodificare il messaggio $\{M\}_{K_{AB}}$ speditogli da A. La sua specifica nel calcolo è data dai seguenti processi:

$$\begin{array}{lcl} P & = & (\nu\ K_{AS})(\nu\ K_{BS})((\nu\ K_{AB})\ A \mid B \mid S\ ) \\ A & = & !\langle\{K_{AB}\}_{K_{AS}}\rangle\ .\ !\langle\{M\}_{K_{AB}}\rangle\ .\ \mathbf{0} \\ S & = & ?(x)\ .\ case\ x\ of\ \{y\}_{K_{AS}}\ in\ !\langle\{y\}_{K_{BS}}\rangle\ .\ \mathbf{0} \\ B & = & ?(x)\ .\ case\ x\ of\ \{y\}_{K_{BS}}\ in\ ?(z)\ .\ case\ z\ of\ \{u\}_y\ in\ \mathbf{0} \end{array}$$

Vengono usati ? e ! rispettivamente per denotare un'azione di input e di output. Inoltre, viene utilizzato *case M of* $\{y\}_k$ *in P* per decriptare i messaggi. Intuitivamente, questo costrutto dice che il processo P viene eseguito se il messaggio M è un ciphertext codificato con la chiave k. Le tracce di computazione di un processo, che descrive un protocollo, possono essere utilizzate per verificare le sue proprietà di sicurezza. Per esempio, analizzando le tracce di computazione di P, si può verificare la proprietà di corrispondenza, cioè stabilire se vengono rispettate le relazioni temporali tra azioni. Nel nostro caso possiamo controllare se B possa venire a conoscenza del messaggio riservato M prima di ricevere il messaggio $\{K_{AB}\}_{K_{BS}}$.

Analizziamo ora un protocollo più semplice per osservare le problematiche che nascono da questo tipo di analisi e per vedere, in modo intuitivo, come risolvere tali problemi. Il protocollo è composto dai seguenti messaggi:

$$\begin{array}{lllll} \text{Messaggio 1} & A \rightarrow B & : & \{M\}_k \\ \text{Messaggio 2} & B \rightarrow A & : & h(M) \end{array}$$

L'agente A manda un messaggio segreto M all'agente B codificato con la chiave condivisa k e, dopo aver ricevuto e decriptato il messaggio, B risponde ad A mandandogli una copia del messaggio codificata con la funzione hash h, così che il messaggio non è rivelato e A sa quello che B ha visto. Questo protocollo è rappresentato dal processo $Q = A \mid B$, dove A e B sono processi sequenziali che rappresentano il comportamento dei rispettivi agenti. Diamo la loro codifica nel calcolo:

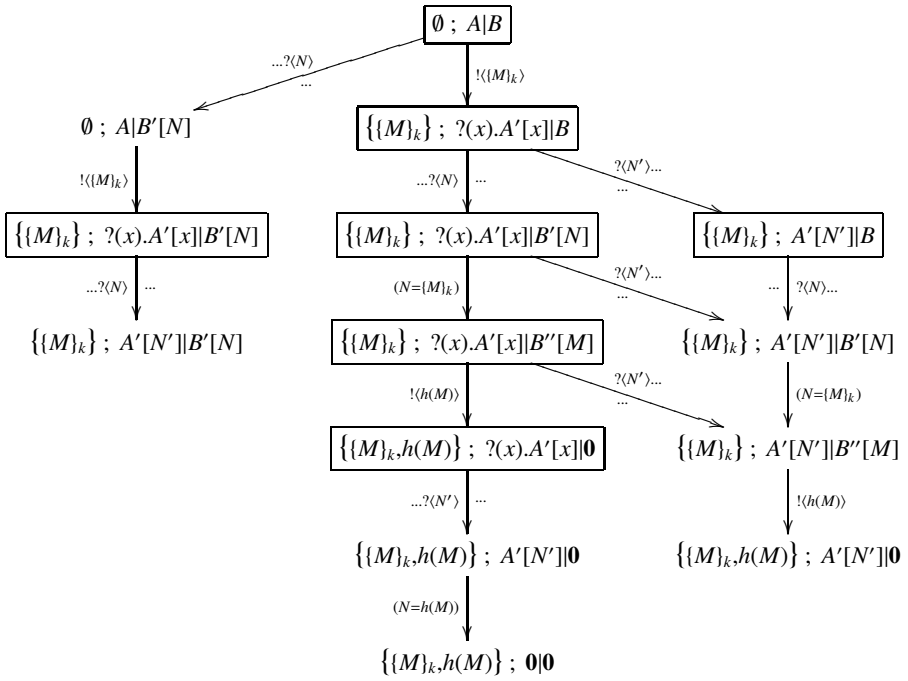
$$\begin{array}{lcl} A & = & !\langle\{M\}_k\rangle \,.\, ?(x) \,.\, A'[x] \\ A'[x] & = & if\ x = h(M)\ then\ \mathbf{0} \\ B & = & ?(x) \,.\, B'[x] \\ B'[x] & = & case\ x\ of\ \{y\}_k\ in\ B''[y] \\ B''[y] & = & !\langle h(y)\rangle \,.\, \mathbf{0} \end{array}$$

dove il costrutto $if\ M = N\ then\ P$ viene utilizzato per testare l'uguaglianza di messaggi. Intuitivamente, questo costrutto dice che il processo P viene eseguito se M e N sono lo stesso messaggio.
È possibile rappresentare l'evoluzione del sistema attraverso un grafo diretto, che identifichiamo con il modello concreto. Un grafo diretto G è una coppia (V,E), dove V è un insieme ed E è una relazione binaria su V. L'insieme V è l'insieme dei nodi di G e i suoi elementi sono chiamati nodi. L'insieme E è l'insieme degli archi di G ed i suoi elementi sono chiamati archi e vengono rappresentati con frecce. Ogni nodo del grafo è etichettato con una coppia, in cui il primo elemento è l'insieme dei messaggi conosciuti attualmente dall'ambiente, o stato di conoscenza, mentre il secondo è lo stato corrente del sistema, cioè il processo da eseguire. Questa coppia rappresenta insieme la conoscenza dell'ambiente e il futuro comportamento del sistema. Gli archi sono etichettati con azioni di input e di output, e con equazioni. Le azioni rappresentano l'evoluzione del processo, cioè le interazioni tra il processo e l'ambiente e sono indicate ripettivamente con $?\langle M\rangle$ e $!\langle M\rangle$, dove M è il messaggio ricevuto o trasmesso dal processo. Per le azioni di input è utilizzata la notazione $...?\langle M\rangle...$ per indicare la presenza di infiniti archi uscenti dal nodo in quanto vanno considerati tutti i messaggi conosciuti dall'ambiente. Le equazioni corrispondono all'esecuzione dei costrutti di if e di $case$ e servono per selezionare quei messaggi ricevuti dalla rete, che consentono al sistema di evolvere. Avendo assunto comunicazione globale, risulta fondamentale l'informazione conosciuta dall'ambiente. La conoscenza dell'ambiente viene utilizzata e aggiornata a seguito di azioni rispettivamente di input e di output.

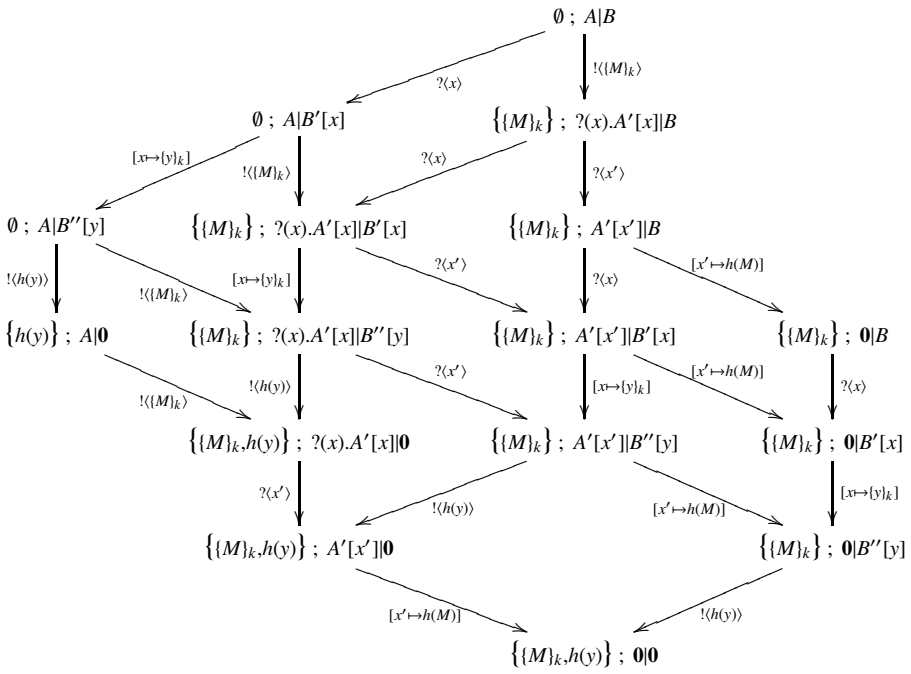
(I) transizioni di input $K\ ;\ P \xrightarrow{?\langle M\rangle} K\ ;\ P'$ sono permesse solo per messaggi M conosciuti dall'ambiente

(O) transizioni di output $K\ ;\ P \xrightarrow{!\langle M\rangle} K'\ ;\ P'$ incrementano la conoscenza dell'ambiente. Si ha infatti che $K' = K \cup \{M\}$.

Vediamo ora il grafo delle transizioni del processo Q.



Perfino il grafo delle computazioni, sopra esposto, è infinito: ogni nodo evidenziato con il box ha infinite transizioni in quanto ad ognuno di essi corrisponde la possibilità di eseguire un'azione di input e i messaggi conosciuti dall'ambiente sono infiniti.
Per eseguire verifiche automatiche, è necessario catturare le informazioni di questa struttura infinita in una finita e ciò può essere fatto usando tecniche simboliche. L'idea è quella di evitare una struttura a ramificazione infinita considerando transizioni di input simboliche, cioè viene introdotta una variabile nuova ad ogni occorrenza di input. Quindi, non si considerano i messaggi che possono essere effettivamente ricevuti dall'ambiente, ma vengono determinati solo i vincoli per cui il processo può evolvere. Come già fatto per il modello concreto, rappresentiamo ora il modello astratto con un grafo diretto, chiamato grafo simbolico delle transizioni. Etichettiamo ogni nodo del grafo con una coppia. Il primo elemento è lo stato di conoscenza dell'ambiente, mentre il secondo è lo stato corrente del sistema. Gli archi sono etichettati con azioni di input e di output e con i vincoli con i quali il processo può evolvere. Questi vincoli sono dati in forma di sostituzioni e sono ottenuti eseguendo i costrutti di *if* e di *case*. Scriviamo $[x \mapsto M]$ per indicare che il messaggio M viene sostituito alle occorrenze libere della variabile x nel processo che descrive il futuro comportamento del sistema. Indichiamo le azioni di input e di output rispettivamente con $?\langle x\rangle$ e $!\langle M\rangle$, dove x è una variabile nuova, cioè che non è mai stata usata prima, e M è il messaggio trasmesso dal processo.
Vediamo ora il grafo simbolico delle transizioni del processo Q.



A differenza di quanto accade nel grafo delle transizioni, in quello simbolico il processo converge in un unico stato finale. Comunque, queste strutture simboliche sono ridondanti e bisogna prestare attenzione quando si analizzano. Infatti, se confrontiamo i due grafi, si nota che alcuni cammini del grafo simbolico non hanno un cammino corrispondente nel grafo delle transizioni.

Per verificare proprietà di segretezza, estendiamo il protocollo in modo da considerare la presenza di *osservatori*, cioè agenti che sono sempre in ascolto dell'ambiente e perciò sono in grado di intercettare tutti i messaggi conosciuti dall'ambiente. Se vengono a conoscenza di qualche segreto, restituiscono un errore. Gli osservatori non devono essere confusi con gli attaccanti che sono impliciti nel modello computazionale. Per controllare se un messaggio riservato M viene rivelato, consideriamo il processo, che modella il protocollo, in parallelo con il processo che modella l'osservatore, dove l'osservatore O è dato dal processo

$$\begin{aligned} O &= ?(x)\ .\ O'[x] \\ O'[x] &= if\ x = M\ then\ !\langle error\rangle\ .\ \mathbf{0} \end{aligned}$$

Se nessuna traccia di computazione contiene l'azione di output $!\langle error\rangle$, siamo sicuri che l'informazione riservata non viene rivelata. Calcolando il grafo delle transizioni del processo con l'osservatore, si può notare che, negli esempi precedenti, non c'è nessun cammino del grafo che contiene $!\langle error\rangle$. Questo non vale nel grafo simbolico, dove, nel caso del Wide Mouthed Frog Protocol, troviamo l'esecuzione simbolica:

$$
\begin{aligned}
\emptyset \,;\, P \mid O \quad &\xrightarrow{!\langle \{K_{AB}\}_{K_{AS}} \rangle} \quad \{K_{AB}\}_{K_{AS}} \,;\, !\{M\}_{K_{AS}} \,.\, \mathbf{0} \mid B \mid S \mid O \\
&\xrightarrow{?\langle x \rangle} \quad \{K_{AB}\}_{K_{AS}} \,;\, !\{M\}_{K_{AS}} \,.\, \mathbf{0} \mid B \mid S \mid O'[x] \\
&\xrightarrow{[x \mapsto M]} \quad \{K_{AB}\}_{K_{AS}} \,;\, !\{M\}_{K_{AS}} \,.\, \mathbf{0} \mid B \mid S \mid !\langle error \rangle \,.\, \mathbf{0} \\
&\xrightarrow{!\langle error \rangle} \quad \{K_{AB}\}_{K_{AS}}, \langle error \rangle \,;\, !\{M\}_{K_{AS}} \,.\, \mathbf{0} \mid B \mid S \mid \mathbf{0}
\end{aligned}
$$

Questa esecuzione simbolica del protocollo non ha una controparte nel modello concreto. Il nostro scopo è stabilire quali cammini nel grafo simbolico delle transizioni hanno un cammino corrispondente nel grafo delle transizioni. Vogliamo quindi, raffinare il modello astratto eliminando i controesempi spuri, cioè quei comportamenti del modello astratto che non hanno una controparte nel modello concreto. Affrontiamo questo problema definendo un algoritmo che calcola i vincoli per cui un messaggio è deducibile dalla conoscenza dell'ambiente. Tale procedura consente di determinare quali cammini del grafo simbolico abbiano una traccia corrispondente nel modello concreto.

Il *Capitolo 6* definisce la sintassi del calcolo, utilizzata per rappresentare i protocolli da analizzare. Dà inoltre una riduzione semantica, che formalizza le regole di interazione tra processi e ambiente, e definisce infine il concetto corrispondente di riduzione simbolica, che permette alle transizioni di input di essere valutate formalmente.
Il *Capitolo 7* fornisce un'analisi algoritmica della conoscenza dell'ambiente, dando una procedura decisionale per il controllo della conoscenza, ristretta a messaggi ground, e una procedura simbolica per l'analisi della conoscenza per messaggi arbitrari.
Il *Capitolo 8* combina la semantica simbolica del Capitolo 6 e la procedura simbolica del Capitolo 7 per ottenere modelli simbolici. Infine mostra la correttezza e la completezza del modello ottenuto, cioè che questo rappresenta esattamente le esecuzioni computazionalmente valide del protocollo.

Capitolo 6

Il Calcolo e Riduzione Simbolica

Definiamo ora la sintassi del calcolo necessaria per rappresentare i protocolli. La sintassi del calcolo è costituita da due entità: i messaggi e i processi. I messaggi sono le informazioni scambiate dai processi, i processi rappresentano lo schema con cui gli agenti comunicano tra loro. Diamo, inoltre, una semantica che formalizza le regole di interazione tra processi e ambiente, e un sistema deduttivo che consente di determinare la conoscenza dell'ambiente a partire dai messaggi scambiati tra gli agenti. Definiamo, infine, la riduzione simbolica che consente alle transizioni di input di essere valutate formalmente e che rappresenta il primo passo per ottenere modelli finiti utilizzabili per la verifica automatica delle proprietà di sicurezza dei protocolli analizzati.

6.1 Sintassi

Introduciamo un dialetto dello spi-calculus [1], cioè un π-calculus [66, 82] esteso con primitive di crittografia, progettato per descrivere ed analizzare protocolli crittografici. Detto dialetto risulta un mezzo adeguato per rappresentare il flusso di informazioni e quindi per ottenere un modello utile per la verifica delle proprietà di sicurezza. Nella costruzione del modello si assume che i meccanismi crittografici non siano vulnerabili, in modo che gli attacchi possono essere portati solo alla struttura del protocollo e ciò porta alle assunzioni sulla crittografia.

- Il solo modo per decriptare un messaggio codificato è conoscere la corrispondente chiave di decodifica.
- Un messaggio criptato non rivela la chiave che è stata utilizzata per codificarlo.
- I messaggi devono avere sufficiente ridondanza in modo che l'algoritmo di decodifica determini se un messaggio cifrato era codificato con la chiave attesa.
- La funzione hash è perfetta. Questo presuppone di considerare tale funzione collision free, cioè una funzione in cui non è possibile trovare due elementi distinti con il medesimo valore di hash, e one-way, cioè una funzione non invertibile.

Assumiamo che la comunicazione sia globale, cioè possa avvenire solo attraverso l'ambiente, e che gli attaccanti, che sono modellati implicitamente, siano dotati di capacità illimitate per manipolare i messaggi. Per questo motivo, nelle azioni di input e di output, non consideriamo i canali attraverso i quali vengono spediti o ricevuti messaggi, e quindi trascuriamo le regole di comunicazione tra sottoprocessi presenti nel π-calculus e nel spi-calculus standard. Inoltre l'analisi è ristretta a processi finiti e ciò implica che il numero di agenti, che partecipano al protocollo, è finito e che non si considera la replicazione. Introduciamo ora formalmente gli oggetti necessari per modellare i protocolli.

Definizione 6.1.1 *Le espressioni formali atomiche sono definite come le categorie disgiunte di* variabili $\mathcal{V}$*, di* nomi $\mathcal{N}$*, di* token $\mathcal{T}$ *e di* coppie di chiavi $\mathcal{K}$*, dove* $\mathcal{V}$*,* $\mathcal{N}$*,* $\mathcal{T}$ *e* $\mathcal{K}$ *sono insiemi infiniti numerabili. Gli elementi della categoria* $\mathcal{K}$ *sono formati da due componenti: una chiave pubblica e una chiave privata. Le categorie* $\mathcal{K}^+$ *e* $\mathcal{K}^-$ *denotano rispettivamente le categorie delle componenti pubbliche e private di* $\mathcal{K}$*. Le categorie* $\mathcal{T}$ *e* $\mathcal{K}^+$ *rappresentano la conoscenza iniziale dell'ambiente.*

Definizione 6.1.2 *La grammatica per i* messaggi *è definita nel seguente modo:*

M,N	::=	x	*variabili*
		n	*nomi*
		τ	*token*
		k^+	*chiave pubblica*
		k^-	*chiave privata*
		(M,N)	*coppia*
		$\{M\}_N$	*codifica a chiave condivisa*
		$\{\!\|M\|\!\}_k$	*codifica a chiave pubblica*
		$h(M)$	*funzione hash*

Vediamo ora in dettaglio il significato di alcuni messaggi, illustrando le assunzioni che comportano.

- Il messaggio $\{M\}_N$ rappresenta il testo cifrato ottenuto codificando il messaggio M con la chiave N attraverso un algoritmo crittografico simmetrico. Si consente l'uso di messaggi arbitrari come chiavi condivise e questo permette di modellare un protocollo dove una chiave è generata utilizzando una funzione crittografica su un nuovo segreto condiviso.

- il messaggio $\{\!|M|\!\}_k$, dove k è una chiave pubblica, rappresenta il risultato della codifica a chiave pubblica di M con k. Sistemi a crittografia a chiave pubblica sono basati su coppie di chiavi: in ogni coppia una delle chiavi è privata ed è in possesso di un singolo agente, mentre l'altra chiave è pubblica. Ogni agente può codificare un messaggio usando la chiave pubblica; solo l'agente che possiede la chiave privata può decriptare il messaggio. Assumiamo che nessuna chiave possa essere ricavata dalle altre e che il solo modo per decriptare un messaggio codificato è conoscere la corrispondente chiave privata. Se k rappresenta una coppia di chiavi, allora k^+ rappresenta la sua metà pubblica e k^- la sua metà privata.

- Il messaggio $h(M)$ rappresenta la codifica attraverso la funzione hash del messaggio M. La mancanza di equazioni del tipo $h(M) = h(M')$ corrisponde ad assumere che la funzione hash h è *collision free*.

Introduciamo ora la nozione di complessità dei messaggi utilizzata in seguito per dimostrare alcuni risultati di questa tesi.

Definizione 6.1.3 *La* complessità *di un messaggio è definita nel seguente modo:*

$$\begin{array}{ll} Comp(M) & = 0 \quad se\ M \in \mathcal{V} \cup \mathcal{N} \cup \mathcal{T} \cup \mathcal{K}^+ \cup \mathcal{K}^- \\ Comp((M,N)) & = 1 + Comp(M) + Comp(N) \\ Comp(\{M\}_N) & = 1 + Comp(M) + Comp(N) \\ Comp(\{\!|M|\!\}_k) & = 1 + Comp(M) \\ Comp(h(M)) & = 1 + Comp(M) \end{array}$$

Inoltre, per un insieme di messaggi K, $Comp(K) = \sum_{M \in K} Comp(M)$.

Definizione 6.1.4 *La grammatica per i* processi *è definita nel seguente modo:*

P,Q	::=	$\boldsymbol{0}$	*nil*
		$P\|Q$	*composizione parallela*
		$?(x).P$	*input*
		$!\langle M\rangle.P$	*output*
		$if\ M = N\ then\ P$	*matching*
		$let\ M = (x,y)\ in\ P$	*pair splitting*
		$case\ M\ of\ \{y\}_N\ in\ P$	*decodifica con chiave condivisa*
		$case\ M\ of\ \{\!|y|\!\}_{k^-}\ in\ P$	*decodifica con chiave privata*

Vediamo ora in dettaglio la semantica dei costrutti con cui vengono costruiti processi complessi . Scriviamo $P[x \mapsto M]$ per indicare il rimpiazzamento di ogni occorrenza libera di x nel processo P con il messaggio M.

- Il processo vuoto **0** non fa niente.
- La composizione $P|Q$ si comporta come i processi P e Q che vengono eseguiti in parallelo. Nel nostro modello questi processi possono comunicare solo con l'ambiente.
- Il processo di input $?(x).P$ è pronto a ricevere dall'ambiente. Se avviene una interazione in cui viene ricevuto il messaggio N, allora il processo si comporta come $P[x \mapsto N]$. Notiamo che la variabile x è legata in P.
- Il processo di output $!\langle M\rangle.P$ è pronto a spedire il messaggio M all'ambiente. Se avviene una interazione, allora viene spedito M e viene eseguito P.
- Il processo $if\ M = N\ then\ P$ si comporta come P se M e N sono lo stesso messaggio, altrimenti non fa niente.

- Il processo *let* $M = (x,y)$ *in* P si comporta come $P[x \mapsto N][y \mapsto L]$ se il messaggio M è la coppia (N,L), altrimenti il processo P viene scartato. Notiamo che le variabili x e y sono legate in P.

- Il processo *case* M *of* $\{y\}_N$ *in* P cerca di decriptare il messaggio M con la chiave N. Se M è un testo codificato della forma $\{L\}_N$, allora il processo si comporterà come $P[y \mapsto L]$, altrimenti il processo P viene scartato. Notiamo che la variabile y è legata in P.

- Il processo *case* M *of* $\{\!|y|\!\}_{k^-}$ *in* P cerca di decriptare il messaggio M con la chiave privata k^-. Se M è un testo codificato della forma $\{\!|L|\!\}_{k^+}$, allora il processo si comporterà come $P[y \mapsto L]$, altrimenti il processo P viene scartato. Notiamo che la variabile y è legata in P.

Nel calcolo è assente un costrutto per ricavare il messaggio M da $h(M)$ e ciò corrisponde all'assunzione che la funzione hash non sia invertibile. L'unica maniera per controllare se un messaggio M è uguale al messaggio N codificato con la funzione hash è calcolare $h(N)$ e verificare che sia uguale a M.
Inoltre, nel linguaggio manca la *restrizione* e la *replicazione* presenti invece nel π-calculus e nello spi-calculus standard. La replicazione è stata omessa perché abbiamo ristretto l'analisi a processi finiti, e in sua assenza, la restrizione può essere omessa senza perdita di generalità. Presentiamo ora alcune definizioni che consentono di definire una sottoclasse di messaggi e processi utilizzati nel seguito della tesi.

Definizione 6.1.5 *Un messaggio è detto essere* ground *se non ha variabili.*

Definizione 6.1.6 *Un processo è detto essere* chiuso *se non ha variabili libere.*

6.2 Semantica delle Riduzioni

Un protocollo, secondo la semantica, è rappresentato dall'insieme delle sue esecuzioni. In questa sezione restringiamo l'analisi ai soli messaggi ground e diamo la semantica delle riduzioni formalizzando le regole di interazione tra processi e l'ambiente. Questa semantica, quindi, è una relazione tra configurazioni che rappresentano la conoscenza corrente dell'ambiente e lo stato del sistema, e utilizza un sistema deduttivo che permette di determinare la conoscenza dell'ambiente a partire dai messaggi scambiati tra gli agenti. Inoltre, consente di costruire un modello che rappresenta l'insieme di tutte le esecuzioni possibili del protocollo, che chiamiamo storie valide del sistema. Presentiamo alcune definizioni che consentono di rappresentare una storia valida del sistema.

Definizione 6.2.1 *Una* sequenza di interazioni *per la semantica delle riduzioni è una lista*

$$t = N_1 \cdot \ldots \cdot N_k$$

dove $N_i \in \{!\langle M\rangle, ?\langle M\rangle\}$ *in cui* M *è un messaggio ground. Le espressioni* $!\langle M\rangle, ?\langle M\rangle$ *sono chiamate rispettivamente azioni di output e di input.*

Definizione 6.2.2 *Una* configurazione *è una coppia*

$$t \; ; \; P$$

dove t è una sequenza di interazioni e P è un processo.

Le configurazioni sono dunque coppie consistenti in una sequenza di azioni e in un processo: la sequenza memorizza la storia delle interazioni tra il processo e l'ambiente, il processo descrive il futuro comportamento del sistema.
Definiamo ora un sistema deduttivo per decidere se l'ambiente è in grado di derivare un messaggio a partire dalla sua conoscenza. Un sistema deduttivo è costituito da regole di inferenza, ognuna di esse specifica come, sotto certe condizioni, da un insieme di messaggi, detti *premesse*, sia possibile derivare la *conclusione*.
La relazione di inferenza è data attraverso un sistema deduttivo, nello stile della deduzione naturale, che utilizza espressioni di sequenti $K \vdash M$, dove K è un insieme di messaggi, che rappresenta la conoscenza dell'ambiente, e M è il messaggio che vogliamo derivare da questo insieme.

Definizione 6.2.3 *Si dice* sequente o (giudizio) *un'espressione $K \vdash M$, dove K è un insieme finito di messaggi ed M è un messaggio.*

Un sistema per rappresentare una regola di inferenza è scrivere tutte le premesse sopra una riga orizzontale e la conclusione immediatamente sotto. L'elenco completo delle regole di inferenza è dato in Tabella 6.1. Il nostro sistema deduttivo prevede due tipi di regole: regole di introduzione e regole di eliminazione. Intuitivamente, le prime determinano come derivare un messaggio, mentre le altre determinano cosa è derivabile da un messaggio. Nel nostro sistema deduttivo identifichiamo le regole di introduzione con le regole (*PAIR*), (*SKENC*), (*PKENC*) e (*HASH*), mentre le regole di eliminazione con le regole ($PROJ_i$), (*SKDEC*) e (*PKDEC*). Gli assiomi del sistema deduttivo sono identificati, invece, con le regole (*TOK*), (*PUB*) e (*AX*).
Le regole di inferenza consentono di rappresentare la prova della validità di un giudizio attraverso alberi. Informalmente, un albero è un grafo non orientato privo di cicli con un nodo distinto, la radice, e con un unico *path* (percorso) dalla radice ad ogni nodo. Un albero di giudizi è una particolare istanza di un albero avente i nodi etichettati con giudizi. La definizione formale è la seguente.

Definizione 6.2.4 *Un* albero di giudizi *è un albero Γ con:*

- *Un insieme di nodi.*
- *Un insieme di archi orientati. Ogni arco è rappresentabile come una coppia ordinata di nodi.*
- *Un unico nodo, detto radice dell'albero, che non ha predecessori.*
- *Un insieme di nodi foglie (o nodi terminali), cioè nodi senza discendenti.*

$$(TOK)\ \frac{}{K \vdash M}\ M \in \mathcal{T}$$

$$(PUB)\ \frac{}{K \vdash M}\ M \in \mathcal{K}^+$$

$$(AX)\ \frac{}{K \vdash M}\ M \in K \backslash \{\mathcal{T} \cup \mathcal{K}^+\}$$

$$(PAIR)\ \frac{K \vdash M \qquad K \vdash N}{K \vdash (M,N)}$$

$$(PROJ_1)\ \frac{K \vdash (M,N)}{K \vdash M}$$

$$(PROJ_2)\ \frac{K \vdash (M,N)}{K \vdash N}$$

$$(SKENC)\ \frac{K \vdash M \qquad K \vdash N}{K \vdash \{M\}_N}$$

$$(SKDEC)\ \frac{K \vdash \{M\}_N \qquad K \vdash N}{K \vdash M}$$

$$(PKENC)\ \frac{K \vdash M}{K \vdash \{\!|M|\!\}_k}\ k \in \mathcal{K}^+$$

$$(PKDEC)\ \frac{K \vdash \{\!|M|\!\}_{k^+} \qquad K \vdash k^-}{K \vdash M}$$

$$(HASH)\ \frac{K \vdash M}{K \vdash h(M)}$$

Tabella 6.1. Sistema deduttivo di $K \vdash M$

- *Un insieme di giudizi, con cui etichetteremo i nodi.*

Definizione 6.2.5 *Dato un albero di giudizi Γ, un* sottoalbero *è costituito da un generico nodo n di Γ e da tutti i suoi discendenti. Il nodo n è la radice del sottoalbero.*

Alla definizione formale si preferisce la più intuitiva notazione grafica

$$\frac{\Delta_1 \qquad \Delta_2}{\delta}$$

per indicare un albero di giudizi Γ avente radice δ e sottoalberi Δ_1 e Δ_2.

Definizione 6.2.6 *Un* albero di deduzione *è un albero in cui i giudizi che etichettano ogni nodo sono derivati dai giudizi che etichettano i nodi connessi ad esso tramite una valida applicazione delle regole di inferenza di Tabella 6.1.*

Definizione 6.2.7 *Un* albero di prova *è un albero di deduzione in cui le foglie sono ottenute attraverso l'applicazione delle regole* (*TOK*)*,* (*PUB*) *e* (*AX*)*.*

Il sitema deduttivo di Tabella 6.1 stabilisce che l'ambiente può derivare un messaggio da quelli ricevuti precedentemente e dalla sua conoscenza iniziale, cioè dai token e dalle chiavi pubbliche. Chiamiamo ora *deduzione* (o *derivazione*) un albero di prova e diciamo che un giudizio è *deducibile* (o *derivabile*) se esiste una deduzione per quel giudizio.
Osservando le regole date in Tabella 6.1 possiamo notare la differenza tra la regola (*SKENC*) e (*PKENC*). Il motivo per cui la regola per la crittografia a chiave pubblica non richiede la chiave per codificare il messaggio deriva dall'assunzione che l'insieme delle chiavi pubbliche appartenga alla conoscenza iniziale dell'ambiente. Osserviamo ora alcune proprietà comuni a certi sistemi deduttivi di cui gode anche il nostro sistema.

Proposizione 6.2.8 *Siano M e N messaggi, e K e K' insiemi di messaggi. Se $K \vdash M$ e $\{M\} \cup K' \vdash N$ allora $K \cup K' \vdash N$.*

In altre parole, se possiamo dedurre un messaggio M da un insieme K, e questo messaggio è utilizzato insieme ai messaggi di K', per derivare un messaggio N, possiamo sostituire il messaggio M con K. Infatti l'informazione in M è contenuta anche in K. Considerando il caso particolare, in cui l'insieme K' coincide con K, si ottiene che, se $K \vdash M$ e $\{M\} \cup K \vdash N$, allora $K \vdash N$. Iterando il risultato così ottenuto, si dimostra la seguente proprietà.

Proposizione 6.2.9 *Siano M e N messaggi, e K e K' insiemi di messaggi. Se, per ogni $M \in K'$, $K \vdash M$ e $K' \vdash N$ allora $K \vdash N$.*

Tale risultato afferma che se, da un insieme di messaggi K deduciamo i messaggi che appartengono ad un insieme K', allora da K deriviamo tutti i messaggi derivabili da K'. Nel seguito della tesi utilizziamo la notazione $K \vdash K'$ per indicare che dall'insieme K deduciamo tutti i messaggi di K'.
L'insieme dei messaggi conosciuti dall'ambiente è sempre infinito poiché il giudizio $K \vdash M$ vale per tutti i messaggi M costruiti a partire dai messaggi in K, dai token e delle chiavi pubbliche. Questo è il motivo per cui le storie valide del sistema sono infinite, infatti ad ogni azione di input il processo può ricevere uno dei messaggi conosciuti dall'ambiente. Poniamo ora l'attenzione all'insieme dei messaggi che il processo ha trasmesso all'ambiente.

Definizione 6.2.10 *Sia t una sequenza di interazione, lo* stato di conoscenza *dell'ambiente $O(t)$ è definito come l'insieme dei messaggi che occorrono nelle azioni di output di t.*

Lo stato di conoscenza è dato dall'insieme dei messaggi che l'ambiente ha acquisito durante l'esecuzione del protocollo. Sapere quali sono i messaggi derivabili dall'ambiente è fondamentale per definire una relazione in grado di costruire una storia valida del sistema.

Definizione 6.2.11 *Definiamo la relazione di* riduzione

$$t\ ;\ P \longrightarrow t'\ ;\ P'$$

come una relazione tra configurazioni che sono formate da una sequenza di messaggi di input e di output e un processo. Diamo in Tabella 6.2 le regole per la semantica delle riduzioni. Denotiamo con $\longrightarrow^*$ la chiusura riflessiva e transitiva di $\longrightarrow$.

Questa relazione comprende la conoscenza corrente dell'ambiente e lo stato del sistema determinandone l'evoluzione. Sia la regola di input che quella di output memorizzano le inter-

(IN)	$t\ ;\ ?(x).P \longrightarrow t \cdot ?\langle M\rangle\ ;\ P[x \mapsto M]$ per M ground e $O(t) \vdash M$ derivabile
(OUT)	$t\ ;\ !\langle M\rangle.P \longrightarrow t \cdot !\langle M\rangle\ ;\ P$
$(SPLIT)$	$t\ ;\ let\ (M,N) = (x,y)\ in\ P \longrightarrow t\ ;\ P[x \mapsto M, y \mapsto N]$
$(SKCASE)$	$t\ ;\ case\ \{M\}_N\ of\ \{x\}_N\ in\ P \longrightarrow t\ ;\ P[x \mapsto M]$
$(PKCASE)$	$t\ ;\ case\ [\![M]\!]_{k^+}\ of\ [\![x]\!]_{k^-}\ in\ P \longrightarrow t\ ;\ P[x \mapsto M]$
$(MATCH)$	$t\ ;\ if\ M = M\ then\ P \longrightarrow t\ ;\ P$
(PAR)	$\dfrac{t\ ;\ P \longrightarrow t'\ ;\ P'}{t\ ;\ P\|Q \longrightarrow t'\ ;\ P'\|Q}$ $\quad$ $\dfrac{t\ ;\ Q \longrightarrow t'\ ;\ Q'}{t\ ;\ P\|Q \longrightarrow t'\ ;\ P\|Q'}$

Tabella 6.2. Regole per la semantica delle riduzioni

azioni con l'ambiente, inoltre, la regola di input è ristretta ai messaggi conosciuti all'ambiente. Osserviamo in Tabella 6.2 come la regola di input dipende dalla relazione $O(t) \vdash M$ che stabilisce se un messaggio M è derivabile da $O(t)$, cioè dallo stato di conoscenza dell'ambiente. Infatti, il processo può ricevere un messaggio dall'ambiente solo se il messaggio M appartiene alla sua conoscenza o può essere dedotto da questa applicando le regole di Tabella 6.1. Le regole (*SPLIT*), (*SKCASE*), (*PKCASE*) e (*MATCH*) vengono utilizzate per selezionare i messaggi che permettono al sistema di evolvere. La regola (*PAR*) definisce come i processi P e Q possano interagire solo con l'ambiente, indipendentemente uno dall'altro.

Rappresentiamo ora l'insieme delle storie valide di un sistema con un grafo diretto, etichettando, a differenza di quanto già presentato, ogni nodo con una configurazione. Chiamiamo tale rappresentazione modello concreto. In questo modo non solo memorizziamo lo stato di conoscenza dell'ambiente, ma anche l'ordine con cui sono avvenute le azioni. Utilizziamo le transizioni definite in Tabella 6.2 come archi del grafo. Le transizioni di input e di output sono etichettate rispettivamente con $?\langle M\rangle$ e $!\langle M\rangle$, dove M è il messaggio ricevuto o trasmesso dal processo. Gli altri archi corrispondono all'esecuzione delle regole di (*SPLIT*),

(*SKCASE*), (*PKCASE*) e (*MATCH*) e determinano quali messaggi permettono al sistema di evolvere.
Introduciamo ora alcuni risultati che consentono di osservare come la semantica costruisce sequenze di interazioni che costituiscono il modello concreto. Prima di introdurre formalmente questi risultati presentiamo alcune definizioni che ne semplificano la descrizione.

Definizione 6.2.12 *Una* run ground *è una sequenza di messaggi ground di input e di output. I messaggi di input sono rappresentati nella sequenza con* $?\langle M\rangle$*, mentre quelli di output con* $!\langle M\rangle$*, dove M identifica rispettivamente il messaggio ground che viene ricevuto o spedito dal processo.*

Non tutte le run ground di un processo rappresentano una storia valida del sistema a causa della valutazione degli input. Infatti potrebbero essere ricevuti messaggi che non sono conosciuti dall'ambiente. Introduciamo così un nuovo concetto che consente di definire una storia valida del sistema.

Definizione 6.2.13 *Una run ground t è una* traccia ground *se, ogni volta che* $t = t' \cdot ?\langle M\rangle \cdot s$*, il giudizio* $O(t') \vdash M$ *è derivabile.*

Lemma 6.2.14 *Le sequenze di interazioni, ottenute applicando le regole di Tabella 6.2, sono tracce ground.*

Una traccia risulta essere una sequenza di eventi che rappresenta una storia valida del sistema e può essere vista come un cammino nel grafo delle transizioni. Per la verifica delle proprietà di sicurezza non interessa studiare solo una possibile traccia del protocollo, ma dobbiamo considerarle tutte. Introduciamo, quindi, un insieme di esecuzioni che rappresenti tutte e sole le storie valide del sistema.

Definizione 6.2.15 *Dato un processo chiuso P, definiamo l'insieme* $Trace(P)$ *come la chiusura dei prefissi di sequenze di interazioni, ottenute attraverso l'applicazione delle regole di Tabella 6.2, cioè*

$$Trace(P) = \{\, t \mid \epsilon \;;\; P \longrightarrow^{*} t \;;\; P' \}$$

Lemma 6.2.16 *L'insieme* $Trace(P)$ *è costituito da tracce ground.*

Quindi, $Trace(P)$ è l'insieme di tutte le tracce del processo, cioè rappresenta tutte le storie valide del sistema. Gli elementi di $Trace(P)$ sono così i cammini che si hanno nel grafo delle transizioni.

6.3 Riduzione Simbolica

Dato che il numero di messaggi conosciuti dall'ambiente è infinito, anche l'insieme delle tracce ground risulta infinito e quindi il modello concreto non può essere utilizzato per una verifica automatica delle proprietà di sicurezza. È necessario definire un nuovo modello che sia finito e che sia in grado di rappresentare esattamente le informazioni contenute in quello

concreto. Tale struttura finita è ottenuta in due fasi attraverso tecniche simboliche seguendo l'approccio di Fiore e Abadi [32]. Il primo passo introduce una semantica simbolica per processi necessaria per costruire un modello finito. In tale semantica le variabili assumono un ruolo fondamentale, infatti le azioni di input sono valutate formalmente, cioè viene introdotta una variabile nuova, che rappresenta il messaggio ricevuto, ad ogni azione di input. Inoltre la semantica calcola i vincoli per cui il sistema può evolvere senza però prendere in considerazione i messaggi conosciuti dall'ambiente. Il modello ottenuto non può essere utilizzato per verificare le proprietà di sicurezza in quanto contiene esecuzioni simboliche che non hanno una controparte nel modello concreto. È compito del secondo passo dell'analisi eliminare tali comportamenti in modo da ottenere un modello in grado di rappresentare esattamente le storie valide del sistema.
Introduciamo ora alcune definizioni che consentono di semplificare la descrizione di alcuni risultati che illustriamo nel seguito di questa tesi.

Definizione 6.3.1 *La funzione Vars è una funzione da termini a variabili. Dato un termine M, $Vars(M)$ restituisce l'insieme delle variabili che appaiono in M.*

Definizione 6.3.2 *La funzione IVars è una funzione da insiemi di termini a variabili. Dato un insieme di termini $\{M_1,...,M_l\}$, $IVars(\{M_1,...,M_l\})$ restituisce l'insieme delle variabili che occorrono nei termini $M_1,...,M_l$, cioè $IVars(\{M_1,...,M_l\}) = Vars(M_1) \cup ... \cup Vars(M_l)$.*

Nella riduzione simbolica non consideriamo i messaggi che possono essere effettivamente ricevuti dall'ambiente, ma stabiliamo solo i vincoli per i quali il processo può evolvere. Rappresentiamo questi vincoli come sostituzioni.

Definizione 6.3.3 *Una* sostituzione *è una funzione da variabili a termini. Una sostituzione si presenta nella forma $[x_1 \mapsto M_1,...,x_n \mapsto M_n]$ con le variabili $x_1,...,x_n$ distinte tra loro e $x_i \neq M_i$ per $i = 1,...,n$. Date le sostituzioni R_1 e R_2, scriviamo $R_1@R_2$ per indicare la loro concatenazione.*

I vincoli che permettono al sistema di evolvere sono calcolati attraverso il *pattern matching*. Analizziamo ora la procedura di unificazione. Due termini per essere unificati, vengono confrontati. Se entrambi i termini sono ground, cioè non contengono variabili, l'unificazione ha successo solo se i termini sono uguali, altrimenti fallisce. Se uno dei termini è una variabile allora questa viene legata all'altro termine che potrebbe essere un qualsiasi termine, e l'unificazione ha successo. Se entrambi i termini sono composti allora ogni coppia di sottotermini viene unificata ricorsivamente e l'unificazione ha luogo solo se tutti i sottotermini sono unificati. Se l'unificazione ha successo, il suo risultato è un insieme di variabili legate, conosciuto come unifier. L'unifier di un insieme di espressioni è così un insieme di sostituzioni di termini tali da rendere tutte le espressioni uguali. La definizione formale è la seguente.

Definizione 6.3.4 *Una sostituzione μ si dice essere un* unifier *di una lista di equazioni di messaggi E, se $M[\mu] = N[\mu]$ per ogni $M = N$ in E.*

Comunque i vincoli, che andremo a definire, non devono essere troppo generali altrimenti si astrae in maniera eccessiva il comportamento del processo rendendo inutile l'analisi. Inoltre, i vincoli non devono essere troppo precisi altrimenti i costi computazionali non giustificano l'astrazione. Introduciamo così il concetto di most general unifier che garantisce di avere un'astrazione in grado di catturare completamente il comportamento del processo senza rendere i costi di computazione eccessivi.

Definizione 6.3.5 *Un unifier è detto essere un* most general unifier (mgu) *se ogni unifier è un'istanza di esso. Scriviamo mgu*$\{M = N\}$ *per indicare il most general unifier di M e N.*

Definiamo ora il concetto di riduzione simbolica. Questa nozione corrisponde alla semantica delle riduzioni data per il modello concreto e consente di costruire le sequenze di interazioni che costituiscono il modello astratto.

Definizione 6.3.6 *Una* sequenza di interazioni *per riduzioni simboliche è una lista*

$$s = N_1 \cdot \ldots \cdot N_k$$

dove $N_i \in \{!\langle M\rangle, ?\langle M\rangle\}$ *in cui M è un messaggio arbitrario. Le espressioni* $!\langle M\rangle, ?\langle M\rangle$ *sono chiamate rispettivamente azioni di output e di input.*

Definizione 6.3.7 *La funzione SVars è una funzione da sequenze di interazioni a variabili. Data una sequenza di interazioni s, SVars*(s) *restituisce l'insieme delle variabili che occorrono nei messaggi della sequenza.*

Definizione 6.3.8 *Riprendendo lo stile della semantica delle riduzioni dato in Tabella 6.2, definiamo la* riduzione simbolica

$$s \,;\, P \rightsquigarrow_R s' \,;\, P'$$

come una relazione tra configurazioni che sono formate da una sequenza di messaggi di input o di output e un processo, e indicizzata da una sostituzione R. Quest'ultima contiene i vincoli, imposti alle variabili, necessari per l'esecuzione del processo. Le regole relative alla riduzione simbolica sono in Tabella 6.3. Scriviamo $s \,;\, P \rightsquigarrow s' \,;\, P'$ *se esiste una riduzione simbolica* $s \,;\, P \rightsquigarrow_R s' \,;\, P'$ *per qualche R, e denotiamo con* $\rightsquigarrow^*$ *la chiusura riflessiva e transitiva di* $\rightsquigarrow$*.*

Questa relazione comprende la conoscenza dell'ambiente e lo stato del sistema, e determina i vincoli attraverso i quali il sistema si evolve.
Sia la regola di input che quella di output memorizzano le interazioni con l'ambiente. Le regole (*SPLIT*), (*SKCASE*), (*PKCASE*) e (*MATCH*) vengono utilizzate per calcolare i vincoli che permettono al sistema di evolvere. La regola (*PAR*) definisce come i processi P e Q interagiscono solo con l'ambiente, indipendentemente uno dall'altro.
Osserviamo ora come le regole di Tabella 6.3 sono utilizzate per costruire le sequenze di interazione che costituiscono il modello astratto. Questo è rappresentato come un grafo diretto

$$(IN) \quad s\ ;\ ?(x).P \ \leadsto_{[\,]}\ s \cdot ?\langle x\rangle\ ;\ P$$
con x che non occorre in s

$$(OUT) \quad s\ ;\ !\langle M\rangle.P \ \leadsto_{[\,]}\ s \cdot !\langle M\rangle\ ;\ P$$

$$(SPLIT) \quad s\ ;\ let\ M = (x,y)\ in\ P \ \leadsto_R\ s[R]\ ;\ P[R]$$
dove $R = mgu\{M = (x,y)\}$
con x e y che non occorrono in s

$$(SKCASE) \quad s\ ;\ case\ M\ of\ \{x\}_N\ in\ P \ \leadsto_R\ s[R]\ ;\ P[R]$$
dove $R = mgu\{M = \{x\}_N\}$
con x che non occorre in s

$$(PKCASE) \quad s\ ;\ case\ M\ of\ \{\!|x|\!\}_{k^-}\ in\ P \ \leadsto_R\ s[R]\ ;\ P[R]$$
dove $R = mgu\{M = \{\!|x|\!\}_{k^+}\}$
con x che non occorre in s

$$(MATCH) \quad s\ ;\ if\ M = N\ then\ P \ \leadsto_R\ s[R]\ ;\ P[R]$$
dove $R = mgu\{M = N\}$

$$(PAR) \quad \frac{s\ ;\ P \ \leadsto_R\ s'\ ;\ P'}{s\ ;\ P|Q \ \leadsto_R\ s'\ ;\ P'|Q[R]} \qquad \frac{s\ ;\ Q \ \leadsto_R\ s'\ ;\ Q'}{s\ ;\ P|Q \ \leadsto_R\ s'\ ;\ P[R]|Q'}$$

Tabella 6.3. Regole per riduzioni simboliche

seguendo lo stile del modello concreto. Etichettiamo ogni nodo del grafo con una configurazione, memorizzando così la storia delle interazioni tra il processo e l'ambiente, e il futuro comportamento del sistema. Utilizziamo le transizioni definite in Tabella 6.3 come archi del grafo. Le transizioni di input e di output sono etichettate rispettivamente con $?\langle x\rangle$ e $!\langle M\rangle$, dove x è una variabile nuova, cioè che non è mai stata utilizzata prima, e M è il messaggio spedito dal processo. Gli altri archi sono etichettati con vincoli dati in forma di sostituzioni e generati dalle regole (*SPLIT*), (*SKCASE*), (*PKCASE*) e (*MATCH*). Scriviamo $[x \mapsto M]$ per indicare che il messaggio M viene sostituito alle occorrenze libere della variabile x nel processo che descrive il futuro comportamento del sistema. Introduciamo ora il concetto di run simbolica che si differenzia dalla run ground, introdotta nella Definizione 6.2.12, in quanto i messaggi che occorrono nella sequenza sono arbitrari.

Definizione 6.3.9 *Una* run *è una sequenza di input/output tale che la prima occorrenza di una variabile è in un messaggio di input.*

Vincolando la prima occorrenza di una variabile ad un messaggio di input, viene rispetata l'assunzione di restringere l'analisi ai soli processi chiusi.

Lemma 6.3.10 *Le sequenze di interazioni, ottenute applicando le regole di Tabella 6.3, sono run.*

Quindi una run può essere vista come un cammino del grafo simbolico delle transizioni. Per la verifica delle proprietà di sicurezza non interessa studiare solo una possibile esecuzione del protocollo, ma dobbiamo considerare tutte le sue possibili esecuzioni. Diamo così la definizione dell'insieme di run di un processo.

Definizione 6.3.11 *Dato un processo chiuso P, definiamo l'insieme Run(P) come l'insieme delle sequenze di interazioni, ottenute attraverso l'applicazione delle regole di Tabella 6.3, cioè*

$$Run(P) = \{s \mid \epsilon\ ;\ P \rightsquigarrow^* s\ ;\ Q\}$$

Notiamo che, in generale, l'insieme delle run simboliche di un processo non è chiuso per prefissi.

Esempio 6.3.12 *Sia* $P = ?(x)\ .\ !\langle M\rangle\ .\ if\ x = N\ then\ \mathbf{0}$ *un processo. Allora*

$$Run(P) = \{\epsilon\ ,\ ?\langle x\rangle\ ,\ ?\langle x\rangle \cdot !\langle M\rangle\ ,\ ?\langle N\rangle \cdot !\langle M\rangle\}$$

Lemma 6.3.13 *L'insieme Run(P) è costituito da run.*

Run(*P*) è l'insieme di tutte le run del processo, cioè l'insieme di tutti i possibili cammini del grafo simbolico delle transizioni. L'insieme delle run simboliche di un processo chiuso è finito, ma non tutte le sequenze che appartengono a *Run*(*P*) hanno una traccia corrispondente nel modello concreto, in quanto la riduzione simbolica delle azioni di input non considera i messaggi effettivamente conosciuti dall'ambiente.
Stabilendo quali sono i vincoli che consentono di passare dal grafo simbolico al grafo delle transizioni, possiamo analizzare strutture simboliche, in quanto finite, e avere la garanzia che i risultati ottenuti valgano anche per il caso reale.

Lemma 6.3.14 *Sia P un processo chiuso.*

1. *Per ogni riduzione* $\epsilon\ ;\ P \rightarrow^* t\ ;\ P'$ *esiste una sostituzione ground* ρ *e una riduzione simbolica* $\epsilon\ ;\ P \rightsquigarrow^* s\ ;\ Q$ *tale che* $s[\rho] = t$ *e* $Q[\rho] = P'$.

2. *Per ogni riduzione simbolica* $\epsilon\ ;\ P \rightsquigarrow^* s\ ;\ Q$ *e sostituzione ground* ρ *tale che* $s[\rho]$ *è una trace ground, esiste una riduzione* $\epsilon\ ;\ P \rightarrow^* t\ ;\ P'$ *tale che* $t = s[\rho]$ *e* $P' = Q[\rho]$.

Corollario 6.3.15 *Per ogni processo chiuso P e run ground t, abbiamo che* $t \in Trace(P)$ *sse esiste una run* $s \in Run(P)$ *e una sostituzione ground* ρ *tale che* $t = s[\rho]$ *è una traccia ground.*

Questo risultato assicura che ogni traccia del grafo delle transizioni ha una run corrispondente nel grafo simbolico. Inoltre, data una run s nel grafo simbolico e una sostituzione ground ρ tale che $s[\rho]$ è una traccia ground, la traccia ground $s[\rho]$ appartiene al grafo delle transizioni. Il nostro obiettivo è così stabilire quali run del grafo simbolico hanno una traccia corrispondente nel grafo delle transizioni. Vogliamo, quindi, raffinare il modello astratto eliminando i *controesempi spuri* [24, 40], cioè quei comportamenti che non hanno una controparte nel modello concreto. Infatti, l'approssimazione, rappresentata dal grafo simbolico, potrebbe non essere completa, cioè, se la proprietà in analisi risulta falsa, il controesempio prodotto potrebbe essere una qualche particolare run nel modello approssimato, che non è presente nel modello concreto. Affronteremo il problema di eliminare i controesempi spuri definendo un algoritmo in grado di calcolare le sostituzioni per cui una run simbolica diventa una traccia ground.

Capitolo 7

Analisi della Conoscenza

Per ottenere un modello finito che rappresenti esattamente le possibili esecuzioni del protocollo e quindi utile per una verifica automatica della correttezza dei protocolli, utilizziamo una procedura simbolica che opera in due passi. Questo capitolo introduce il secondo passo dell'analisi simbolica, che consente di determinare la conoscenza dell'ambiente e quindi di raffinare il grafo simbolico delle transizioni, eliminando i controesempi spuri, cioè quei comportamenti simbolici che non hanno una controparte nel modello concreto.
Il sistema deduttivo di Tabella 6.1 non assicura la terminazione della deduzione in quanto calcola derivazioni che contengono cicli. Allora, a partire da questo sistema, definiamo nuovi sistemi deduttivi che consentono di analizzare algoritmicamente la conoscenza dell'ambiente, cioè di verificare in modo automatico se un messaggio è derivabile dall'ambiente sulla base delle sue conoscenze. Questi sistemi sono dati in modo da rendere efficiente la loro implementazione. A partire dai nuovi sistemi introdotti definiamo una procedura decisionale per il controllo della conoscenza ristretta a messaggi ground e, poi, la generalizziamo fornerndo una procedura simbolica per l'analisi della conoscenza per messaggi arbitrari, cioè per messaggi contenenti variabili.

Esempio 7.0.1 *Sia* $P = !\langle\{n\}_{h(n)}\rangle \,.\, \mathbf{0} \mid ?(x) \,.\, case\ x\ of\ \{y\}_n\ in\ !\langle y\rangle \,.\, \mathbf{0}$ *un processo. L'insieme delle sue tracce ground è dato dalla chiusura dei prefissi delle seguenti due tracce.*

$$\begin{array}{ll} (1) & !\langle\{n\}_{h(n)}\rangle \cdot ?\langle M\rangle \\ (2) & ?\langle N\rangle \cdot !\langle\{n\}_{h(n)}\rangle \end{array}$$

dove $\{n\}_{h(n)} \vdash M$ *e* $\vdash N$*. D'altra parte il suo insieme di run simboliche è dato dalla chiusura dei prefissi delle seguenti run.*

$$\begin{array}{ll} (i) & !\langle\{n\}_{h(n)}\rangle \cdot ?\langle x\rangle \\ (ii) & ?\langle x\rangle \cdot !\langle\{n\}_{h(n)}\rangle \\ (iii) & !\langle\{n\}_{h(n)}\rangle \cdot ?\langle\{y\}_n\rangle \cdot !\langle y\rangle \\ (iv) & ?\langle\{y\}_n\rangle \cdot !\langle\{n\}_{h(n)}\rangle \cdot !\langle y\rangle \\ (v) & ?\langle\{y\}_n\rangle \cdot !\langle y\rangle \cdot !\langle\{n\}_{h(n)}\rangle \end{array}$$

Le run (*i*) *e* (*ii*) *sono la controparte simbolica delle tracce* (1) *e* (2)*, mentre le altre run sono senza significato. Quella in* (*iii*) *perché non ci sono sostituzioni R tali che* $\{n\}_{h(n)} \vdash \{y\}_n[R]$ *sia derivabile, e, similmente quelle in* (*iv*) *e* (*v*) *perché non ci sono sostituzioni R tali che il giudizio* $\vdash \{y\}_n[R]$ *sia deducibile.*

Questo esempio evidenzia la necessità di una procedura per classificare run simboliche e quindi che occorre definire un *message derivation engine* [25] che analizzi la conoscenza dell'ambiente. Tale risultato comporta sia decidere se la relazione $K \vdash M$ è valida, con M messaggio ground e K insieme di messaggi ground, sia, più in generale, calcolare le sostituzioni per cui un messaggio arbitrario può essere dedotto da un insieme di messaggi e quindi determinare i vincoli per cui i comportamenti simbolici possono essere utilizzati in fase di verifica delle proprietà di sicurezza.

7.1 Controllo della Conoscenza

Proponiamo una procedura per decidere se un giudizio $K \vdash M$ è derivabile, per M messaggio ground e K insieme finito di messaggi ground. Una tale procedura è conosciuta quando le chiavi utilizzate dagli algoritmi crittografici simmetrici sono atomiche [4, 25, 17], per esempio solo nomi. Consideriamo ora il caso generale dove messaggi arbitrari possono essere usati come chiavi condivise. Questa generalità è utile per modellare quei protocolli che utilizzano chiavi ottenute attraverso funzioni crittografiche.
Prima di introdurre la procedura decisionale, presentiamo alcune definizioni che consentono di progettare un algoritmo efficiente per tale scopo. Definiamo innanzitutto il concetto di derivazione semplice che garantisce uno spazio delle soluzioni finito e quindi l'esistenza di un algoritmo in grado di determinare se un giudizio è derivabile. Dimostriamo, poi, che ogni giudizio derivabile ha una derivazione semplice e quindi che possiamo limitare l'analisi a questa classe di derivazioni senza perdere soluzioni. A questo punto, il nostro obiettivo è introdurre un sistema deduttivo volto a costruire derivazioni semplici. Definiamo così un sistema di regole di inferenza, basato sui sottotermini dei messaggi conosciuti dall'ambiente, in grado di indirizzare la derivazione all'applicazione degli assiomi. Osservando, però, che l'insieme dei sottotermini è ridondante per i nostri scopi, introduciamo la nozione di sottotermine attivo e, basandoci su questa, definiamo un nuovo sistema deduttivo che risulta essere equivalente a quello definito in Tabella 6.1. In questo modo possiamo lavorare con il nuovo sistema deduttivo ed essere sicuri che i risultati ottenuti valgano anche per quello di Tabella 6.1. Da questo nuovo sistema definiamo una procedura in grado di decidere se un giudizio è derivabile.

7.1.1 Derivazioni Semplici

Il sistema deduttivo in Tabella 6.1 non garantisce l'esistenza di un algoritmo in grado di decidere se un giudizio è derivabile. Infatti, con tale sistema, è possibile costruire derivazioni che possono contenere cicli, portando così alla non terminazione di un eventuale algoritmo che implementa tali regole di inferenza.

Esempio 7.1.1 *Siano M e N messaggi, e K un insieme di messaggi. La seguente deduzione, ottenuta applicando le regole di inferenza di Tabella 6.1, contiene cicli.*

$$(PROJ_1)\ \dfrac{(PAIR)\ \dfrac{\dfrac{\Delta_1}{K \vdash M} \quad \dfrac{\Delta_2}{K \vdash N}}{K \vdash (M,N)}}{K \vdash M}$$

Definiamo, quindi, una classe particolare di derivazioni, che chiamiamo derivazioni semplici, in grado di evitare la situazione presentata nell'esempio precedente. In altre parole, vogliamo eliminare la presenza di cicli che rende infinito il numero di derivazioni calcolabili per un giudizio.

Definizione 7.1.2 *Una derivazione è* semplice *se, in ogni cammino del suo albero di derivazione, un giudizio appare al più una volta.*

In altro modo, utilizzando una definizione induttiva, affermiamo che le derivazioni ottenute, applicando le regole (*TOK*), (*PUB*) e (*AX*), sono semplici, e ogni altro tipo di derivazione è semplice se le sue sottoderivazioni sono semplici e la conclusione non appare in esse.
Il seguito di questa sezione mostra come le derivazioni, ottenute applicando le regole di inferenza di Tabella 6.1, godono delle seguenti proprietà:

completezza : giudizi derivabili hanno almeno una derivazione semplice.

computabilità : l'insieme di derivazioni semplici di un giudizio è finito ed effettivamente numerabile.

Queste proprietà sono fondamentali per definire un algoritmo in grado di calcolare la conoscenza dell'ambiente e quindi che consenta di raffinare il grafo simbolico delle transizioni, eliminando quei comportamenti che non hanno una controparte nel modello concreto.
Vediamo ora, in base alla Definizione 7.1.2, quali vincoli vengono imposti all'applicazioni delle regole attraverso alcuni esempi.

Esempio 7.1.3 *A partire dall'esempio 7.1.1, determiniamo quali sono le regole, applicali alle premesse della regola* ($PROJ_1$)*, che restituiscono una derivazione semplice.*

$$(PROJ_1)\ \dfrac{(AX)\ \dfrac{}{K \vdash (M,N)}}{K \vdash M} \qquad (PROJ_1)\ \dfrac{(PROJ_1)\ \dfrac{\dfrac{\Delta}{K \vdash ((M,N),N')}}{K \vdash (M,N)}}{K \vdash M}$$

$$(PROJ_1)\ \dfrac{(SKDEC)\ \dfrac{\dfrac{\Delta_1}{K \vdash \{(M,N)\}_{N'}} \quad \dfrac{\Delta_2}{K \vdash N'}}{K \vdash (M,N)}}{K \vdash M}$$

$$(PROJ_1)\ \dfrac{(PKDEC)\ \dfrac{\Delta_1}{K \vdash \{\!|(M,N)|\!\}_{k^+}} \quad \dfrac{\Delta_2}{K \vdash k^-}}{K \vdash (M,N)}}{K \vdash M}$$

Le uniche regole applicabili risultano essere (*AX*)*,* (*PROJ*$_i$)*,* (*SKDEC*) *e* (*PKDEC*)*. Infatti* (*SKENC*) *e* (*PKENC*) *non possono essere applicate perché il messaggio nella conclusione non è della forma* $\{M\}_N$ *o* $\{\!|M|\!\}_{k^+}$*. Il medesimo discorso vale per la regola* (*HASH*)*. Dall'Esempio 7.1.1 si osserva che l'applicazione della regola* (*PAIR*) *rende la derivazione non semplice.*

Esempio 7.1.4 *Seguendo quanto fatto nell'Esempio 7.1.3, accertiamo quali sono le regole, applicabili alle premesse della regola* (*SKDEC*)*, in grado di garantire che le derivazioni ottenute siano semplici.*

$$(SKDEC)\ \dfrac{(AX)\ \dfrac{}{K \vdash \{M\}_N} \quad \dfrac{\Delta}{K \vdash N}}{K \vdash M}$$

$$(SKDEC)\ \dfrac{(PROJ_1)\ \dfrac{\dfrac{\Delta_1}{K \vdash (\{M\}_N, N')}}{K \vdash \{M\}_N} \quad \dfrac{\Delta_2}{K \vdash N}}{K \vdash M}$$

$$(SKDEC)\ \dfrac{(SKDEC)\ \dfrac{\dfrac{\Delta_1}{K \vdash \{\{M\}_N\}_{N'}} \quad \dfrac{\Delta_2}{K \vdash N'}}{K \vdash \{M\}_N} \quad \dfrac{\Delta_3}{K \vdash N}}{K \vdash M}$$

$$(SKDEC)\ \dfrac{(PKDEC)\ \dfrac{\dfrac{\Delta_1}{K \vdash \{\!|\{M\}_N|\!\}_{k^+}} \quad \dfrac{\Delta_2}{K \vdash k^-}}{K \vdash \{M\}_N} \quad \dfrac{\Delta_3}{K \vdash N}}{K \vdash M}$$

Le uniche regole applicabli sono le regole (*AX*)*,* (*PROJ*$_i$)*,* (*SKDEC*) *e* (*PKDEC*)*, mentre la regola* (*PAIR*) *non può essere utilizzata perché il messaggio nella conclusione non è della forma* (*M*,*N*)*. Il medesimo discorso vale per le regole* (*PKENC*) *e* (*HASH*)*. L'applicazione della regola* (*SKENC*)*, invece, rende la derivazione non semplice, come si vede di seguito.*

$$(SKDEC)\ \dfrac{(SKENC)\ \dfrac{\dfrac{\Delta_1}{K \vdash M} \quad \dfrac{\Delta_2}{K \vdash N}}{K \vdash \{M\}_N} \quad \dfrac{\Delta_2}{K \vdash N}}{K \vdash M}$$

Esempio 7.1.5 *Sia M un messaggio e K un insieme di messaggi. Per la Definizione 7.1.2, le seguenti derivazioni sono semplici in quanto le due occorrenze del giudizio* $K \vdash M$ *non appartengono allo stesso cammino dell'albero di derivazione.*

$$(PAIR)\ \frac{\dfrac{\Delta}{K \vdash M} \quad \dfrac{\Delta}{K \vdash M}}{K \vdash (M,M)}$$

$$(SKENC)\ \frac{\dfrac{\Delta}{K \vdash M} \quad \dfrac{\Delta}{K \vdash M}}{K \vdash \{M\}_M}$$

Dimostriamo ora le proprietà delle derivazioni, utilizzate per provare alcuni risultati necessari per il raggiungimento degli obiettivi proposti. È nostra intenzione mostrare che ogni giudizio derivabile ha una derivazione semplice e che ogni messaggio asserito in una derivazione semplice è un sottotermine di quello asserito nella conclusione o di qualche messaggio appartenente alla conoscenza dell'ambiente.
La prossima proposizione esprime come un giudizio derivabile abbia sempre una derivazione semplice. Questo risultato consente di semplificare la ricerca nello spazio delle derivazioni di un giudizio, fornendo uno strumento in grado di migliorare l'efficienza dell'algoritmo per il controllo della conoscenza.

Proposizione 7.1.6 *Sia M un messaggio e K un insieme di messaggi. Se $K \vdash M$, allora esiste una derivazione semplice di $K \vdash M$.*

Dim. Sia M un messaggio, K un insieme di messaggi e S un albero di prova di profondità p del giudizio $K \vdash M$. La dimostrazione è data per induzione sulla profondità dell'albero di prova.

Caso base : Se S ha profondità 1, allora l'albero di prova è ottenuto applicando un assioma del sistema deduttivo. Per la Definizione 7.1.2, S è semplice.

Passo induttivo : Supponiamo che S abbia profondità $p > 1$. Allora S è costruito da sottoalberi di profondità minore. Per induzione, questi ultimi possono essere sostituiti da derivazioni semplici. Dimostriamo che esiste una derivazione semplice per il giudizio $K \vdash M$ distinguendo i casi sul numero di occorrenze di $K \vdash M$ in S. Se il giudizio $K \vdash M$ occorre solo nella conclusione della derivazione, S è semplice. Se esiste un'altra occorrenza di $K \vdash M$ in S, sia S' il sottoalbero di S che ha come conclusione l'occorrenza più alta di $K \vdash M$. Allora S' è la derivazione semplice cercata. □

Il risultato, sopra esposto, garantisce l'esistenza di un algoritmo in grado di calcolare una derivazione semplice a partire da una qualsiasi derivazione ottenuta mediante il sistema deduttivo di Tabella 6.1.

Osservazione 7.1.7 *Se un giudizio è derivabile, allora esiste una derivazione semplice per il giudizio stesso. Definiamo un algoritmo tale che, data una derivazione ottenuta applicando le regole di Tabella 6.1, restituisca una derivazione semplice. Sia M un messaggio, K un insieme di messaggi e S una derivazione per il giudizio $K \vdash M$.*

1. *Se S è semplice, abbiamo raggiunto il risultato voluto.*

2. *Se S non è semplice, cioè se esiste un giudizio che appare almeno due volte in un cammino dell'albero, rimpiazziamo il sottoalbero generato dall'occorrenza del giudizio a livello più basso, o l'intero albero nel caso in cui il giudizio che si ripete sia la conclusione, con il sottoalbero generato dall'occorrenza più alta del giudizio.*

Se la derivazione risultante è semplice, ci fermiamo, altrimenti riapplichiamo il punto 2. Tale procedura termina in quanto la dimensione dell'albero decresce ad ogni passo.

Esempio 7.1.8 *Sia M un messaggio e K un insieme di messaggi. Calcoliamo una derivazione semplice del giudizio $K \vdash M$ nel sistema deduttivo di Tabella 6.1 a partire da una qualsiasi derivazione attraverso l'applicazione della procedura presentata nell'Osservazione 7.1.7. Data la derivazione S del giudizio $K \vdash M$*

$$\dfrac{\dfrac{\dfrac{\dfrac{(AX)\ \overline{K \vdash M_3}}{K \vdash M_1}}{K \vdash M_4} \qquad \dfrac{(AX)\ \overline{K \vdash M_2}}{K \vdash M_1}}{K \vdash M_1} \qquad \dfrac{(AX)\ \overline{K \vdash M_2}}{K \vdash M_2}}{K \vdash M}$$

otteniamo la derivazione semplice S' del giudizio $K \vdash M$

$$\dfrac{\dfrac{(AX)\ \overline{K \vdash M_3}}{K \vdash M_1} \qquad (AX)\ \overline{K \vdash M_2}}{K \vdash M}$$

Il lemma seguente mostra come ogni messaggio, asserito in una derivazione semplice, è un sottotermine del messaggio nella conclusione o di qualche messaggio appartenente all'insieme da cui deduciamo il messaggio nella conclusione. Questo risultato garantisce che il numero di derivazioni semplici è finito e quindi l'esistenza di un algoritmo per calcolarle.

Lemma 7.1.9 *Sia M un messaggio, K un insieme di messaggi e S una derivazione di $K \vdash M$. Se S è semplice, ogni messaggio asserito in S è un sottotermine di M o di qualche messaggio in K.*

Dim. Sia M un messaggio, K un insieme di messaggi, S un albero di prova semplice di profondità p del giudizio $K \vdash M$. Diamo una dimostrazione per induzione sulla profondità dell'albero di prova.

Caso base : Se S ha profondità 1, allora l'albero è ottenuto applicando un assioma del sistema deduttivo. È immediato verificare che, per le regole (TOK), (PUB) e (AX), il messaggio M è un sottotermine di se stesso.

Passo Induttivo : Supponiamo che S abbia profondità $p > 1$. Allora S è costituito da sottoalberi di profondità minore. Per induzione, questi ultimi possono essere sostituiti da alberi in cui ogni messaggio asserito è un sottotermine di un messaggio in K o del messaggio nella conclusione della sottoderivazione. Dimostriamo che ogni messaggio

asserito in S è un sottotermine di M o di un messaggio in K, distinguendo i casi sull'ultima regola applicata in S. Se l'ultima regola applicata è una regola di introduzione il risultato vale in quanto i messaggi nelle premesse sono sottotermini del messaggio che occorre nella conclusione. Se l'ultima regola applicata è una regola di eliminazione, distinguiamo i seguenti casi.

regola ($PROJ_i$)**:** Analizziamo solo il caso per la regola ($PROJ_1$), in quanto quello per la regola ($PROJ_2$) è analogo. Consideriamo la derivazione semplice S del giudizio $K \vdash M$

$$(PROJ_1)\ \frac{\dfrac{\Sigma_1}{K \vdash (M,N)}}{K \vdash M}\ (M,N) \in AST(K)$$

Sia S_1 la derivazione semplice di $K \vdash (M,N)$. Per ipotesi induttiva, ogni messaggio asserito in S_1 è un sottotermine di un messaggio in K o di (M,N). Inoltre, dato che S è semplice, il giudizio $K \vdash M$ non occorre in Σ_1. Allora (M,N) è un sottotermine di qualche messaggio in K, e quindi ogni messaggio asserito in S è un sottotermine di un messaggio in K.

regola ($SKDEC$)**:** Consideriamo la derivazione semplice S del giudizio $K \vdash M$

$$(SKDEC)\ \frac{\dfrac{\Sigma_1}{K \vdash \{M\}_N} \quad \dfrac{\Sigma_2}{K \vdash N}}{K \vdash M}\ \{M\}_N \in AST(K)$$

Siano S_1 e S_2 le derivazioni semplici rispettivamente di $K \vdash \{M\}_N$ e $K \vdash N$. Per ipotesi induttiva, ogni messaggio asserito in S_1 e S_2 è un sottotermine di un messaggio dell'insieme K o del messaggio nella conclusione della derivazione. Inoltre, dato che S è semplice, il giudizio $K \vdash M$ non occorre in Σ_1 e in Σ_2. Allora $\{M\}_N$ e N sono sottotermini di qualche messaggio in K. Allora ogni messaggio asserito in S è un sottotermine di un messaggio in K.

regola ($PKDEC$)**:** Consideriamo la derivazione semplice S del giudizio $K \vdash M$

$$(PKDEC)\ \frac{\dfrac{\Sigma_1}{K \vdash \{\!|M|\!\}_{k^+}} \quad \dfrac{\Sigma_2}{K \vdash k^-}}{K \vdash M}\ \{\!|M|\!\}_{k^+} \in AST(K)$$

Siano S_1 e S_2 le derivazioni semplici rispettivamente di $K \vdash \{\!|M|\!\}_{k^+}$ e $K \vdash k^-$. Per ipotesi induttiva, ogni messaggio asserito in S_1 e S_2 è un sottotermine di un messaggio dell'insieme K o del messaggio nella conclusione della derivazione. Inoltre, dato che S è semplice, il giudizio $K \vdash M$ non occorre in Σ_1 e in Σ_2. Allora $\{\!|M|\!\}_{k^+}$ e k^- sono sottotermini di qualche messaggio in K. Allora ogni messaggio asserito in S è un sottotermine di un messaggio in K. □

7.1.2 Sottotermini

Finora abbiamo osservato che le derivazioni semplici godono di proprietà interessanti. Il nostro obiettivo è definire un sistema deduttivo in grado di calcolare derivazioni semplici

per cercare di sfruttare tali proprietà. Introduciamo così il concetto di sottotermine che consente di definire un sistema deduttivo in grado di migliorare l'efficienza dell'algoritmo per determinare la conoscenza dell'ambiente definito in Tabella 6.1.

Definizione 7.1.10 *I* sottotermini *di un messaggio sono definiti nel seguente modo:*

$$\begin{array}{lcl} ST(\tau) & = & \tau \\ ST(n) & = & n \\ ST(k^+) & = & k^+ \\ ST(k^-) & = & k^- \\ ST((M,N)) & = & \{(M,N)\} \cup ST(M) \cup ST(N) \\ ST(\{M\}_N) & = & \{\{M\}_N\} \cup ST(M) \cup ST(N) \\ ST(\{\!|M|\!\}_k) & = & \{\{\!|M|\!\}_k,k\} \cup ST(M) \\ ST(h(M)) & = & \{h(M)\} \cup ST(M) \end{array}$$

Inoltre, per un insieme di messaggi K, $ST(K) = \bigcup_{M \in K} ST(M)$.

Lo scopo di utilizzare i sottotermini è indirizzare la deduzione all'applicazione degli assiomi. In altre parole, quando si applicano le regole di inferenza, i messaggi nelle premesse devono essere sottotermini dei messaggi conosciuti dall'ambiente o del messaggio che occorre nella conclusione della deduzione, per non intraprendere cammini che portano a derivazioni non semplici. Vediamo ora come deve essere modificato il sistema deduttivo in base alle osservazioni fatte. Le regole di introduzione non necessitano di controlli aggiuntivi in quanto i messaggi nelle premesse sono sottotermini del messaggio nella conclusione. Il problema nasce quando si considerano le regole di eliminazione. In questo caso si deve controllare che i messaggi asseriti nelle premesse siano sottotermini di quelli che appartengono alla conoscenza dell'ambiente. Infatti, se il messaggio nelle premesse non è un sottotermine di un messaggio che occorre nella conoscenza dell'ambiente, per essere derivato, deve essere scomposto ottenendo una nuova occorrenza del giudizio nella conclusione della derivazione e quindi la derivazione ottenuta non è semplice. Limitiamo il controllo ai messaggi nelle premesse, che hanno come sottotermine il messaggio nella conclusione, in quanto la situazione appena descritta può avvenire solo per tali messaggi. I messaggi asseriti nelle premesse, utilizzati come chiavi di decodifica, sono sottotermini del messaggio codificato o di qualche messaggio che appartiene alla conoscenza dell'ambiente e quindi non è necessaria una loro verifica.

Definizione 7.1.11 *Sia M un messaggio e K un insieme di messaggi. Definiamo un nuovo tipo di* giudizio *come un'espressione della forma* $K \Vdash_{ST} M$. *Un* albero di deduzione *è un albero in cui i giudizi che etichettano ogni nodo sono derivati dai giudizi che etichettano i nodi connessi ad esso tramite una valida applicazione delle regole di inferenza di Tabella 7.1. Un* albero di prova *è un albero di deduzione in cui le foglie sono ottenute attraverso l'applicazione delle regole* (*TOK*)*,* (*PUB*) *e* (*AX*) *di tale sistema deduttivo.*

La definizione data per sottotermini non è ancora ottimale per l'efficienza dell'algoritmo che decide la derivabilità del messaggio dalle conoscenze dell'ambiente. L'utilizzo dell'insieme

$$(TOK)\ \frac{}{K \Vdash_{ST} M}\ M \in \mathcal{T}$$

$$(PUB)\ \frac{}{K \Vdash_{ST} M}\ M \in \mathcal{K}^{+}$$

$$(AX)\ \frac{}{K \Vdash_{ST} M}\ M \in K \backslash \{\mathcal{T} \cup \mathcal{K}^{+}\}$$

$$(PAIR)\ \frac{K \Vdash_{ST} M \qquad K \Vdash_{ST} N}{K \Vdash_{ST} (M,N)}$$

$$(PROJ_1)\ \frac{K \Vdash_{ST} (M,N)}{K \Vdash_{ST} M}\ M \notin \mathcal{T} \cup \mathcal{K}^{+} \quad (M,N) \in ST(K)$$

$$(PROJ_2)\ \frac{K \Vdash_{ST} (M,N)}{K \Vdash_{ST} N}\ N \notin \mathcal{T} \cup \mathcal{K}^{+} \quad (M,N) \in ST(K)$$

$$(SKENC)\ \frac{K \Vdash_{ST} M \qquad K \Vdash_{ST} N}{K \Vdash_{ST} \{M\}_N}$$

$$(SKDEC)\ \frac{K \Vdash_{ST} \{M\}_N \qquad K \Vdash_{ST} N}{K \Vdash_{ST} M}\ M \notin \mathcal{T} \cup \mathcal{K}^{+} \quad \{M\}_N \in ST(K)$$

$$(PKENC)\ \frac{K \Vdash_{ST} M}{K \Vdash_{ST} \{\!|M|\!\}_k}\ k \in \mathcal{K}^{+}$$

$$(PKDEC)\ \frac{K \Vdash_{ST} \{\!|M|\!\}_{k^+} \qquad K \Vdash_{ST} k^-}{K \Vdash_{ST} M}\ M \notin \mathcal{T} \cup \mathcal{K}^{+} \quad \{\!|M|\!\}_{k^+} \in ST(K)$$

$$(HASH)\ \frac{K \Vdash_{ST} M}{K \Vdash_{ST} h(M)}$$

Tabella 7.1. Sistema deduttivo $\Vdash_{ST}$

dei sottotermini è ristretto rispetto alla precisione di tale insieme, in quanto vengono controllate solo le premesse delle regole di eliminazione. Notiamo come nomi, chiavi pubbliche e chiavi private, token e messaggi codificati con funzioni hash non occorrono nelle premesse di queste regole. L'unico caso in cui troviamo questi messaggi nelle premesse, è nella regola (*SKDEC*) e nella regola (*PKDEC*), ma in questo caso occorrono solo come chiavi per decodificare il messaggio.

Esempio 7.1.12 *Sia M un messaggio e K un insieme di messaggi. Consideriamo la derivazione per il giudizio* $K \Vdash_{ST} M$ *dove l'insieme dei messaggi conosciuti dall'ambiente è* $K =$

$\{(M_1,M_2),(M_3,M_4)\}$ *e il messaggio da verificare è* $M = \{(M_1,M_2)\}_{h(M_3)}$. *L'insieme dei sottotermini di K risulta essere* $ST(K) = \{M_1,M_2,M_3,M_4,(M_1,M_2),(M_3,M_4)\}$.

$$\dfrac{(SKENC)\ \dfrac{(AX)\ \dfrac{}{K \Vdash_{ST} (M_1,M_2)} \qquad (HASH)\ \dfrac{(PROJ_1)\ \dfrac{(AX)\ \dfrac{}{K \Vdash_{ST} (M_3,M_4)}}{K \Vdash_{ST} M_3}}{K \Vdash_{ST} h(M_3)}}{K \Vdash_{ST} \{(M_1,M_2)\}_{h(M_3)}}$$

Durante l'applicazione delle regole di introduzione non viene controllato se i messaggi nelle premesse apparengono a ST(K). Nella precedente derivazione l'unico controllo viene fatto per la premessa della regola $(PROJ_1)$.
Consideriamo ora la derivazione per il giudizio $K \Vdash_{ST} M$ *dove l'insieme dei messaggi conosciuti dall'ambiente è* $K = \{(M_1,M_2),M_4,\{M_3\}_{M_4}\}$ *e il messaggio che vogliamo verificare è* $M = \{(M_1,M_2)\}_{h(M_3)}$. *L'insieme dei sottotermini di K risulta essere* $ST(K) = \{M_1,M_2,M_3,M_4,(M_1,M_2),\{M_3\}_{M_4}\}$.

$$\dfrac{(SKENC)\ \dfrac{(AX)\ \dfrac{}{K \Vdash_{ST} (M_1,M_2)} \qquad (HASH)\ \dfrac{(SKDEC)\ \dfrac{(AX)\ \dfrac{}{K \Vdash_{ST} \{M_3\}_{M_4}} \qquad (AX)\ \dfrac{}{K \Vdash_{ST} M_4}}{K \Vdash_{ST} M_3}}{K \Vdash_{ST} h(M_3)}}{K \Vdash_{ST} \{(M_1,M_2)\}_{h(M_3)}}$$

Appare evidente che, per la regola (*SKDEC*)*, viene richiesto che solo il messaggio cifrato, che occorre nelle premesse, appartenga a* $ST(K)$*, mentre non viene richiesto nessun controllo per la chiave.*

7.1.3 Sottotermini Attivi

Abbiamo visto fin qui che l'insieme dei sottotermini è utilizzato per indirizzare la deduzione all'applicazione degli assiomi. Definiamo ora una nuova nozione per sottotermine necessaria per calcolare l'insieme dei messaggi che possono occorrere nelle premesse di una regola di eliminazione e quindi in grado di migliorare l'efficienza dell'algoritmo per il calcolo della conoscenza dell'ambiente. L'idea è considerare solo i messaggi che possono occorrere nelle premesse di una regola di eliminazione e che abbiano il messaggio nella conclusione come sottotermine.

Definizione 7.1.13 *I* sottotermini attivi *di un messaggio sono definiti nel seguente modo:*

$$\begin{array}{lcl}
AST(\tau) & = & \emptyset \\
AST(n) & = & \emptyset \\
AST(k^+) & = & \emptyset \\
AST(k^-) & = & \emptyset \\
AST((M,N)) & = & \{(M,N)\} \cup AST(M) \cup AST(N) \\
AST(\{M\}_N) & = & \{\{M\}_N\} \cup AST(M) \\
AST(\{\!|M|\!\}_k) & = & \{\{\!|M|\!\}_k\} \cup AST(M) \\
AST(h(M)) & = & \emptyset
\end{array}$$

Inoltre, per un insieme di messaggi K, $AST(K) = \bigcup_{M \in K} AST(M)$.

Sia nella definizione di *ST* che in quella di *AST*, non sono state considerate le variabili in quanto abbiamo ristretto l'analisi solo a messaggi ground. Analizziamo ora i motivi che ci hanno spinto a definire l'insieme dei sottotermini attivi. I sottotermini attivi dei token, dei nomi, delle coppie di chiavi per la crittografia a chiave pubblica e della funzione hash sono l'insieme vuoto in quanto non vengono mai richiesti durante l'esecuzione dell'algoritmo. Infatti il controllo viene effettuato solo per le premesse delle regole di eliminazione per indirizzare la deduzione all'applicazione degli assiomi, cioè per garantire che i messaggi nelle premesse di quelle regole siano sottotermini di messaggi conosciuti dall'ambiente. Nel caso di messaggi codificati a chiave condivisa, invece, viene verificato solo il messaggio cifrato e non la chiave. Questo è dovuto al fatto che, in generale, dopo aver applicato la regola (*SKDEC*), nella derivazione della chiave non si ottiene il messaggio che occorre nella conclusione della regola, mentre è possibile riottenerlo applicando la regola (*SKENC*) al messaggio codificato. Quindi controlliamo solo se il messaggio nelle premesse delle regole di eliminazione appare nei sottotermini attivi dei messaggi conosciuti dall'ambiente. In questo modo siamo certi del fatto che non si costruisce un messaggio che si allontana dalle conoscenze dell'ambiente e che porta sicuramente a derivazioni non semplici.

Esempio 7.1.14 *Sia* $M = ((a,b),\{c\}_{\{d\}_e})$ *un messaggio. Allora*

$$\begin{array}{lcl} ST(M) & = & \{((a,b),\{c\}_{\{d\}_e}),(a,b),\{c\}_{\{d\}_e},\{d\}_e,a,b,c,d,e\} \\ AST(M) & = & \{((a,b),\{c\}_{\{d\}_e}),(a,b),\{c\}_{\{d\}_e}\} \end{array}$$

Confrontando la Definizione 7.1.13 con la Definizione 7.1.10 è immediato verificare che, dato un insieme di messaggi K, $AST(K) \subset ST(K)$. Definiamo ora un nuovo sistema deduttivo che utilizza la nozione di sottotermine attivo.

Definizione 7.1.15 *Sia M un messaggio e K un insieme di messaggi. Chiamiamo* giudizio *un'espressione della forma* $K \Vdash M$. *Un* albero di deduzione *è un albero in cui i giudizi che etichettano ogni nodo sono derivati dai giudizi che etichettano i nodi connessi ad esso tramite una valida applicazione delle regole di inferenza di Tabella 7.2. Un* albero di prova *è un albero di deduzione in cui le foglie sono ottenute attraverso l'applicazione delle regole* (*TOK*)*,* (*PUB*) *e* (*AX*) *di tale sistema deduttivo.*

Osservando i sistemi deduttivi fin qui presentati, risulta evidente che i sistemi dati in Tabella 7.1 e in Tabella 7.2 sono sottosistemi di quello dato in Tabella 6.1 e quindi ogni derivazione ottenuta applicando le regole dei sistemi, che utilizzano i sottotermini e i sottotermini attivi per indirizzare la deduzione, sono deduzioni anche nel sistema di Tabella 6.1. L'esempio seguente mostra che il sistema in Tabella 7.2 non garantisce che le derivazioni, ottenute applicando le sue regole, siano semplici evidenziando la necessità di una nuova condizione.

Esempio 7.1.16 *Consideriamo la derivazione per il giudizio* $K \Vdash M_1$ *dove la conoscenza dell'ambiente è data da* $K = \{M_1,M_2,\{(M_1,M_2)\}_{M_3}\}$. *L'insieme dei sottotermini attivi dell'ambiente risulta essere* $AST(K) = \{\{(M_1,M_2)\}_{M_3},(M_1,M_2)\}$.

$$(TOK)\ \frac{}{K \Vdash M}\ M \in \mathcal{T}$$

$$(PUB)\ \frac{}{K \Vdash M}\ M \in \mathcal{K}^+$$

$$(AX)\ \frac{}{K \Vdash M}\ M \in K \backslash \{\mathcal{T} \cup \mathcal{K}^+\}$$

$$(PAIR)\ \frac{K \Vdash M \qquad K \Vdash N}{K \Vdash (M,N)}$$

$$(PROJ_1)\ \frac{K \Vdash (M,N)}{K \Vdash M}\ M \notin \mathcal{T} \cup \mathcal{K}^+ \quad (M,N) \in AST(K)$$

$$(PROJ_2)\ \frac{K \Vdash (M,N)}{K \Vdash N}\ N \notin \mathcal{T} \cup \mathcal{K}^+ \quad (M,N) \in AST(K)$$

$$(SKENC)\ \frac{K \Vdash M \qquad K \Vdash N}{K \Vdash \{M\}_N}$$

$$(SKDEC)\ \frac{K \Vdash \{M\}_N \qquad K \Vdash N}{K \Vdash M}\ M \notin \mathcal{T} \cup \mathcal{K}^+ \quad \{M\}_N \in AST(K)$$

$$(PKENC)\ \frac{K \Vdash M}{K \Vdash \{\!|M|\!\}_k}\ k \in \mathcal{K}^+$$

$$(PKDEC)\ \frac{K \Vdash \{\!|M|\!\}_{k^+} \qquad K \Vdash k^-}{K \Vdash M}\ M \notin \mathcal{T} \cup \mathcal{K}^+ \quad \{\!|M|\!\}_{k^+} \in AST(K)$$

$$(HASH)\ \frac{K \Vdash M}{K \Vdash h(M)}$$

Tabella 7.2. Sistema deduttivo $\Vdash$

$$(PROJ_1)\ \frac{\dfrac{\Delta_1}{K \Vdash M_1} \qquad \dfrac{\Delta_2}{K \Vdash M_2}}{K \Vdash (M_1,M_2)}}{K \Vdash M_1}$$

Questa derivazione, pur rispettando le regole di Tabella 7.2, non è semplice.

Il nostro scopo è mostrare come il sistema deduttivo, appena introdotto, è equivalente a quello dato in Tabella 6.1. Prima di provare formalmente questo risultato, mostriamo che il nuovo sistema deduttivo gode di alcune proprietà che abbiamo dimostrato per il sistema di Tabella 6.1. Il seguente lemma mostra che i messaggi che occorrono in una derivazione semplice, ottenuta applicando le regole di inferenza di Tabella 7.2, sono sottotermini del messaggio

nella conclusione della derivazione o dei messaggi da cui viene dedotto il messaggio nella conclusione.

Lemma 7.1.17 *Sia M un messaggio, K un insieme di messaggi e S una derivazione del giudizio $K \Vdash M$. Se S è semplice, ogni messaggio asserito in S è un sottotermine di M o di qualche messaggio in K.*

Dim. Sia M un messaggio, K un insieme di messaggi e S l'albero di prova semplice del giudizio $K \Vdash M$ di profondità p. Diamo una dimostrazione per induzione sulla profondità dell'albero di prova.

Caso base : Se S ha profondità 1, allora l'albero è ottenuto applicando un assioma del sistema deduttivo. È immediato verificare che, per le regole (TOK), (PUB) e (AX), il messaggio M è un sottotermine di se stesso.

Passo Induttivo : Supponiamo che S abbia profondità $p > 1$. Allora S è costituito da sottoalberi di profondità minore. Per induzione, questi ultimi possono essere sostituiti da alberi in cui ogni messaggio asserito è un sottotermine di un messaggio in K o del messaggio nella conclusione della sottoderivazione. Dimostriamo che ogni messaggio asserito in S è un sottotermine di M o di un messaggio in K, distinguendo i casi sull'ultima regola applicata in S. Se l'ultima regola applicata è una regola di introduzione il risultato vale in quanto i messaggi nelle premesse sono sottotermini del messaggio che occorre nella conclusione. Se l'ultima regola applicata è una regola di eliminazione distinguiamo i seguenti casi.

regola $(PROJ_i)$**:** Analizziamo solo il caso per la regola $(PROJ_1)$, in quanto quello per la regola $(PROJ_2)$ è analogo. Consideriamo la derivazione semplice S del giudizio $K \Vdash M$

$$(PROJ_1)\ \frac{\dfrac{\Sigma_1}{K \Vdash (M,N)}}{K \Vdash M}\ (M,N) \in AST(K)$$

Sia S_1 la derivazione semplice di $K \Vdash (M,N)$. Per ipotesi induttiva, ogni messaggio asserito in S_1 appartiene a $ST(\{(M,N)\} \cup K)$. Per applicare la regola $(PROJ_1)$, (M,N) deve appartenere a $AST(K)$. Allora ogni messaggio asserito nella derivazione S appartiene a $ST(K)$.

regola $(SKDEC)$**:** Consideriamo la derivazione semplice S del giudizio $K \Vdash M$

$$(SKDEC)\ \frac{\dfrac{\Sigma_1}{K \Vdash \{M\}_N} \qquad \dfrac{\Sigma_2}{K \Vdash N}}{K \Vdash M}\ \{M\}_N \in AST(K)$$

Siano S_1 e S_2 le derivazioni semplici rispettivamente di $K \Vdash \{M\}_N$ e $K \Vdash N$. Per ipotesi induttiva, ogni messaggio asserito in S_1 appartiene a $ST(\{\{M\}_N\} \cup K)$ e ogni messaggio asserito in S_2 appartiene a $ST(\{N\} \cup K)$. Per applicare la regola $(SKDEC)$, $\{M\}_N$ deve appartenere a $AST(K)$. Quindi anche N appartiene ai sottotermini di K. Allora ogni messaggio asserito nella derivazione S appartiene a $ST(K)$.

regola (*PKDEC*)**:** Consideriamo la derivazione semplice S del giudizio $K \Vdash M$

$$(PKDEC)\ \frac{\dfrac{\Sigma_1}{K \Vdash \{\!|M|\!\}_{k^+}} \quad \dfrac{\Sigma_2}{K \Vdash k^-}}{K \Vdash M}\ \{\!|M|\!\}_{k^+} \in AST(K)$$

Siano S_1 e S_2 le derivazioni semplici rispettivamente di $K \Vdash \{\!|M|\!\}_{k^+}$ e $K \Vdash k^-$. Per ipotesi induttiva, ogni messaggio asserito in S_1 appartiene a $ST(\{\{\!|M|\!\}_{k^+}\} \cup K)$ e ogni messaggio asserito in S_1 appartiene a $ST(\{k^-\} \cup K)$. Per applicare la regola (*PKDEC*), il messaggio $\{\!|M|\!\}_{k^+}$ deve appartenere a $AST(K)$. Inoltre, per derivare il giudizio $K \Vdash k^-$, la chiave privata k^- deve essere un sottotermine di qualche messaggio in K. Allora ogni messaggio asserito nella derivazione S appartiene a $ST(K)$. □

I prossimi due teoremi sono fondamentali per dimostrare l'esistenza di un algoritmo che consente di raffinare il grafo delle transizioni eliminando quelle run che non hanno una controparte nel modello concreto. Il primo mostra che il sistema deduttivo appena introdotto è equivalente a quello dato in Tabella 6.1. In questo modo operiamo con il sistema di Tabella 7.2, computazionalmente più efficiente, sapendo che i risultati ottenuti sono validi anche nel sistema di Tabella 6.1. Il secondo teorema è fondamentale per dimostrare che è decidibile verificare se un giudizio è derivabile. Per provare l'equivalenza dei due sistemi deduttivi si utilizza il seguente lemma, il cui enunciato afferma che ogni derivazione semplice nel sistema di Tabella 6.1 è una derivazione nel sistema di Tabella 7.2.

Lemma 7.1.18 *Ogni derivazione semplice nel sistema deduttivo di Tabella 6.1 è effettivamente una derivazione nel sistema di Tabella 7.2.*

Dim. Sia M un messaggio, K un insieme di messaggi, S l'albero di prova semplice di $K \vdash M$ di profondità p. Diamo una dimostrazione per induzione sulla profondità dell'albero di prova.

Caso base : Se S ha profondità 1, allora l'albero è ottenuto applicando un assioma del sistema deduttivo. In questo caso il lemma vale in quanto gli assiomi sono uguali nei due sistemi deduttivi.

Passo Induttivo : Supponiamo che S abbia profondità $p > 1$. Allora S è costituito da sottoalberi di profondità minore. Per induzione, questi ultimi sono alberi ottenuti applicando le regole di inferenza di Tabella 7.2. Dimostriamo che la derivazione S è ottenuta applicando le regole di Tabella 7.2, distinguendo i casi sull'ultima regola applicata in S. Se l'ultima regola applicata è una regola di introduzione il risultato vale in quanto queste regole sono uguali nei due sistemi. Se l'ultima regola applicata è una regola di eliminazione distinguiamo i seguenti casi.

regola ($PROJ_i$) : Analizziamo solo il caso per la regola ($PROJ_1$), in quanto è analogo a quello per la regola ($PROJ_2$). Supponiamo che S sia della forma

$$(PROJ_1)\ \frac{\dfrac{\Delta_1}{K \vdash (M,N)}}{K \vdash M}$$

Sia S' l'albero di prova del giudizio $K \Vdash (M,N)$. Per induzione, S' è ottenuto applicando le regole di Tabella 7.2. Per il Lemma 7.1.9, $(M,N) \in ST(K)$. Dato che non esiste alcuna regola in cui il messaggio nella conclusione occorre come chiave in un messaggio delle premesse e che $K \Vdash (M,N)$ è derivabile, allora (M,N) deve appartenere all'insieme $AST(K)$. Ma questo è quanto richiesto dalla regola $(PROJ_1)$ del sistema in Tabella 7.2. Allora S è una derivazione ottenuta applicando le regole di Tabella 7.2.

regola $(SKDEC)$: Supponiamo che S sia della forma

$$(SKDEC)\ \frac{\dfrac{\Delta_1}{K \vdash \{M\}_N} \qquad \dfrac{\Delta_2}{K \vdash N}}{K \vdash M}$$

Siano S_1 e S_2, rispettivamente, gli alberi di prova dei giudizi $K \Vdash \{M\}_N$ e $K \vdash N$. Per induzione, S_1 e S_2 sono ottenuti applicando le regole di Tabella 7.2. Per il Lemma 7.1.9, $\{M\}_N$ e N appartengono a $ST(K)$. Dato che non esiste alcuna regola in cui il messaggio nella conclusione occorre come chiave in un messaggio delle premesse e che $K \Vdash \{M\}_N$ è derivabile, allora $\{M\}_N$ deve appartenere all'insieme $AST(K)$. Ma questo è quanto richiesto dalla regola $(SKDEC)$ del sistema in Tabella 7.2. Allora S è una derivazione ottenuta applicando le regole di Tabella 7.2.

regola $(PKDEC)$: Supponiamo che S sia della forma

$$(PKDEC)\ \frac{\dfrac{\Delta_1}{K \vdash \{\!|M|\!\}_{k^+}} \qquad \dfrac{\Delta_2}{K \vdash k^-}}{K \vdash M}$$

Siano S_1 e S_2, rispettivamente, gli alberi di prova dei giudizi $K \Vdash \{\!|M|\!\}_{k^+}$ e $K \vdash k^-$. Per induzione, S_1 e S_2 sono ottenuti applicando le regole di Tabella 7.2. Per il Lemma 7.1.9, $\{\!|M|\!\}_{k^+}$ e k^- appartengono a $ST(K)$. Dato che non esiste alcuna regola in cui il messaggio nella conclusione occorre come chiave in un messaggio delle premesse e che $K \Vdash \{\!|M|\!\}_{k^+}$ è derivabile, allora $\{\!|M|\!\}_{k^+}$ deve appartenere all'insieme $AST(K)$. Ma questo è quanto richiesto dalla regola $(PKDEC)$ del sistema in Tabella 7.2. Allora S è una derivazione ottenuta applicando le regole di Tabella 7.2. □

Abbiamo provato che una derivazione semplice ottenuta applicando le regole di Tabella 6.1 è anche una derivazione ottenuta applicando le regole di Tabella 7.2. Utilizziamo questo risultato per dimostrare che i sistemi in Tabella 6.1 e in Tabella 7.2 sono equivalenti.

Teorema 7.1.19 *Sia M un messaggio e K un insieme di messaggi. Il giudizio $K \vdash M$ è derivabile sse esiste una derivazione del giudizio $K \Vdash M$.*

Dim. ($\Leftarrow$) Banale. Infatti il sistema deduttivo in Tabella 7.2 è un sottosistema di quello di Tabella 6.1.

($\Rightarrow$) Sia M un messaggio e K un insieme di messaggi. Per la Proposizione 7.1.6, se il giudizio $K \vdash M$ è derivabile, esiste una derivazione semplice di $K \vdash M$. Per il Lemma 7.1.18, una derivazione semplice, ottenuta applicando le regole di inferenza del sistema deduttivo di Tabella 6.1, è una derivazione ottenuta applicando le regole del sistema di Tabella 7.2. Allora il giudizio $K \Vdash M$ è derivabile. □

Dimostriamo ora che il numero di derivazioni semplici per un giudizio $K \Vdash M$ è finito. Ricordando che l'analisi si concentra solo su processi finiti, è lecito supporre che K sia un insieme finito di messaggi e di conseguenza anche $AST(K)$ è un insieme finito. Inoltre, per la Definizione 7.1.2, un giudizio può comparire solo una volta in un cammino dell'albero di derivazione. Le regole di introduzione hanno nelle premesse un sottotermine del messaggio che occorre nella conclusione, e quindi è immediato concludere che le regole di introduzione possono essere applicate solo un numero finite di volte. L'applicazione delle regole di eliminazione, invece, è vincolata al fatto che nelle premesse sia presente un elemento di $AST(K)$, che è finito. Se le derivazioni sono semplici, non è possibile utilizzare due volte lo stesso elemento di $AST(K)$, e quindi anche le regole di eliminazione possono essere applicate solo un numero finito di volte. Concludendo, lo spazio delle soluzioni è finito e quindi il numero di derivazioni semplici è finito. Diamo ora una dimostrazione formale di questo risultato.

Teorema 7.1.20 *Sia M un messaggio e K un insieme di messaggi. L'insieme delle derivazioni semplici del giudizio $K \Vdash M$ è finito ed esiste una procedura che, dati K e M, le enumera.*

Dim. Dimostriamo che le derivazioni semplici sono finite trovando un upper bound al numero di tali derivazioni. Per il Lemma 7.1.17, in una derivazione semplice i messaggi asseriti sono sottotermini di qualche messaggio in K o del messaggio nella conclusione della derivazione. Sia $S = K \cup \{M\}$ e n il numero di elementi di S. Il numero di alberi, che è possibile costruire dato un insieme di nodi, può essere descritto dalla ricorrenza

$$\begin{cases} T(1) &= 2 \\ T(n) &= n(3 + 4T(n-1) + 4(T(n-1))^2) + 1 \end{cases}$$

Possiamo notare che

$$T(n) \leq 11n(T(n-1))^2 + 1$$

Allora, risolvendo l'ultima ricorrenza, si ottiene che il numero di derivazioni semplici è $O((11n)^{2^{n+1}})$.
Presentiamo in Figura 7.1 e 7.2 la procedura che enumera le derivazioni semplici del giudizio $K \Vdash M$, mentre in A.1.2 forniamo il suo codice in ML. Questa procedura esegue una visita in profondità dell'albero delle derivazioni e restituisce le sottoderivazioni semplici trovate. La procedura è implementata da Enumerate e dalle sue sottoprocedure Analyse e Synthetise. La sottoprocedura Analyse implementa le regole di introduzione, mentre la sottoprocedura Synthetise implementa le regole (*TOK*), (*PUB*), (*AX*) e le regole di eliminazione. La procedura Enumerate richiede tre parametri:

```
Enumerate(K,M,F) =
    if (M ∈ F)
      then [ ]
      else (Analyse(K,M,F) @ Synthetise(K,M,F))

Analyse(K,M,F) =
    let F' = {M} ∪ F
    in  case M of
        (M1,M2)  ⇒ if Enumerate(K,M1,F') = [ ] orelse Enumerate(K,M2,F') = [ ]
                   then [ ]
                   else for P1 ∈ Enumerate(K,M1,F')
                        union for P2 ∈ Enumerate(K,M2,F')
                                union { pair(P1,P2) }
        {M1}M2   ⇒ if Enumerate(K,M1,F') = [ ] orelse Enumerate(K,M2,F') = [ ]
                   then [ ]
                   else for P1 ∈ Enumerate(K,M1,F')
                        union for P2 ∈ Enumerate(K,M2,F')
                                union { skenc(P1,P2) }
        {|M'|}k  ⇒ if k ∉ K+ orelse Enumerate(K,M',F') = [ ]
                   then [ ]
                   else for P ∈ Enumerate(K,M',F')
                        union { pkenc(P,k) }
        h(M')    ⇒ if Enumerate(K,M',F') = [ ]
                   then [ ]
                   else for P ∈ Enumerate(K,M',F')
                        union { hash(P) }
        ——       ⇒ [ ]
```

Figura 7.1. Procedura Enumerate e sottoprocedura Analyse

K : l'insieme dei messaggi conosciuti dall'ambiente.

M : il messaggio che si sta analizzando.

F : l'insieme dei messaggi che sono già stati analizzati.

□

Quindi verificare se il giudizio $K \vdash M$ è derivabile, è decidibile. Questo risultato garantisce l'esistenza di una procedura decisionale per stabilire la derivabilità del giudizio $K \vdash M$.

7.1.4 Procedura Decisionale

Finora abbiamo mostrato che il numero di derivazioni semplici di un giudizio è finito e l'esistenza di un algoritmo che le enumera. Questi risultati garantiscono l'esistenza di un algoritmo in grado di decidere se un messaggio ground è deducibile da un insieme di messaggi ground. A partire dal sistema deduttivo di Tabella 7.2, definiamo un algoritmo per

```
Synthetise(K,M,F) =
    Synthetisebase(K,M,F) @ Synthetiseproj(K,M,F) @
    Synthetiseskdec(K,M,F) @ Synthetisepkdec(K,M,F)

Synthetisebase(K,M,F) =
    case M of
          τ ⇒ [tok(M)]
          k+ ⇒ [pub(M)]
          – ⇒ if M ∈ K
                then [ax(M)]
                else [ ]

Synthetiseproj(K,M,F) =
    let F' = {M} ∪ F
    in  for (N1,N2) ∈ AST(K)\F'
        union if (N1 ≠ M andalso N2 ≠ M)
                then [ ]
                else for P ∈ Synthetise(K,(N1,N2),F)
                     union { proj(P) }

Synthetiseskdec(K,M,F) =
    let F' = {M} ∪ F
    in  for {M}N ∈ AST(K)\F'
        union if Synthetise(K,{M}N,F') = [ ] andalso Enumerate(K,N,F') = [ ]
                then [ ]
                else for P1 ∈ Synthetise(K,{M}N,F')
                       union for P2 ∈ Enumerate(K,N,F')
                             union { skdec(P1,P2) }

Synthetisepkdec(K,M,F) =
    let F' = {M} ∪ F
    in  for ⟦M⟧k+ ∈ AST(K)\F'
        union if Synthetise(K,⟦M⟧k+,F') = [ ] andalso Enumerate(K,k–,F') = [ ]
                then [ ]
                else for P1 ∈ Synthetise(K,⟦M⟧k+,F')
                       union for P2 ∈ Enumerate(K,k–,F')
                             union { pkdec(P1,P2) }
```

Figura 7.2. Sottoprocedura Synthetise usata nella procedura Enumerate

calcolare la conoscenza dell'ambiente. Come già evidenziato, questo sistema non garantisce che le derivazioni trovate siano semplici e quindi, per sfruttare le proprietà di questa classe di derivazioni, si deve controllare ad ogni passo che i giudizi da analizzare non siano già stati considerati precedentemente.
In Figura 7.3 presentiamo la procedura decisionale in pseudocodice, mentre in A.1.1 diamo il codice in ML. Questa procedura si basa sulla ricerca nello spazio delle derivazioni semplici. L'intento è costruire una derivazione applicando le regole di Tabella 7.2 e tenendo

traccia dei giudizi analizzati in modo da garantire che le derivazioni trovate siano semplici. La procedura decisionale è data dal programma Check. Questo richiama le sottoprocedure Analyse e Synthetise. La prima implementa le regole di introduzione, mentre la seconda le regole (TOK), (PUB), (AX) e le regole di eliminazione. La procedura Check richiede in input tre parametri:

K: l'insieme dei messaggi conosciuti dall'ambiente.

M: il messaggio che stiamo analizzando.

F: l'insieme dei messaggi che sono già stati analizzati.

Check(K,M,F) =
$(M \notin F)$ andalso (Analyse(K,M,F) orelse Synthetise(K,M,F))

Analyse(K,M,F) =
let $F' = \{M\} \cup F$
in case M of
$(M_1,M_2) \Rightarrow$ Check(K,M_1,F') andalso Check(K,M_2,F')
$\{M_1\}_{M_2} \Rightarrow$ Check(K,M_1,F') andalso Check(K,M_2,F')
$\{\!|M'|\!\}_k \Rightarrow k \in \mathcal{K}^+$ andalso Check(K,M',F')
$h(M') \Rightarrow$ Check(K,M',F')
—— $\Rightarrow$ false

Synthetise(K,M,F) =
case M of
$\tau \Rightarrow$ true
$k^+ \Rightarrow$ true
$- \Rightarrow$ $(M \in K)$
orelse
let $F' = \{M\} \cup F$
in $\mathrm{Orelse}_{(N_1,N_2)\in AST(K)\setminus F'} \left(\begin{array}{l} (N_1 = M \text{ orelse } N_2 = M) \\ \text{andalso} \\ \text{Synthetise}(K,(N_1,N_2),F') \end{array} \right)$
orelse
$\mathrm{Orelse}_{\{M\}_N\in AST(K)\setminus F'} \left(\begin{array}{l} \text{Synthetise}(K,\{M\}_N,F') \\ \text{andalso} \\ \text{Check}(K,N,F') \end{array} \right)$
orelse
$\mathrm{Orelse}_{\{\!|M|\!\}_{k^+}\in AST(K)\setminus F'} \left(\begin{array}{l} \text{Synthetise}(K,\{\!|M|\!\}_{k^+},F') \\ \text{andalso} \\ \text{Check}(K,k^-,F') \end{array} \right)$

Figura 7.3. Procedura decisionale Check per il controllo della conoscenza

Con i prossimi due teoremi mettiamo in evidenza che la procedura Check termina e che è in grado di determinare se esiste una derivazione per il giudizio $K \Vdash M$. Questi risultati sono fondamentali per definire una verifica automatica del flusso delle informazioni, e quindi per studiare le proprietà di sicurezza dei protocolli.

Teorema 7.1.21 *Il programma* Check *termina su tutti gli input.*

Dim. Sia M un messaggio e F l'insieme dei messaggi analizzati. La condizione ($M \notin F$) garantisce che le derivazioni trovate siano semplici. Per il Teorema 7.1.20, lo spazio delle derivazioni semplici è finito. Il programma Check fa una ricerca nello spazio delle soluzioni e termina non appena trova una derivazione semplice. Se il programma, dopo aver esplorato l'intero spazio delle soluzioni, non trova una derivazione valida, restitusce false. Allora la procedura termina per ogni input. □

Teorema 7.1.22 *Sia M un messaggio e K un insieme di messaggi. Allora il giudizio* $K \Vdash M$ *è derivabile sse* Check$(K,M,\emptyset)$ = true.

Dim. ($\Rightarrow$) Sia M un messaggio, K un insieme di messaggi e F l'insieme dei messaggi analizzati. Se il giudizio $K \Vdash M$ è derivabile, allora, per l'Osservazione 7.1.7, esiste una derivazione semplice S del giudizio $K \Vdash M$. La condizione $M \notin F$ nella procedura Check garantisce che la derivazione trovata sia semplice. Diamo una dimostrazione per induzione sulla struttura delle derivazioni.

Caso base : Se è stato applicato un assioma, il messaggio M è un token, una chiave pubblica o appartiene a K. In questo caso Check(K,M,F) = Synthetise(K,M,F) = true.

Passo Induttivo : Nel passo induttivo si suppone che il teorema valga per le premesse e si dimostra che vale anche per le conclusioni ottenute applicando le regole di Tabella 7.2.

regola (*PAIR*) : Sia S la derivazione semplice del giudizio $K \Vdash (M_1,M_2)$ e la regola (*PAIR*) l'ultima regola applicata. Per induzione, Check(K,M_1,F) = true e Check(K,M_2,F) = true. Se S è semplice, Check$(K,M_1,F \cup \{(M_1,M_2)\})$ = true e Check$(K,M_2,F \cup \{(M_1,M_2)\})$ = true. Allora Check$(K,(M_1,M_2),F)$ = Analyse$(K,(M_1,M_2),F)$ = true.

regola (*SKENC*) : Sia S la derivazione semplice del giudizio $K \Vdash \{M_1\}_{M_2}$ e (*SKENC*) l'ultima regola applicata nalla derivazione. Per induzione, Check(K,M_1,F) = true e Check(K,M_2,F) = true. Se S è semplice, Check$(K,M_1,F \cup \{\{M_1\}_{M_2}\})$ = true e Check$(K,M_2,F \cup \{\{M_1\}_{M_2}\})$ = true. Allora Check$(K,\{M_1\}_{M_2},F)$ = Analyse$(K,\{M_1\}_{M_2},F)$ = true.

regola (*PKENC*) : Sia S la derivazione semplice del giudizio $K \Vdash \{\!|M|\!\}_k$, (*PKENC*) l'ultima regola applicata. Allora $k \in K^+$ come richiesto dalla procedura Check. Per induzione, Check(K,M,F) = true. Se S è semplice, Check$(K,M,F \cup \{\{\!|M|\!\}_k\})$ = true. Allora Check$(K,\{\!|M|\!\}_k\},F)$ = Analyse$(K,\{\!|M|\!\}_k,F)$ = true.

regola (*HASH*) : Sia S la derivazione semplice del giudizio $K \Vdash h(M)$ e la regola (*HASH*) l'ultima regola applicata. Per induzione, Check(K,M,F) = true. Se S è semplice, Check$(K,M,F \cup \{h(M)\})$ = true. Allora Check$(K,h(M),F)$ = Analyse$(K,h(M),F)$ = true.

regola ($PROJ_i$) : Sia S la derivazione semplice del giudizio $K \Vdash M$ e la regola ($PROJ_i$) l'ultima regola applicata. Allora $(N_1,N_2) \in AST(K)$ con $N_1 = M$ o $N_2 = M$ come richiesto da Check. Per induzione, Check(K,(N_1,N_2),F) = true. Se la derivazione S è semplice, Check(K,(N_1,N_2),$F \cup \{M\}$) = true. Questo implica che Synthetise(K,(N_1,N_2),$F \cup \{M\}$) = true altrimenti S non è semplice. Allora Check(K,M,F) = Synthetise(K,M,F) = true.

regola ($SKDEC$) : Sia S la derivazione semplice del giudizio $K \Vdash M$ e la regola ($SKDEC$) l'ultima regola applicata. Allora $\{M\}_N \in AST(K)$ come richiesto da Check. Per induzione, Check(K,$\{M\}_N$,F) = true e Check(K,N,F) = true. Se S è semplice, Check(K,$\{M\}_N$,$F \cup \{M\}$) = true e Check(K,N,$F \cup \{M\}$) = true. Quindi Synthetise(K,$\{M\}_N$,$F \cup \{M\}$) = true, altrimenti S non è semplice. Allora Check(K,M,F) = Synthetise(K,M,F) = true.

regola ($PKDEC$) : Sia S la derivazione semplice del giudizio $K \Vdash M$ e la regola ($PKDEC$) l'ultima regola applicata. Allora $\{\!|M|\!\}_{k^+} \in AST(K)$ come richiesto da Check. Per induzione, Check(K,$\{\!|M|\!\}_{k^+}$,F) = true e Check(K,k^-,F) = true. Se S è semplice, Check(K,$\{\!|M|\!\}_{k^+}$,$F \cup \{M\}$) = true e Check(K,k^-,$F \cup \{M\}$) = true. Quindi Synthetise(K,$\{\!|M|\!\}_{k^+}$,$F \cup \{M\}$) = true, altrimenti S non è semplice. Allora Check(K,M,F) = Synthetise(K,M,F) = true.

($\Leftarrow$) Il programma Check esplora esaustivamente lo spazio delle derivazioni semplici e, poiché tale spazio è finito, termina. Se termina restituendo true significa che nella sua ricerca ha trovato una derivazione semplice e quindi il giudizio $K \Vdash M$ è derivabile. □

Finora abbiamo definito un algoritmo per determinare se un messaggio ground è derivabile da un insieme di messaggi ground. Questo, però, non è sufficiente per analizzare il modello simbolico in quanto in esso sono presenti anche le variabili. Dobbiamo definire, quindi, un algoritmo in grado di calcolare i vincoli per cui un messaggio arbitrario è deducibile dalla conoscenza dell'ambiente.

7.2 Analisi della Conoscenza

Per una verifica automatica delle proprietà di sicurezza è necessario un modello che sia finito e che rappresenti esattamente le possibili esecuzioni del protocollo. Mediante la semantica simbolica abbiamo costruito un modello che si è finito, ma in cui sono presenti comportamenti non computazionamente validi. Il nostro obiettivo è raffinare il grafo simbolico delle transizioni eliminando le sequenze di interazioni che non hanno una controparte nel modello concreto. Per costruire il modello, utilizzato in fase di verifica, è necessario considerare anche le variabili in quanto queste entità sono introdotte dalla regola di input della semantica simbolica. È nostra intenzione, quindi, generalizzare la procedura decisionale di Figura 7.3 per occuparci di messaggi arbitrari.
In questa sezione presentiamo un sistema deduttivo capace di analizzare la conoscenza dell'ambiente, cioè in grado di calcolare le sostituzioni che rendono derivabile un giudizio. Dal nuovo sistema definiamo una procedura per l'analisi della conoscenza che viene utilizzata per

stabilire quali run del grafo simbolico hanno una traccia corrispondente nel modello concreto. In questo modo siamo in grado di raffinare il grafo simbolico eliminando i controesempi spuri.

Esempio 7.2.1 *Consideriamo il processo*

$$\begin{array}{lcl} P & = & !\langle\{n\}_{h(n)}\rangle \,.\, !\langle\{h(n)\}_n\rangle \,.\, ?(x) \,.\, P'[x] \\ P'[x] & = & \mathit{let}\ x = (y,z)\ \mathit{in}\ \mathit{case}\ y\ \mathit{of}\ \{y'\}_n\ \mathit{in}\ P''[y',z] \\ P''[y',z] & = & \mathit{case}\ z\ \mathit{of}\ \{z'\}_{h(n)}\ \mathit{in}\ \mathit{if}\ y' = z'\ \mathit{then}\ !\langle \mathit{equal}\rangle \,.\, \boldsymbol{0} \end{array}$$

L'insieme delle run simboliche è dato dalla chiusura dei prefissi delle seguenti run:

$$\begin{array}{ll} (i) & !\langle\{n\}_{h(n)}\rangle \cdot !\langle\{h(n)\}_n\rangle \cdot ?\langle x\rangle \,, \\ & !\langle\{n\}_{h(n)}\rangle \cdot !\langle\{h(n)\}_n\rangle \cdot ?\langle(y,z)\rangle \,, \\ (ii) & !\langle\{n\}_{h(n)}\rangle \cdot !\langle\{h(n)\}_n\rangle \cdot ?\langle(\{y'\}_n,z)\rangle \,, \\ & !\langle\{n\}_{h(n)}\rangle \cdot !\langle\{h(n)\}_n\rangle \cdot ?\langle(\{y'\}_n,\{z'\}_{h(n)})\rangle \,, \\ (iii) & !\langle\{n\}_{h(n)}\rangle \cdot !\langle\{h(n)\}_n\rangle \cdot ?\langle(\{y'\}_n,\{y'\}_{h(n)})\rangle \cdot !\langle \mathit{equal}\rangle \end{array}$$

Le run simboliche in (*i*) *sono tracce sotto la sostituzione vuota, cioè dove x, y e z sono un qualsiasi messaggio conosciuto dall'ambiente. Comunque questo non è il caso per le altre run. La prima run in* (*ii*) *è una traccia solo sotto il vincolo* $[y' \mapsto h(n)]$*, dove la chiave n non è conosciuta dall'ambiente e z è un qualunque messaggio conosciuto dall'ambiente. La seconda run in* (*ii*) *è una traccia sotto il vincolo* $[y' \mapsto h(n)\ ,\ z' \mapsto n]$*, dove entrambe le chiavi n e h(n) non sono conosciute dall'ambiente. Infine, non esiste nessun vincolo per* y' *nella run di* (*iii*) *tale che questa possa diventare una traccia.*

Per analizzare run simboliche, estendiamo alcune definizioni in modo da considerare le variabili in quanto ora si analizzano messaggi arbitrari, cioè messaggi che possono contenere delle variabili, e non solo messaggi ground come in Sezione 7.1. Estendiamo così il sistema deduttivo di Tabella 6.1 con la seguente regola di inferenza.

$$(VAR)\ \frac{}{K \vdash M}\ M \in \mathcal{V}$$

e il sistema deduttivo di Tabella 7.2 con la seguente regola di inferenza.

$$(VAR)\ \frac{}{K \Vdash M}\ M \in \mathcal{V}$$

Intuitivamente queste regole dicono che una variabile può essere dedotta da qualsiasi insieme di messaggi. Dovremo estendere anche la definizione di sottotemine e di sottotermine attivo aggiungendo rispettivamente

$$ST(x) = x \quad e \quad AST(x) = \emptyset$$

con $x \in \mathcal{V}$. La scelta fatta per la funzione *AST* è dovuta al fatto che le variabili non possono trovarsi nelle premesse delle regole di eliminazione.

7.2.1 Vincoli

Il sistema deduttivo di Tabella 7.2 si limita a stabilire se un messaggio ground è deducibile da un insieme di messaggi ground. Il nostro obiettivo è estendere tale sistema in modo da calcolare i vincoli per cui un messaggio arbitrario è deducibile dalla conoscenza dell'ambiente. In questo modo otteniamo uno strumento che consente di calcolare le esecuzioni del protocollo computazionalmente valide a partire dal grafo simblico delle transizioni. Definiamo ora il sistema deduttivo che consente di calcolare i vincoli che rendono derivabile un giudizio.

Definizione 7.2.2 *Sia M un messaggio, K un insieme di messaggi e R una sostituzione. Chiamiamo* giudizio *un'espressione della forma $R : K \Vdash M$. Un* albero di deduzione *è un albero in cui i giudizi che etichettano ogni nodo sono derivati dai giudizi che etichettano i nodi connessi ad esso tramite una valida applicazione delle regole di inferenza di Tabella 7.3. Un* albero di prova *è un albero di deduzione in cui le foglie sono ottenute attraverso l'applicazione delle regole* (TOK),(PUB),(VAR) *e* (AX) *di tale sistema deduttivo.*

La lettura intuitiva del giudizio $R : K \Vdash M$ risulta essere *"R potrebbe convalidare il giudizio $K \Vdash M$"* oppure *"il giudizio $K \Vdash M$ potrebbe valere applicando la sostituzione R"*. Indichiamo con $R_1@R_2$ la concatenazione delle sostituzioni R_1 e R_2.
Il sistema in Tabella 7.3 risulta così una versione simbolica del sistema deduttivo di Tabella 7.2 che è alla base della procedura decisionale. Diamo ora una lettura operazionale di alcune regole per capire il loro funzionamento.

- Le regole (TOK), (VAR) e (PUB) dicono rispettivamente che da ogni insieme di messaggi si deduce un token, una variabile o una chiave pubblica senza introdurre vincoli.
- La regola (AX) afferma che il modo più generale in cui M corrisponde ai messaggi in K è dato dalle sostituzioni R ottenute interpretando l'inferenza $K \Vdash M$ come un assioma, cioè R è il most general unifier di M e M', dove $M' \in K$.
- Le sostituzioni, calcolate applicando la regola $(PAIR)$, sono date dalle sostituzioni R_1 ottenute derivando il messaggio M dall'insieme K concatenate con le sostituzioni R_2 ottenute derivando i messaggio $N[R_1]$ dall'insieme $K[R_1]$.
- Le sostituzioni, calcolate applicando la regola $(SKDEC)$, sono date dal most general unifier R del messaggio, che occorre nella conclusione, e dei messaggi che occorrono sotto codifica in un sottotermine attivo di K, concatenate con le sostituzioni R_1, ottenute derivando il messaggio codificato $\{M[R]\}_{N[R]}$ dall'insieme $K[R]$, e con le sostituzioni R_2, ottenute derivando la chiave $N[R][R_1]$ dall'insieme $K[R][R_1]$.

Notiamo che abbiamo dato sequenzialità, e dunque asimmetria, alle regole $(PAIR)$, $(SKENC)$, $(SKDEC)$ e $(PKDEC)$. Questa scelta, invece dell'introduzione di regole simmetriche più eleganti, consente di definire un algoritmo efficiente per calcolare le sostituzioni che rendono un messaggio arbitrario deducibile dalla conoscenza dell'ambiente.

Esempio 7.2.3 *Sia $K = \{\{n\}_{h(n)}, \{h(n)\}_n\}$ un insieme di messaggi. Applicando il sistema deduttivo di Tabella 7.3, i giudizi*

$$(TOK)\ \frac{}{[\,] : K \Vdash \tau}$$

$$(VAR)\ \frac{}{[\,] : K \Vdash x}$$

$$(PUB)\ \frac{}{[\,] : K \Vdash k}$$

$$(AX)\ \frac{}{mgu\{M = M'\} : K \Vdash M} \quad M' \in K\ ,\ M \notin \mathcal{T} \cup \mathcal{V} \cup \mathcal{K}^+$$

$$(PAIR)\ \frac{R_1 : K \Vdash M \qquad R_2 : K[R_1] \Vdash N[R_1]}{R_1@R_2 : K \Vdash (M,N)}$$

$$(PROJ_1)\ \frac{R' : K[R] \Vdash (M[R],N[R])}{R@R' : K \Vdash M} \quad \begin{array}{c} M \notin \mathcal{T} \cup \mathcal{V} \cup \mathcal{K}^+\ , \\ (M',N) \in AST(K) \\ R = mgu\{M = M'\} \end{array}$$

$$(PROJ_2)\ \frac{R' : K[R] \Vdash (M[R],N[R])}{R@R' : K \Vdash N} \quad \begin{array}{c} N \notin \mathcal{T} \cup \mathcal{V} \cup \mathcal{K}^+\ , \\ (M,N') \in AST(K) \\ R = mgu\{N = N'\} \end{array}$$

$$(SKENC)\ \frac{R_1 : K \Vdash M \qquad R_2 : K[R_1] \Vdash N[R_1]}{R_1@R_2 : K \Vdash \{M\}_N}$$

$$(SKDEC)\ \frac{R_1 : K[R] \Vdash \{M[R]\}_{N[R]} \qquad R_2 : K[R][R_1] \Vdash N[R][R_1]}{R@R_1@R_2 : K \Vdash M} \quad \begin{array}{c} M \notin \mathcal{T} \cup \mathcal{V} \cup \mathcal{K}^+, \\ \{M'\}_N \in AST(K) \\ R = mgu\{M = M'\} \end{array}$$

$$(PKENC)\ \frac{R : K \Vdash M}{R : K \Vdash [\![M]\!]_k}\ k \in \mathcal{K}^+$$

$$(PKDEC)\ \frac{R_1 : K[R] \Vdash [\![M[R]]\!]_{k^+} \qquad R_2 : K[R][R_1] \Vdash k^-}{R@R_1@R_2 : K \Vdash M} \quad \begin{array}{c} M \notin \mathcal{T} \cup \mathcal{V} \cup \mathcal{K}^+, \\ [\![M']\!]_{k^+} \in AST(K) \\ R = mgu\{M = M'\} \end{array}$$

$$(HASH)\ \frac{R : K \Vdash M}{R : K \Vdash h(M)}$$

Tabella 7.3. Sistema deduttivo per l'analisi della conoscenza

$$\begin{array}{rl} [\,] : & K \Vdash x \\ [\,] : & K \Vdash (y,z) \\ [y' \mapsto h(n)] : & K \Vdash (\{y'\}_n,z) \\ [y' \mapsto h(n), z' \mapsto n] : & K \Vdash (\{y'\}_n,\{z'\}_{h(n)}) \end{array}$$

sono derivabili, mentre non esiste alcuna sostituzione R per cui $R : K \Vdash (\{y'\}_n,\{y'\}_{h(n)})$ *sia derivabile.*

Il sistema deduttivo per l'analisi della conoscenza non calcola tutte le sostituzioni, per cui un giudizio è derivabile, ma si limita a trovare le sostituzioni più generali che rendono il giudizio

derivabile. Se fosse il contrario l'algoritmo che implementa questo sistema deduttivo non terminerebbe in quanto, in alcuni casi, esistono infinite derivazioni.

Esempio 7.2.4 *Sia x una variabile e M un messaggio. Esistono infinite sostituzioni che rendono derivabile il giudizio $x \Vdash M$, ma la sola sostituzione calcolata dal sistema deduttivo di Tabella 7.3 è $R = [x \mapsto M]$.*

L'esempio seguente mostra come la derivabilità di $R : K \Vdash M$ non implica quella del giudizio $K[R] \Vdash M[R]$.

Esempio 7.2.5 *Il giudizio $[x \mapsto n] : \{n\}_k \Vdash (x,\{x\}_k)$ è derivabile, ma il giudizio $\{n\}_k \Vdash (n,\{n\}_k)$ non lo è.*

$$\dfrac{(VAR)\ \dfrac{}{[\,] : \{n\}_k \Vdash x} \qquad (AX)\ \dfrac{}{[x \mapsto n] : \{n\}_k[\,] \Vdash \{x\}_k[\,]}}{(PAIR)\ \dfrac{}{[\,]@[x \mapsto n] : \{n\}_k \Vdash (x,\{x\}_k)}}$$

Si verifica banalmente, invece, che $\{n\}_k \Vdash (n,\{n\}_k)$ non è valido.

Comunque, per sostituzioni vuote, c'è una corrispondenza stretta tra il sistema deduttivo di Tabella 7.2 e quello di Tabella 7.3. Dimostriamo questa affermazione nella proposizione seguente.

Proposizione 7.2.6 *Sia M un messaggio e K un insieme di messaggi. Allora il giudizio $[\,] : K \Vdash M$ è derivabile sse $K \Vdash M$ è derivabile.*

Dim. ($\Rightarrow$) Sia M un messaggio, K un insieme di messaggi e S un albero di prova del giudizio $[\,] : K \Vdash M$ di profondità p. Diamo una dimostrazione per induzione sulla profondità dell'albero di prova.

Caso base : Se l'albero di prova ha profondità 1, è ottenuto applicando un assioma del sistema deduttivo. Se il messaggio M è un token, una variabile o una chiave pubblica, M è deducibile, rispettivamente attraverso le regole (*TOK*), (*VAR*) e (*PUB*) del sistema di Tabella 7.2. Se S è ottenuto applicando la regola (*AX*) del sistema di Tabella 7.3 e il most general unifier, calcolato tra M e uno dei messaggi in K, è la sostituzione vuota, allora $M \in K$. Quindi M può essere dedotto anche nel sistema di Tabella 7.2 applicando la regola (*AX*).

Passo induttivo : Supponiamo che S abbia profondità $p > 1$. Allora S è costituito da sottoalberi di profondità minore. Per induzione, esiste un sottoalbero di prova ottenuto applicando le regole del sistema deduttivo di Tabella 7.2 per i giudizi che occorrono nei sottoalberi di S. Distinguiamo i casi sull'ultima regola applicata in S.

regola (*PAIR*) : Consideriamo la derivazione S del giudizio $[\,] : K \Vdash (M,N)$

$$(PAIR)\ \dfrac{\dfrac{\Sigma_1}{[\,] : K \Vdash M} \qquad \dfrac{\Sigma_2}{[\,] : K[\,] \Vdash N[\,]}}{[\,] : K \Vdash (M,N)}$$

Per induzione, esistono gli alberi di prova dei giudizi $K \Vdash M$ e $K \Vdash N$. Allora esiste un albero di prova di $K \Vdash (M,N)$ in cui l'ultima regola applicata è la regola (*PAIR*).

regola ($PROJ_i$) : Analizziamo solo la regola ($PROJ_1$)in quanto il caso per la regola ($PROJ_2$) è analogo. Consideriamo la derivazione S del giudizio $[\,] : K \Vdash M$

$$(PROJ_1)\ \frac{\dfrac{\Sigma_1}{[\,] : K[\,] \Vdash (M[\,],N[\,])}}{[\,] : K \Vdash M}$$

Se $mgu\{M = M'\} = [\,]$, $(M,N) \in AST(K)$. Per induzione, esiste un albero di prova del giudizio $K \Vdash (M,N)$. Allora esiste un albero di prova di $K \Vdash M$ in cui l'ultima regola applicata è la regola ($PROJ_1$).

regola (*SKENC*) : Consideriamo la derivazione S del giudizio $[\,] : K \Vdash \{M\}_N$

$$(SKENC)\ \frac{\dfrac{\Sigma_1}{[\,] : K \Vdash M} \qquad \dfrac{\Sigma_1}{[\,] : K[\,] \Vdash N[\,]}}{[\,] : K \Vdash \{M\}_N}$$

Per induzione, esistono gli alberi di prova dei giudizi $K \Vdash M$ e $K \Vdash N$. Allora esiste un albero di prova di $K \Vdash \{M\}_N$ in cui l'ultima regola applicata è la regola (*SKENC*).

regola (*SKDEC*) : Consideriamo la derivazione S del giudizio $[\,] : K \Vdash M$

$$(SKDEC)\ \frac{\dfrac{\Sigma_1}{[\,] : K[\,] \Vdash \{M[\,]\}_{N[\,]}} \qquad \dfrac{\Sigma_2}{[\,] : K[\,] \Vdash N[\,]}}{[\,] : K \Vdash M}$$

Se $mgu\{M = M'\} = [\,]$, $\{M\}_N \in AST(K)$. Per induzione, esistono gli alberi di prova dei giudizi $K \Vdash \{M\}_N$ e $K \Vdash N$. Allora esiste un albero di prova di $K \Vdash M$ in cui l'ultima regola applicata è la regola (*SKDEC*).

regola (*PKENC*) : Consideriamo la derivazione S del giudizio $[\,] : K \Vdash \{\!|M|\!\}_k$ con k chiave pubblica

$$(PKENC)\ \frac{\dfrac{\Sigma_1}{[\,] : K \Vdash M}}{[\,] : K \Vdash \{\!|M|\!\}_k}$$

Per induzione, esiste un albero di prova del giudizio $K \Vdash M$. Allora esiste un albero di prova di $K \Vdash \{\!|M|\!\}_k$ in cui l'ultima regola applicata è la regola (*PKENC*).

regola (*PKDEC*) : Consideriamo la derivazione S del giudizio $[\,] : K \Vdash M$

$$(PKDEC)\ \frac{\dfrac{\Sigma_1}{[\,] : K[\,] \Vdash \{\!|M[\,]|\!\}_{k^+}} \qquad \dfrac{\Sigma_2}{[\,] : K[\,] \Vdash k^-}}{[\,] : K \Vdash M}$$

Se $mgu\{M = M'\} = [\,]$, $\{\!|M|\!\}_{k^+} \in AST(K)$. Per induzione, esistono gli alberi di prova dei giudizi $K \Vdash \{\!|M|\!\}_{k^+}$ e $K \Vdash k^-$. Allora esiste un albero di prova di $K \Vdash M$ in cui l'ultima regola applicata è la regola (*PKDEC*).

regola (*HASH*) : Consideriamo la derivazione S del giudizio $[\,] : K \Vdash h(M)$

$$(HASH)\ \frac{\dfrac{\Sigma_1}{[\,] : K \Vdash M}}{[\,] : K \Vdash h(M)}$$

Per induzione, esiste un albero di prova del giudizio $K \Vdash M$. Allora esiste un albero di prova di $K \Vdash M$ in cui l'ultima regola applicata è la regola (*HASH*).

($\Leftarrow$) Sia M un messaggio, K un insieme di messaggi e S l'albero di prova del giudizio $K \Vdash M$ di profondità p. Diamo una dimostrazione per induzione sulla profondità dell'albero di prova.

Caso base : Se l'albero di prova ha profondità 1, è ottenuto applicando un assioma del sistema deduttivo. Se il messaggio M è un token, una variabile o una chiave pubblica, allora M è deducibile, rispettivamente attraverso le regole (*TOK*), (*VAR*) e (*PUB*) del sistema di Tabella 7.3. Se S è ottenuto applicando la regola (*AX*) del sistema di Tabella 7.2, allora $M \in K$. Quindi, M può essere dedotto anche nel sistema di Tabella 7.3 attraverso la regola (*AX*) e il most general unifier, calcolato tra M e uno dei messaggi in K, è la sostituzione vuota.

Passo induttivo : Supponiamo che S abbia profondità $p > 1$. Allora S è costituito da sottoalberi di profondità minore. Per induzione, esiste un albero di prova, ottenuto applicando le regole del sistema deduttivo di Tabella 7.3, per i giudizi che occorrono nei sottoalberi di S e la sostituzione che lo convalida è la sostituzione vuota. Distinguiamo i casi sull'ultima regola applicata in S.

regola (*PAIR*) : Consideriamo la derivazione S del giudizio $K \Vdash (M,N)$

$$(PAIR)\ \frac{\dfrac{\Delta_1}{K \Vdash M} \quad \dfrac{\Delta_2}{K \Vdash N}}{K \Vdash (M,N)}$$

Per induzione, esistono gli alberi di prova per i giudizi $[\,] : K \Vdash M$ e $[\,] : K \Vdash N$. Allora esiste la derivazione

$$(PAIR)\ \frac{\dfrac{\Sigma_1}{[\,] : K \Vdash M} \quad \dfrac{\Sigma_2}{[\,] : K[\,] \Vdash N[\,]}}{[\,] : K \Vdash (M,N)}$$

regola ($PROJ_i$) : Analizziamo solo la regola ($PROJ_1$), in quanto il caso per la regola ($PROJ_2$) è analogo. Consideriamo la derivazione S del giudizio $K \Vdash M$

$$(PROJ_1)\ \frac{\dfrac{\Delta_1}{K \Vdash (M,N)}}{K \Vdash M}$$

In questo caso $(M,N) \in AST(K)$ e quindi $mgu\{M = M'\} = [\,]$ per $(M',N) \in AST(K)$. Per induzione, esiste un albero di prova del giudizio $[\,] : K \Vdash (M,N)$. Allora esiste la derivazione

$$(PROJ_1)\ \frac{\dfrac{\Sigma_1}{[\,] : K[\,] \Vdash (M[\,],N[\,])}}{[\,] : K \Vdash M}$$

regola $(SKENC)$: Consideriamo la derivazione S del giudizio $K \Vdash \{M\}_N$

$$(SKENC)\ \frac{\dfrac{\Delta_1}{K \Vdash M} \qquad \dfrac{\Delta_2}{K \Vdash N}}{K \Vdash \{M\}_N}$$

Per induzione, esistono gli alberi di prova dei giudizi $[\,] : K \Vdash M$ e $[\,] : K \Vdash N$. Allora esiste la derivazione

$$(SKENC)\ \frac{\dfrac{\Sigma_1}{[\,] : K \Vdash M} \qquad \dfrac{\Sigma_1}{[\,] : K[\,] \Vdash N[\,]}}{[\,] : K \Vdash \{M\}_N}$$

regola $(SKDEC)$: Consideriamo la derivazione S del giudizio $K \Vdash M$

$$(SKDEC)\ \frac{\dfrac{\Delta_1}{K \Vdash \{M\}_N} \qquad \dfrac{\Delta_2}{K \Vdash N}}{K \Vdash M}$$

In questo caso $\{M\}_N \in AST(K)$ e quindi $mgu\{M = M'\} = [\,]$ per $\{M'\} \in AST(K)$. Per induzione, esistono gli alberi di prova dei giudizi $[\,] : K \Vdash \{M\}_N$ e $[\,] : K \Vdash N$. Allora esiste la derivazione

$$(SKDEC)\ \frac{\dfrac{\Sigma_1}{[\,] : K[\,] \Vdash \{M[\,]\}_{N[\,]}} \qquad \dfrac{\Sigma_2}{[\,] : K[\,] \Vdash N[\,]}}{[\,] : K \Vdash M}$$

regola $(PKENC)$: Consideriamo la derivazione S del giudizio $K \Vdash \{\!|M|\!\}_k$ con k chiave pubblica

$$(PKENC)\ \frac{\dfrac{\Delta_1}{K \Vdash M}}{K \Vdash \{\!|M|\!\}_k}$$

Per induzione, esiste un albero di prova del giudizio $[\,] : K \Vdash M$. Allora esiste una albero di prova del giudizio $[\,] : K \Vdash \{\!|M|\!\}_k$ in cui l'ultima regola applicata è la regola $(PKENC)$.

regola $(PKDEC)$: Consideriamo la derivazione S del giudizio $K \Vdash M$

$$(PKDEC)\ \frac{\dfrac{\Delta_1}{K \Vdash \{\!|M|\!\}_{k^+}} \qquad \dfrac{\Delta_2}{K \Vdash k^-}}{K \Vdash M}$$

In questo caso $\{\!|M|\!\}_{k^+} \in AST(K)$ e quindi $mgu\{M = M'\} = [\,]$ per $\{\!|M'|\!\}_{k^+} \in AST(K)$. Per induzione, esistono gli alberi di prova dei giudizi $[\,] : K \Vdash \{\!|M|\!\}_{k^+}$ e $[\,] : K \Vdash k^-$. Allora esiste la derivazione

$$(PKDEC)\ \frac{\dfrac{\Sigma_1}{[\,]:K[\,]\Vdash \{\!|M[\,]|\!\}_{k^+}} \qquad \dfrac{\Sigma_2}{[\,]:K[\,]\Vdash k^-}}{[\,]:K\Vdash M}$$

regola (*HASH*) : Consideriamo la derivazione S del giudizio $K \Vdash h(M)$

$$(HASH)\ \frac{\dfrac{\Delta_1}{K\Vdash M}}{K\Vdash h(M)}$$

Per induzione, esiste un albero di prova del giudizio $[\,]:K\Vdash M$. Allora esiste un albero di prova del giudizio $[\,]:K\Vdash h(M)$ in cui l'ultima regola applicata è la regola (*HASH*). □

Notiamo che l'istanza $\frac{\Sigma[R]}{K[R]\Vdash M[R]}$ della derivazione semplice $\frac{\Sigma}{R:K\Vdash M}$ potrebbe essere una derivazione, ma non necessariamente semplice come mostra il seguente esempio.

Esempio 7.2.7 *Sia $K=\{a,k,\{a\}_{k'},\{\{x\}_k\}_{\{a\}_k}\}$ un insieme di messaggi. La seguente derivazione per il giudizio $R:K\Vdash(\{x\}_k,\{x\}_{k'})$ è semplice.*

$$\dfrac{\dfrac{(AX)\ \dfrac{}{[\,]:K\Vdash\{\{x\}_k\}_{\{a\}_k}}}{(SKDEC)\ \dfrac{}{[\,]:K\Vdash\{x\}_k}} \qquad \dfrac{\dfrac{(AX)\ \dfrac{}{[\,]:K\Vdash a} \qquad (AX)\ \dfrac{}{[\,]:K\Vdash k}}{(SKENC)\ \dfrac{}{[\,]:K\Vdash\{a\}_k}}}{(PAIR)\ \dfrac{}{[\,]:K\Vdash(\{x\}_k,\{x\}_{k'})}} \qquad \dfrac{(AX)\ \dfrac{}{[x\mapsto a]:K\Vdash\{x\}_{k'}}}{[x\mapsto a]:K\Vdash(\{x\}_k,\{x\}_{k'})}$$

La stessa cosa non si può dire per il giudizio $K[R]\Vdash(\{x\}_k,\{x\}_{k'})[R]$.

7.2.2 Procedura Simbolica

Il nostro obiettivo è ora definire una procedura per calcolare i vincoli per i quali un messaggio arbitrario può essere dedotto da un insieme finito di messaggi arbitrari. Presentiamo in Figura 7.4 e in Figura 7.5 l'algoritmo per l'analisi della conoscenza. Questo si basa sul sistema deduttivo di Tabella 7.3 e può essere considerato una versione simbolica della procedura decisionale di Figura 7.3. In A.2.1 diamo il codice ML di questa procedura.
La procedura Realise è costituita dalle sottoprocedure Analyse e Synthetise e fa una ricerca nello spazio delle derivazioni semplici ottenute applicando le regole del sistema deduttivo per l'analisi della conoscenza. La procedura Analyse codifica le regole (*PAIR*), (*SKENC*), (*PKENC*) e (*HASH*), mentre Synthetise codifica le altre. La procedura Realise richiede in input tre parametri:

K: l'insieme dei messaggi conosciuti dall'ambiente.

M: il messaggio che stiamo analizzando.

F: l'insieme dei messaggi che sono già stati analizzati.

Definiamo in Figura 7.6 la procedura Constraints che calcola i vincoli per cui un giudizio è effettivamente deducibile, iterando la procedura Realise. In A.2.2 forniamo il codice ML. La procedura Constraints richiede in input due parametri:

```
Realise(K,M,F) =
    if (M ∈ F)
    then ∅
    else Analyse(K,M,F) ∪ Synthetise(K,M,F)

Analyse(K,M,F) =
    let F' = {M} ∪ F
    in  case M of
            (M1,M2)  ⇒ for R1 ∈ Realise(K,M1,F')
                           union for R2 ∈ Realise(K[R1],M2[R1],F'[R1])
                                 union { R1@R2 }
            {M1}M2   ⇒ for R1 ∈ Realise(K,M1,F')
                           union for R2 ∈ Realise(K[R1],M2[R1],F'[R1])
                                 union { R1@R2 }
            ⦃M'⦄k    ⇒ Realise(K,M',F')
            h(M')    ⇒ Realise(K,M',F')
            ——       ⇒ ∅
```

Figura 7.4. Procedura simbolica Realise per l'analisi della conoscenza

K: l'insieme dei messaggi conosciuti dall'ambiente.

M: il messaggio che stiamo analizzando.

Esempio 7.2.8 *Sia $M = (x,\{x\}_k)$ un messaggio e $K = \{\{n\}_k\}$ un insieme di messaggi. Notiamo come* Constraints$(K,M) = \emptyset$*, mentre* Realise$(K,M,\emptyset) = \{[x \mapsto n]\}$.

Con i prossimi due teoremi dimostriamo che le procedure Realise e Constraints terminano e sono in grado di calcolare le sostituzioni R tali per cui i giudizi $R : K \Vdash M$ e $K[R] \Vdash M[R]$ siano derivabili. Questi risultati sono fondamentali per definire un modello finito utilizzabile per una verifica automatica delle proprietà di sicurezza dei protocolli.

Teorema 7.2.9 *Sia M un messaggio e K un insieme di messaggi. Il programma* Realise *termina su tutti gli input e il giudizio $R : K \Vdash M$ è derivabile per ogni sostituzione $R \in$* Realise$(K,M,\emptyset)$.

Dim. Sia M un messaggio, K un insieme di messaggi e F l'insieme dei messaggi analizzati. Per il Lemma 7.1.17, in una derivazione semplice i messaggi asseriti sono sottotermini di M o di qualche messaggio in K. Dato che K è un insieme finito, l'insieme dei sottotermini di $\{M\} \cup K$ è finito. Realise richiama le due sottoprocedure Analyse e Synthetise. La prima aggiunge il messaggio M in F e analizza un sottotermine di M. Synthetise, se simula le regole (TOK), (VAR) e (PUB), restituisce la sostituzione vuota e termina, altrimenti aggiunge il messaggio M in F ed esegue le regole di eliminazione applicabili. Allora, dopo al più un numero finito di passi, F è uguale all'insieme dei sottotermini di $\{M\} \cup K$, e quindi Realise

```
Synthetise(K,M,F) =
    case M of
        τ ⇒ {[ ]}
        x ⇒ {[ ]}
        k+ ⇒ {[ ]}
        − ⇒     { mgu{M = M'} | M' ∈ K }
                ∪
                   let F' = {M} ∪ F
                   in  for (N1,N2) ∈ AST(K)\F'
                         union let R = mgu{M = N1}
                                   in for R' ∈ Synthetise(K[R],(M[R],N2[R]),F'[R])
                                       union { R@R' }
                                ∪
                                  let R = mgu{M = N2}
                                  in for R' ∈ Synthetise(K[R],(N1[R],M[R]),F'[R])
                                       union { R@R' }
                       ∪
                         for {M'}N ∈ AST(K)\F'
                         union let R = mgu{M = M'}
                                   in for R1 ∈ Synthetise(K[R],{M[R]}N[R],F'[R])
                                       union for R2 ∈ Realise(K[R][R1],N[R][R1],F'[R][R1])
                                                union { R@R1@R2 }
                       ∪
                         for {|M'|}k+ ∈ AST(K)\F'
                         union let R = mgu{M = M'}
                                   in for R1 ∈ Synthetise(K[R],{|M[R]|}k+,F'[R])
                                       union for R2 ∈ Realise(K[R][R1],k−,F'[R][R1])
                                                union { R@R1@R2 }
```

Figura 7.5. Sottoprocedura Synthetise per l'analisi della conoscenza

```
Constraints(K,M) =
    for R ∈ Realise(K,M,∅)
    union if R = [ ]
            then {[ ]}
            else for R' ∈ Constraints(K[R],M[R])
                   union { R@R' }
```

Figura 7.6. Procedura simbolica Constraints per l'analisi della conoscenza

termina.
Osservando le regole di inferenza di Tabella 7.3, è immediato verificare che il giudizio $R : K \Vdash M$ è derivabile per ogni sostituzione $R \in \mathsf{Realise}(K,M,F)$. □

Teorema 7.2.10 *Sia M un messaggio e K un insieme di messaggi. Il programma* Constraints *termina su tutti gli input e, per ogni sostituzione $R \in$ Constraints(K,M), il giudizio $K[R] \Vdash M[R]$ è derivabile.*

Dim. Sia M un messaggio e K un insieme di messaggi. Dimostriamo, per induzione sul numero di variabili in $\{M\} \cup K$, che Constraints(K,M) termina su tutti gli input. La procedura richiama Realise(K,M,$\emptyset$), che, per il Teorema 7.2.9, termina.

Caso base : Se $|VarsI(K \cup \{M\})| = 0$, Constraints termina ristituendo la sostituzione vuota.

Passo induttivo : Se $|VarsI(K \cup \{M\})| > 0$, distinguiamo due casi

1. Realise(K,M,$\emptyset$) = [ ]: Constraints termina ristituendo la sostituzione vuota.
2. Realise(K,M,$\emptyset$) $\neq$ [ ]: Realise calcola le sostituzioni R tali per cui il giudizio $R : K \Vdash M$ è derivabile. In questo caso Constraints(K,M) richiama Constraints($K[R]$,$M[R]$). Per induzione, dato che il numero di variabili in $\{M[R]\} \cup K[R]$ è minore del numero di variabili in $\{M\} \cup K$, Constraints(K,M) termina.

Dimostriamo che, per ogni sostituzione $R \in$ Constraints(K,M), $K[R] \Vdash M[R]$ è derivabile. Constraints(K,M) richiama Realise(K,M,$\emptyset$). La procedura Realise calcola le sostituzioni $R^{(1)}$ tali per cui esiste una derivazione del giudizio $R^{(1)} : K \Vdash M$. La procedura Constraints reitera il procedimento fino a quando non trova la sostituzione $R^{(i+1)} = [\ ]$. Vediamo in dettaglio le iterazioni dell'algoritmo.

$$
\begin{array}{rcl}
R^{(1)} & : & K \Vdash M \\
R^{(2)} & : & K[R^{(1)}] \Vdash M[R^{(1)}] \\
R^{(3)} & : & K[R^{(1)}][R^{(2)}] \Vdash M[R^{(1)}][R^{(2)}] \\
\dots & & \dots \\
R^{(n)} & : & K[R^{(1)}][R^{(2)}]...[R^{(n-1)}] \Vdash M[R^{(1)}][R^{(2)}]...[R^{(n-1)}] \\
R^{(n+1)} & : & K[R^{(1)}][R^{(2)}]...[R^{(n-1)}][R^{(n)}] \Vdash M[R^{(1)}][R^{(2)}]...[R^{(n-1)}][R^{(n)}]
\end{array}
$$

dove $R^{(n+1)} = [\ ]$. Abbiamo visto sopra che questo accade dopo al più un numero finito di passi. Sia $R = R^{(1)} @ R^{(2)} @ \cdots @ R^{(n)}$. Per la Proposizione 7.2.6, se $[\ ] : K[R] \Vdash M[R]$, allora $K[R] \Vdash M[R]$. □

Abbiamo così una procedura che calcola i vincoli per cui possiamo dedurre giudizi simbolici. Questa procedura è fondamentale per determinare quali run sono computazionalmente valide, in quanto è in grado di calcolare i vincoli per cui ogni messaggio ricevuto dal processo è derivabile dalla conoscenza dell'ambiente. Ora quindi possiamo facilmente calcolare modelli simbolici di run, e di conseguenza di processi.

Capitolo 8

Modelli simbolici

Una verifica automatica delle proprietà di sicurezza di un protocollo richiede un modello finito che rappresenti esattamente le possibili esecuzioni del protocollo stesso. Nello sviluppo della nostra tesi abbiamo osservato che il modello concreto è infinito a causa degli infiniti messaggi conosciuti dall'ambiente; per cui, attraverso la riduzione simbolica, abbiamo costruito il grafo simbolico delle transizioni. Questo è si finito, ma non tutti i suoi cammini sono computazionalmente validi a causa della valutazione dei messaggi ricevuti dal processo. Quindi l'analisi delle proprietà di sicurezza su tale grafo potrebbe evidenziare la non correttezza del protocollo, fornendo come controesempio una sequenza di interazioni che non ha una controparte nel modello concreto.
Questo capitolo combina la semantica simbolica e la procedura simbolica per l'analisi della conoscenza per raffinare il grafo simbolico delle transizioni, eliminando i controesempi spuri. Il nostro obiettivo, infatti, è ottenere una rappresentazione finita di tutte le possibili esecuzioni del protocollo, necessaria per eseguire una verifica automatica delle proprietà di sicurezza. In particolare, nella fase di raffinamento tentiamo di unificare i messaggi che occorrono nelle azioni di input con i messaggi appartenenti alla conoscenza dell'ambiente. Iterando questa procedura, è possiblile verificare l'esistenza di una sostituzione per cui una run simbolica ha una controparte nel modello concreto.
Prima di introdurre il modello, utilizzato in fase di verifica, definiamo una nozione che permette di rappresentare una storia valida del sistema, che chiamiamo traccia. L'idea è calcolare le sostituzioni per cui ogni messaggio ricevuto da un processo è deducibile dalla conoscenza corrente dell'ambiente, cioè i vincoli per cui una run del grafo simbolico possa diventare una traccia. Calcoliamo tali sostituzioni attraverso la procedura Model, che dimostriamo terminare per ogni run. In questo modo otteniamo gli strumenti per definire modelli simbolici finiti che risultano corretti e completi, dove per correttezza intendiamo che ogni traccia, che gli appartiene, è un'astrazione di qualche sequenza di interazione del modello concreto, e per completezza intendiamo che ogni traccia del modello concreto è una concretizzazione di qualche sequenza di interazione del modello simbolico. La correttezza e la completezza del modello simbolico sono indispensabili per verificare le proprietà di sicurezza ed essere sicuri che i risultati ottenuti valgano anche nel modello concreto. Per dimostrare

la correttezza e la completezza utilizziamo il concetto di rispettabilità che si rivela uno strumento idoneo per mettere in relazione gli elementi del modello simbolico con quelli del modello concreto e quindi per verificare la loro corrispondenza.
In letteratura esistono alcuni lavori [4, 17] che definiscono un modello completo solo nel caso in cui le chiavi, utilizzate dagli algoritmi crittografici simmetrici, sono atomiche. Questo capitolo definisce, prima, un modello per questo caso particolare e, poi, lo estende per considerare il caso in cui le chiavi condivise sono messaggi arbitrari. Questa generalità risulta utile per modellare protocolli in cui certe chiavi non sono solo entità primitive, ad esempio nomi, ma sono ottenute applicando funzioni crittografiche a qualche nuovo segreto.
Teniamo presente che abbiamo già esteso il sistema deduttivo di Tabella 6.1 a messaggi arbitrari attraverso la regola di inferenza

$$(VAR)\ \frac{}{K \vdash M}\ M \in \mathcal{V}$$

e che questa regola stabilisce che una variabile è derivabile da qualsiasi insieme di messaggi.

8.1 Traccia

Introduciamo il concetto di traccia simbolica corrispondente alla nozione di traccia ground data in Definizione 6.2.13. Questa nozione serve per rappresentare una storia valida del sistema, cioè una possibile esecuzione del protocollo, ed è equivalente al concetto di *traccia in forma risolta* introdotto da Boreale [17, 18]. La nuova definizione di traccia si distingue dalla precedente in quanto ora i messaggi, che occorrono nella sequenza di interazioni, possono contenere variabili. Comunque, la prima occorrenza di una variabile in un messaggio della traccia deve essere in un messaggio di input, in quanto si analizzano processi chiusi, cioè processi in cui non occorrono variabili libere.

Definizione 8.1.1 *Una* run *è una sequenza di messaggi di input/output tali che la prima occorrenza di una variabile è in un messaggio di input.*

Non tutte le run di un processo rappresentano una sequenza di interazioni computazionalmente valida a causa della valutazione degli input, in quanto si considerano messaggi non conosciuti dall'ambiente. Introduciamo così un nuovo concetto per definire una storia valida del sistema. L'idea è verificare se ogni messaggio ricevuto dal processo è deducibile dalla conoscenza corrente dell'ambiente.

Definizione 8.1.2 *Una run t è una* traccia *se, ogni volta che $t = t' \cdot ?\langle M \rangle \cdot s$, t' è una traccia e il giudizio $O(t') \vdash M$ è derivabile.*

Per la definizione appena introdotta è immediata la verifica della seguente proprietà delle tracce.

Proposizione 8.1.3 *I prefissi di una traccia sono tracce.*

Dim. Il risultato vale per definizione di traccia. □

8.2 Caso Particolare: Chiavi Condivise Limitate a Nomi

Il modello più semplice basato su quello di Dolev-Yao, in grado di formalizzare protocolli che utilizzano crittografia simmetrica, assume che tutte le chiavi condivise siano atomiche, cioè o costanti o variabili che possono essere istanziate solo con costanti. Questo semplifica l'analisi della conoscenza in quanto l'insieme dei termini derivabili dall'agente ostile a partire da un insieme di termini T, è uguale a $synth(analz(T))$, dove *synth* e *analz* sono rispettivamente la chiusura per sintesi e per analisi degli insiemi di termini definite da Paulson [73]. Intuitivamente, $analz(T)$ è l'insieme di tutti i termini ottenuti scomponendo i termini in T in accordo con le regole che definiscono le capacità degli attaccanti e $synth(T)$ è l'insieme di tutti i termini ottenuti combinando e codificando i termini in T. Assumendo chiavi atomiche, l'analisi dei termini risulta lineare sulla profondità della struttura dei termini.
In letteratura esistono alcuni lavori, come quello di Amadio e Lugiez [4] e quello di Boreale [17], che si basano su tecniche simboliche. Entrambi questi lavori utilizzano un calcolo dei processi per modellare sistemi crittografici, ma si limitano a trattare il caso in cui le chiavi utilizzate negli algoritmi crittografici simmetrici sono ristrette a nomi. In questa sezione seguiamo il loro esempio restringendo la nostra analisi a questo caso particolare. In seguito, cerchiamo di definire un modello in grado di rappresentare il caso in cui le chiavi possono essere qualsiasi messaggio.
Il vincolo posto sulle chiavi porta a definire una nuova grammatica dei messaggi. Questa si differenzia da quella data in Definizione 6.1.2 per il fatto che le chiavi condivise sono ristrette a nomi.

Definizione 8.2.1 *La nuova grammatica per* messaggi *è definita nel seguente modo:*

M,N	::=	x	*variabili*
		n	*nomi*
		k^+	*chiave pubblica*
		k^-	*chiave privata*
		τ	*token*
		(M,N)	*coppia*
		$\{M\}_n$	*codifica a chiave condivisa*
		$\{\!\|M\|\!\}_k$	*codifica a chiave pubblica*
		$h(M)$	*funzione hash*

8.2.1 Atomi

Il limite sulle chiavi suggerisce di decomporre gli insiemi di messaggi in parti atomiche. L'idea è costruire un insieme i cui elementi sono irriducibili. Gli atomi sono così introdotti per semplificare la conoscenza dell'ambiente senza, però, perdere informazioni. Intuitivamente, gli atomi sono calcolati scomponendo i messaggi di un insieme, in accordo con le regole di inferenza del sistema deduttivo, ed eliminando i messaggi che appartengono alla conoscenza iniziale dell'ambiente. Definiamo ora la funzione Atoms e la funzione $\overline{\text{Atoms}}$ che consentono di calcolare un insieme atomico di messaggi contenente la stessa informazione dell'insieme di partenza.

Definizione 8.2.2 *La funzione* Atoms *è una funzione da insiemi di messaggi a insiemi di messaggi. La definizione formale è la seguente:*

$$
\begin{array}{lll}
(1) & \mathsf{Atoms}(\{M\} \cup K) & = \mathsf{Atoms}(K) \ \textit{se}\ M \in \mathcal{V} \cup \mathcal{T} \cup \mathcal{K}^{+} \\
(2) & \mathsf{Atoms}(\{(M,N)\} \cup K) & = \mathsf{Atoms}(\{M,N\} \cup K) \\
(3) & \mathsf{Atoms}(\{\{M\}_n,n\} \cup K) & = \mathsf{Atoms}(\{M,n\} \cup K) \\
(4) & \mathsf{Atoms}(\{[\![M]\!]_{k^+},k^-\} \cup K) & = \mathsf{Atoms}(\{M,k^-\} \cup K) \\
(5) & \mathsf{Atoms}(\{[\![M]\!]_{k^+}\} \cup K) & = \mathsf{Atoms}(K) \ \textit{se}\ K \vdash M \\
(6) & \mathsf{Atoms}(\{h(M)\} \cup K) & = \mathsf{Atoms}(K) \ \textit{se}\ K \vdash M \\
(7) & \mathsf{Atoms}(K) & = K \quad \textit{altrimenti}
\end{array}
$$

Facciamo un'analisi delle regole appena introdotte. La prima afferma che le variabili, i token e le chiavi pubbliche non appartengono all'insieme degli atomi. Queste entità, infatti, sono derivabili da qualsiasi insieme di messaggi e quindi non è indispensabile memorizzarle. Le successive tre regole sono equivalenti, rispettivamente, all'applicazione delle regole ($PROJ_i$), ($SKDEC$) e ($PKDEC$). Le regole (5) e (6) affermano che l'ambiente può cercare di decriptare un messaggio codificato attraverso un algoritmo crittografico asimmetrico o una funzione hash provando a cifrare tutti i messaggi da lui conosciuti. L'ultima regola viene applicata quando non è possibile fare alcun matching e restituisce l'insieme degli atomi. Allora la definizione di atomo asserisce che il sistema deduttivo dell'intruso non può derivare informazioni atomiche che non siano presenti nella conoscenza a sua disposizione. Questo significa che l'agente ostile non può intuire informazioni di cui non ha nessuna traccia fra le proprie conoscenze. Possiamo osservare che la funzione Atoms è ben definita in quanto la complessità dell'insieme decresce ad ogni passo.

Definizione 8.2.3 *Sia K un insieme di messaggi. L'insieme atomico di messaggi associato a K è calcolato nel seguente modo:*

$$
\overline{\mathsf{Atoms}}(K) = \begin{cases} K & \textit{se}\ \mathsf{Atoms}(K) = K \\ \overline{\mathsf{Atoms}}(\mathsf{Atoms}(K)) & \textit{altrimenti} \end{cases}
$$

Allora la funzione $\overline{\mathsf{Atoms}}$ è definita in modo tale da restituire un insieme irriducibile di messaggi. La precedente notazione si è resa necessaria per non confondere, nel seguito della tesi, l'insieme atomico risultante con gli insiemi utilizzati per calcolare tale insieme.

Esempio 8.2.4 *Dato l'insieme di messaggi* $K = \{(d,h(b)),\{(a,c)\}_d,h((a,d))\}$

$$
\overline{\mathsf{Atoms}}(K) = \{a,c,d,h(b)\}
$$

Ai fini della nostra tesi dimostriamo alcune proprietà della funzione $\overline{\mathsf{Atoms}}$. La funzione $\overline{\mathsf{Atoms}}$, dato in input un insieme di messaggi K, restituisce un insieme di messaggi da cui si deducono gli stessi messaggi deducibili da K, cioè contengono la stessa informazione dell'insieme di partenza. Inoltre, l'insieme $\overline{\mathsf{Atoms}}(K)$ è minimale, cioè ogni messaggio è deducibile da questo insieme o gli appartiene o può essere ottenuto per composizione dai messaggi dell'insieme.

Le regole della Definizione 8.2.2, evidenziano alcune proprietà che caratterizzano le sequenze utilizzate per calcolare gli insiemi di atomi. Queste proprietà sono in parte utili per dimostrare alcuni risultati, presentati in questa tesi, e in parte hanno lo scopo di fornire un'idea intuitiva più completa del funzionamento della funzione $\overline{\mathsf{Atoms}}$.

Osservazione 8.2.5 *Possiamo vedere l'insieme delle regole della Definizione 8.2.2, come un sistema di riscrittura da sinistra a destra dove le condizioni delle regole sono date dal matching con gli operandi. Pertanto, se* $\overline{\mathsf{Atoms}}(K) = K'$*, allora esiste una sequenza finita di espressioni* $E_0,...,E_n$ *dove ogni espressione* E_i *è della forma* $\mathsf{Atoms}(K_i)$ *per qualche* K_i*, e ogni espressione* E_{i+1} *è ricavata da* E_i *applicando le regole del sistema di riscrittura. In particolare, abbiamo che* $E_0 = \mathsf{Atoms}(K)$ *e* $E_n = \mathsf{Atoms}(K')$*.*

Introduciamo ora alcune definizioni che consentono di semplificare la descrizione di alcuni risultati illustrati nel seguito del capitolo. L'idea è considerare le sequenze di espressioni, ottenute applicando il sistema di riscrittura dell'Osservazione 8.2.5, in cui le variabili sono gli ultimi termini considerati.

Definizione 8.2.6 *Una sequenza finita di espressioni della forma* $\mathsf{Atoms}(K_i)$ *per qualche* K_i *è in* forma normale *se è ottenuta applicando le regole del sistema di riscrittura dell'Osservazione 8.2.5 e se le espressioni, ottenute applicando la regola (1) nel caso in cui il matching viene fatto con una variabile, sono in coda alla sequenza.*

Definizione 8.2.7 *Sia* $E_0,...,E_n$ *una sequenza in forma normale. Il* prefisso normale *della sequenza* $E_0,...,E_n$ *è il più grande prefisso della sequenza che non contiene le espressioni ottenute applicando la regola (1) nel caso in cui il matching viene fatto con una variabile.*

La scelta di utilizzare sequenze in forma normale non impone vincoli aggiuntivi all'analisi in quanto ogni sequenza di espressioni ne ha almeno una equivalente in forma normale come mostra il seguente lemma.

Lemma 8.2.8 *Sia K un insieme di messaggi e* $K' = \overline{\mathsf{Atoms}}(K)$*. Esiste una sequenza* $F_0,...,F_n$ *in forma normale in cui* $F_0 = \mathsf{Atoms}(K)$ *e* $F_n = \mathsf{Atoms}(K')$*.*

Dim. Sia K un insieme di messaggi e $K' = \overline{\mathsf{Atoms}}(K)$. Sia $E_0,...,E_n$ una sequenza di espressioni della forma $\mathsf{Atoms}(K_i)$ per qualche K_i, ottenuta applicando le regole di riscrittura dell'Osservazione 8.2.5, dove $E_0 = \mathsf{Atoms}(K)$ e $E_n = \mathsf{Atoms}(K')$. Applicando le regole nello stesso ordine con cui è stata ottenuta $E_0,...,E_n$ eccetto la regola (1) nel caso in cui il matching viene fatto con una variabile, costruiamo la nuova sequenza $F_0,...,F_{n-l}$ tale che $F_0 = E_0$ e $F_{n-l} = \mathsf{Atoms}(\{x_1,...,x_l\} \cup K')$. Applicando l volte la regola (1), otteniamo $F_n = \mathsf{Atoms}(K')$. Allora la sequenza $F_0,...,F_n$ è in forma normale, $F_0 = \mathsf{Atoms}(K)$ e $F_n = \mathsf{Atoms}(K')$. □

Abbiamo ristretto le chiavi condivise a nomi e quindi queste entità primitive assumono un'importanza fondamentale nella costruzione del modello. I prossimi tre lemmi mettono in luce il comportamento della funzione Atoms e della funzione $\overline{\mathsf{Atoms}}$ in presenza di nomi e di chiavi private. Il primo afferma che nessuna regola del sistema di riscrittura dell'Osservazione 8.2.5 elimina i nomi e le chiavi private. Il secondo estende questo risultato a

sequenze di espressioni ottenute applicando le regole del sistema di riscrittura. Il terzo lemma mostra come l'appartenenza di nomi e chiavi private all'insieme degli atomi è invariante sotto sostituzione.

Lemma 8.2.9 *Sia ζ un nome o una chiave pubblica e $E = \mathsf{Atoms}(K_i)$ e $F = \mathsf{Atoms}(K_j)$. Se F è ottenuta da E applicando una regola del sistema di riscrittura dell'Osservazione 8.2.5 e $\zeta \in K_i$, allora $\zeta \in K_j$.*

Dim. Sia ζ un nome o una chiave pubblica. Nel sistema di riscrittura dell'Osservazione 8.2.5 non esiste nessuna regola che elimina ζ. □

Lemma 8.2.10 *Sia ζ un nome o una chiave pubblica e $E_0,...,E_n$ una sequenza di espressioni della forma $\mathsf{Atoms}(K_i)$ per qualche K_i con $E_0 = \mathsf{Atoms}(K_0)$ e $E_n = \mathsf{Atoms}(K_n)$. Se $E_0,...,E_n$ è ottenuta applicando le regole del sistema di riscrittura dell'Osservazione 8.2.5 e $\zeta \in K_0$, allora $\zeta \in K_n$.*

Dim. Il risultato si ottiene dal Lemma 8.2.9 per induzione sulla lunghezza della sequenza ottenuta applicando le regole del sistema di riscrittura. □

Lemma 8.2.11 *Sia ζ un nome o una chiave privata, K un insieme di messaggi e R una sostituzione. Se $\zeta \in \overline{\mathsf{Atoms}}(K)$ allora $\zeta \in \overline{\mathsf{Atoms}}(K[R])$.*

Dim. Sia ζ un nome o una chiave privata, K un insieme di messaggi, $K_a = \overline{\mathsf{Atoms}}(K)$ e R una sostituzione. Per il Lemma 8.2.8, esiste una sequenza $F_0,...,F_n$ in forma normale in cui $F_0 = \mathsf{Atoms}(K)$ e $F_n = \mathsf{Atoms}(K_a)$. Sia $F_0,...,F_m$ il prefisso normale di $F_0,...,F_n$. Se $\zeta \in \overline{\mathsf{Atoms}}(K)$, dato che la regola (1) dell'Osservazione 8.2.5 non introduce nuovi messaggi, esiste $F_j = \mathsf{Atoms}(K_j)$ in $F_0,...,F_m$ tale che $\zeta \in K_j$. Applicando la sequenza di regole con cui è stata ottenuta $F_0,...,F_j$ a $K_j[R]$ otteniamo la nuova sequenza $E_0,...,E_n$ dove $E_0 = \mathsf{Atoms}(K[R])$ e $E_j = \mathsf{Atoms}(K_j[R])$. Allora $\zeta \in K_j[R]$ in quanto le sostituzioni sono funzioni da variabili a termini. Per il Lemma 8.2.10, se $\zeta \in K_j[R]$, allora $\zeta \in \overline{\mathsf{Atoms}}(K[R])$. □

La prossima proposizione è utilizzata per dimostrare alcuni risultati illustrati nel seguito di questa tesi e il suo enunciato esprime come, dato un insieme di messaggi K, l'insieme $\overline{\mathsf{Atoms}}(K)$ contiene la stessa informazione di K. Questo risultato consente di semplificare la struttura delle derivazioni in quanto la funzione Atoms simula l'applicazione delle regole di eliminazione, rendendo così la fase di raffinamento più efficiente. L'idea è sfruttare il fatto che i messaggi appartenenti ad $\overline{\mathsf{Atoms}}(K)$ sono deducibili da K e che gli elementi di K sono deducibili da $\overline{\mathsf{Atoms}}(K)$. I risultati intermedi, necessari per la dimostrazione, sono ottenuti studiando i passi eseguiti dal sistema di riscrittura per calcolare l'insieme degli atomi. Ricordiamo che si utilizza la notazione $K \vdash K'$ per indicare che da K si deducono tutti i messaggi di K', dove K e K' sono insiemi di messaggi.

Lemma 8.2.12 *Sia $E = \mathsf{Atoms}(K_i)$ e $F = \mathsf{Atoms}(K_j)$. Se F è ottenuta da E applicando una regola del sistema di riscrittura dell'Osservazione 8.2.5, allora $K_i \vdash K_j$ e $K_j \vdash K_i$.*

Dim. La dimostrazione è data per casi sulla regola del sistema di riscrittura applicata per passare da K_i a K_j.

Regola (1) : In questo caso $K_i = K_j \cup \{M\}$ con $M \in \mathcal{V} \cup \mathcal{T} \cup \mathcal{K}^+$. Allora $K_i \vdash K_j$ perché $K_j \subset K_i$ e $K_j \vdash K_i$ perché le variabili, i token e le chiavi pubbliche sono derivabili da qualsiasi insieme di messaggi.

Regola (2) : In questo caso $K_i = K_j \setminus \{M,N\} \cup \{(M,N)\}$. Abbiamo che $K_i \vdash M$ e $K_i \vdash N$, e che $K_j \vdash (M,N)$, allora $K_i \vdash K_j$ e $K_j \vdash K_i$.

Regola (3) : In questo caso $K_i = K_j \setminus \{M,n\} \cup \{\{M\}_n,n\}$. Abbiamo che $K_i \vdash M$ e che $K_j \vdash \{M\}_n$, allora $K_i \vdash K_j$ e $K_j \vdash K_i$.

Regola (4) : In questo caso $K_i = K_j \setminus \{M,k^-\} \cup \{\{\!|M|\!\}_{k^+},k^-\}$. Abbiamo che $K_i \vdash M$ e che $K_j \vdash \{\!|M|\!\}_{k^+}$, allora $K_i \vdash K_j$ e $K_j \vdash K_i$.

Regola (5) : In questo caso $K_i = K_j \cup \{\{\!|M|\!\}_{k^+}\}$ con $K_j \vdash M$. Allora $K_i \vdash K_j$ perché $K_j \subset K_i$ e che $K_j \vdash K_i$ perché $K_j \vdash \{\!|M|\!\}_{k^+}$.

Regola (6) : In questo caso $K_i = K_j \cup \{h(M)\}$ con $K_j \vdash M$. Allora $K_i \vdash K_j$ perché $K_j \subset K_i$ e che $K_j \vdash K_i$ perché $K_j \vdash h(M)$. □

Il risultato appena raggiunto mostra come i passi del sistema di riscrittura preservano il contenuto informativo degli insiemi a loro associati. Estendendo questo risultato, proviamo che $K \vdash \overline{\mathsf{Atoms}}(K)$ e $\overline{\mathsf{Atoms}}(K) \vdash K$.

Lemma 8.2.13 *Sia K un insieme di messaggi. Allora* $K \vdash \overline{\mathsf{Atoms}}(K)$ *e* $\overline{\mathsf{Atoms}}(K) \vdash K$.

Dim. Sia K un insieme di messaggi e $K' = \overline{\mathsf{Atoms}}(K)$. Consideriamo una sequenza di espressioni $E_0,...,E_n$ della forma $\mathsf{Atoms}(K_i)$ per qualche K_i ottenuta applicando le regole del sistema di riscrittura dell'Osservazione 8.2.5 dove $E_0 = \mathsf{Atoms}(K)$ e $E_n = \mathsf{Atoms}(K')$. Il risultato si ottiene dal Lemma 8.2.12 per induzione sulla lunghezza della sequenza ottenuta applicando le regole del sistema di riscrittura. □

Poiché dall'insieme di messaggi K si deducono tutti i messaggi che appartengono a $\overline{\mathsf{Atoms}}(K)$ e da $\overline{\mathsf{Atoms}}(K)$ si deducono tutti quelli di K, risulta evidente che K e $\overline{\mathsf{Atoms}}(K)$ hanno lo stesso contenuto informativo.

Proposizione 8.2.14 *Sia M un messaggio e K un insieme di messaggi. Allora il giudizio* $K \vdash M$ *è derivabile sse* $\overline{\mathsf{Atoms}}(K) \vdash M$ *è derivabile.*

Dim. Deriva direttamente dal Lemma 8.2.13 e dalla Proposizione 6.2.9. □

Quindi, l'insieme $\overline{\mathsf{Atoms}}(K)$, per come è stato definito, contiene la stessa informazione dell'insieme K. Infatti, dai due insiemi è possibile dedurre gli stessi messaggi, e questo consente di semplificare la struttura delle derivazioni in quanto le regole di eliminazione sono implicite nella funzione Atoms come si osserva nella proposizione seguente.

Proposizione 8.2.15 *Sia M un messaggio e K un insieme atomico di messaggi. Se $K \vdash M$, allora esiste un albero di prova che non utilizza regole di eliminazione.*

Dim. Sia M un messaggio, K un insieme atomico di messaggi e Γ un albero di prova di profondità p del giudizio $K \vdash M$. Dimostriamo il risultato per induzione sulla profondità dell'albero di prova.

Caso base : Se Γ ha profondità 1, allora l'albero è ottenuto applicando un assioma del sistema deduttivo.

Passo induttivo : Supponiamo che Γ abbia profondità $p > 1$. Allora Γ è costituito da sottoalberi di profondità minore. Per induzione, questi ultimi possono essere sostituiti da alberi costruiti utilizzando solo regole costruttive, cioè assiomi e regole di introduzione. Dimostriamo che esiste un albero di prova del giudizio $K \vdash M$ in cui non sono utilizzate regole di eliminazione, distinguendo i casi sull'ultima regola applicata in Γ. Se l'ultima regola applicata è una regola di introduzione il risultato vale. Se l'ultima regola applicata è una regola di eliminazione, distinguiamo i seguenti casi.

regola ($PROJ_i$) : Consideriamo solo il caso della regola ($PROJ_1$), in quanto il caso della regola ($PROJ_2$) è analogo. Sia Γ l'albero di prova del giudizio $K \vdash M$ e l'ultima regola applicata sia la regola ($PROJ_1$). Per ipotesi induttiva, nella derivazione del giudizio $K \vdash (M,N)$ non sono applicate regole di eliminazione, e, per definizione di atomo, (M,N) non appartiene a K.

$$(PROJ_1)\ \dfrac{(PAIR)\ \dfrac{\dfrac{\Gamma_1}{K \vdash M} \quad \dfrac{\Gamma_2}{K \vdash N}}{K \vdash (M,N)}}{K \vdash M}$$

Allora l'albero cercato, costruito solo con regole costruttive, è

$$\dfrac{\Gamma_1}{K \vdash M}$$

regola ($SKDEC$) : Sia Γ l'albero di prova del giudizio $K \vdash M$ e l'ultima regola applicata sia la regola ($SKDEC$). Per ipotesi induttiva, nella derivazione dei giudizi $K \vdash \{M\}_n$ e $K \vdash n$ non sono applicate regole di eliminazione. Allora l'unica regola costruttiva per derivare $K \vdash n$ è la regola (AX) e quindi $n \in K$. Per definizione di atomo, $\{M\}_n$ non appartiene a K.

$$(SKDEC)\ \dfrac{(SKENC)\ \dfrac{\dfrac{\Gamma_1}{K \vdash M} \quad \dfrac{\Gamma_2}{K \vdash n}}{K \vdash \{M\}_n} \quad \dfrac{\Gamma_2}{K \vdash n}}{K \vdash M}$$

Allora l'albero cercato, costruito solo con regole costruttive, è

$$\dfrac{\Gamma_1}{K \vdash M}$$

regola (*PKDEC*) : Sia Γ l'albero di prova del giudizio $K \vdash M$ e l'ultima regola applicata sia la regola (*PKDEC*). Per ipotesi induttiva, nella derivazione dei giudizi $K \vdash \{\!|M|\!\}_{k^+}$ e $K \vdash k^-$ non sono applicate regole di eliminazioni. Allora l'unica regola costruttiva per derivare $K \vdash k^-$ è la regola (*AX*) e quindi $k^- \in K$. Per definizione di atomo, $\{\!|M|\!\}_{k^+}$ non appartiene a K.

$$\begin{array}{l}(PKENC)\\(PKDEC)\end{array} \dfrac{\dfrac{\dfrac{\Gamma_1}{K \vdash M}}{K \vdash \{\!|M|\!\}_{k^+}} \quad \dfrac{\Gamma_2}{K \vdash k^-}}{K \vdash M}$$

Allora l'albero cercato, costruito solo con regole costruttive, è

$$\frac{\Gamma_1}{K \vdash M}$$

□

A questo punto è interessante notare la relazione tra le derivazioni semplici, introdotte nel capitolo precedente, e il risultato appena raggiunto.

Proposizione 8.2.16 *Sia M un messaggio e K un insieme atomico di messaggi. Allora una derivazione di $K \vdash M$ è semplice se e solo se non utilizza regole di eliminazione.*

Inoltre, i risultati appena raggiunti consentono di stabilire la relazione tra i messaggi deducibili da un insieme di atomi e i messaggi che appartengono all'insieme stesso. In particolare un nome, o una chiave privata, appartiene ad un insieme di atomi se e solo se esiste una derivazione dell'entità dall'insieme stesso.

Corollario 8.2.17 *Sia ζ un nome o una chiave privata e K un insieme atomico di messaggi. Allora $K \vdash \zeta$ sse $\zeta \in K$.*

Dim. Sia ζ un nome o una chiave privata e K un insieme di messaggi. Per la Proposizione 8.2.15, se $K \vdash \zeta$, esiste un albero di prova che non utilizza regole di eliminazione. Allora l'unica regola costruttiva per derivare $K \vdash \zeta$ è la regola (*AX*) e quindi $\zeta \in K$. Se $\zeta \in K$, allora $K \vdash \zeta$ utilizzando la regola (*AX*). □

La prossima proposizione mostra che l'insieme degli atomi è minimale e ciò è ottenuto stabilendo le condizioni di appartenenza di un messaggio ad un insieme atomico. Abbiamo già visto che nomi o chiavi private, se sono deducibili da un insieme di messaggi K, appartengono a $\overline{\mathsf{Atoms}}(K)$. I prossimi tre lemmi, presentati in questo paragrafo, studiano l'appartenenza all'insieme $\overline{\mathsf{Atoms}}(K)$ di messaggi ottenuti attraverso operatori di ordine superiore.

Lemma 8.2.18 *Sia n un nome, M un messaggio, K un insieme di messaggi e $K \vdash \{M\}_n$. Allora $\{M\}_n \in \overline{\mathsf{Atoms}}(K)$ sse $K \nvdash n$.*

Dim. ($\Rightarrow$) Sia $\{M\}_n \in \overline{\mathsf{Atoms}}(K)$. Supponiamo per assurdo che $K \vdash n$. Per la Proposizione 8.2.14 e per il Corollario 8.2.17, $n \in \overline{\mathsf{Atoms}}(K)$. Allora, per definizione di atomo, $\{M\}_n \notin \overline{\mathsf{Atoms}}(K)$. Allora $K \nvdash n$.

($\Leftarrow$) Se $K \vdash \{M\}_n$ e $K \nvdash n$, per la Proposizione 8.2.14, $\overline{\mathsf{Atoms}}(K) \vdash \{M\}_n$ e $\overline{\mathsf{Atoms}}(K) \nvdash n$. Per la Proposizione 8.2.15, esiste un albero di prova di $\overline{\mathsf{Atoms}}(K) \vdash \{M\}_n$ che utilizza solo regole di introduzione o assiomi. L'unica regola costruttiva applicabile è la regola (*AX*). Allora $\{M\}_n \in \overline{\mathsf{Atoms}}(K)$. □

Lemma 8.2.19 *Sia k^+ una chiave pubblica, M un messaggio, K un insieme di messaggi e $K \vdash \{\!|M|\!\}_{k^+}$. Allora $\{\!|M|\!\}_{k^+} \in \overline{\mathsf{Atoms}}(K)$ sse $K \nvdash M$.*

Dim. ($\Rightarrow$) Se $\{\!|M|\!\}_{k^+} \in \overline{\mathsf{Atoms}}(K)$, $\overline{\mathsf{Atoms}}(K) \vdash \{\!|M|\!\}_{k^+}$ applicando la regola (*AX*). Supponiamo per assurdo che $K \vdash M$. Allora, per definizione di atomo, $\{\!|M|\!\}_{k^+} \notin \overline{\mathsf{Atoms}}(K)$. Allora $K \nvdash M$.

($\Leftarrow$) Se $K \vdash \{\!|M|\!\}_{k^+}$ e $K \nvdash M$, per la Proposizione 8.2.14, $\overline{\mathsf{Atoms}}(K) \vdash \{\!|M|\!\}_{k^+}$ e $\overline{\mathsf{Atoms}}(K) \nvdash M$. Per la Proposizione 8.2.15, esiste un albero di prova di $\overline{\mathsf{Atoms}}(K) \vdash \{\!|M|\!\}_{k^+}$ che utilizza solo regole di introduzione e assiomi. L'unica regola costruttiva applicabile è la regola (*AX*). Allora $\{\!|M|\!\}_{k^+} \in \overline{\mathsf{Atoms}}(K)$. □

Lemma 8.2.20 *Sia M un messaggio e K un insieme di messaggi e $K \vdash h(M)$. Allora $h(M) \in \overline{\mathsf{Atoms}}(K)$ sse $K \nvdash M$.*

Dim. ($\Rightarrow$) Se $h(M) \in \overline{\mathsf{Atoms}}(K)$, $\overline{\mathsf{Atoms}}(K) \vdash h(M)$ applicando la regola (*AX*). Supponiamo per assurdo che $K \vdash M$. Allora, per definizione di atomo, $h(M) \notin \overline{\mathsf{Atoms}}(K)$. Allora $K \nvdash M$

($\Leftarrow$) Se $K \vdash h(M)$ e $K \nvdash M$, per la Proposizione 8.2.14, $\overline{\mathsf{Atoms}}(K) \vdash h(M)$ e $\overline{\mathsf{Atoms}}(K) \nvdash M$. Per la Proposizione 8.2.15, esiste un albero di prova di $\overline{\mathsf{Atoms}}(K) \vdash h(M)$ che utilizza solo regole di introduzione o assiomi. L'unica regola costruttiva applicabile è la regola (*AX*). Allora $h(M) \in \overline{\mathsf{Atoms}}(K)$. □

Il primo passo per mostrare che un insieme atomico è costituito da termini irriducibili è provare come la funzione $\overline{\mathsf{Atoms}}$ è idempotente, cioè, dato un insieme di messaggi K, $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K)) = \overline{\mathsf{Atoms}}(K)$.

Lemma 8.2.21 *La funzione $\overline{\mathsf{Atoms}}$ è idempotente.*

Dim. Sia K un insieme di messaggi e $K' = \overline{\mathsf{Atoms}}(K)$. L'insieme $\overline{\mathsf{Atoms}}(K')$ è calcolato mediante la funzione Atoms. In questo caso l'unica regola applicabile è la regola (7) della Definizione 8.2.2. Allora, per definizione di $\overline{\mathsf{Atoms}}$, $\overline{\mathsf{Atoms}}(K') = \overline{\mathsf{Atoms}}(K)$. □

Dimostriamo ora che ogni messaggio deducibile da un insieme di atomi o gli appartiene o è ottenuto per composizione dai messaggi dell'insieme. Per provare questo risultato, utilizziamo il seguente lemma il quale afferma che calcolare $\overline{\mathsf{Atoms}}(K \cup \{M\})$ equivale a calcolare

$\overline{\mathsf{Atoms}}(K)$ e poi applicare nuovamente la funzione $\overline{\mathsf{Atoms}}$ all'insieme ottenuto unito con il messaggio M.

Lemma 8.2.22 *Sia M un messaggio e K un insieme di messaggi. Allora $\overline{Atoms}(K \cup \{M\}) = \overline{Atoms}(\overline{Atoms}(K) \cup \{M\})$.*

Dim. Sia M un messaggio, K un insieme di messaggi, $K' = \overline{\mathsf{Atoms}}(K)$ e $K'' = \overline{\mathsf{Atoms}}(K \cup \{M\})$. Consideriamo la sequenza di espressioni E_0,...,E_n della forma $\mathsf{Atoms}(K_i)$ per qualche K_i, ottenuta applicando le regole di riscrittura dell'Osservazione 8.2.5, dove $E_0 = \mathsf{Atoms}(K)$ e $E_n = \mathsf{Atoms}(K'')$. Sia E_0,...,E_n costruita in modo tale che il prefisso E_0,...,E_j sia ottenuto senza utilizzare M e $E_j = \mathsf{Atoms}(K' \cup \{M\})$. Allora la sequenza E_0,...,E_n è uguale a quella usata per calcolare $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{M\})$. Allora $\overline{\mathsf{Atoms}}(K \cup \{M\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{M\})$. □

Proposizione 8.2.23 *Sia M un messaggio e K un insieme di messaggi. Per $K \vdash M$ derivabile, $\overline{Atoms}(K \cup \{M\}) = \overline{Atoms}(K)$.*

Dim. Sia M un messaggio e K un insieme di messaggi. Diamo una dimostrazione per induzione sulla complessità di M.

Caso base : Se M ha complessità 0, allora $M \in \mathcal{V} \cup \mathcal{N} \cup \mathcal{T} \cup \mathcal{K}^- \cup \mathcal{K}^+$. Se M è una variabile, un token o una chiave pubblica, allora $\overline{\mathsf{Atoms}}(K \cup \{M\}) = \overline{\mathsf{Atoms}}(K)$, per la Definizione 8.2.2 punto (1). Se M è un nome o una chiave privata, per la Proposizione 8.2.14 e il Corollario 8.2.17, $M \in \overline{\mathsf{Atoms}}(K)$. Per il Lemma 8.2.22, $\overline{\mathsf{Atoms}}(K \cup \{M\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{M\})$. Allora $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{M\})$ è uguale a $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K))$. Per il Lemma 8.2.21, $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K)) = \overline{\mathsf{Atoms}}(K)$.

Passo induttivo : Supponiamo che M abbia complessità $p > 0$. Distinguiamo i casi sugli operatori di alto livello.

case $M = (M_1, M_2)$: Per la Definizione 8.2.2 punto (2), $\overline{\mathsf{Atoms}}(K \cup \{(M_1, M_2)\})$ è uguale a $\overline{\mathsf{Atoms}}(K \cup \{M_1\} \cup \{M_2\})$. Per il Lemma 8.2.22, quest'ultimo equivale a $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K \cup \{M_1\}) \cup \{M_2\})$. Per la regola (*PAIR*), $K \vdash M_1$ e $K \vdash M_2$. Per induzione, dato che $K \vdash M_1$, $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K \cup \{M_1\}) \cup \{M_2\}) = \overline{\mathsf{Atoms}}(K \cup \{M_2\})$. Per induzione, dato che $K \vdash M_2$, $\overline{\mathsf{Atoms}}(K \cup \{M_2\}) = \overline{\mathsf{Atoms}}(K)$.

case $M = \{M'\}_n$: Sia $K \vdash \{M'\}_n$. Distinguiamo due casi.

1. $K \nvdash n$: In questo caso, per il Lemma 8.2.18, $\{M'\}_n \in \overline{\mathsf{Atoms}}(K)$. Per il Lemma 8.2.22, $\overline{\mathsf{Atoms}}(K \cup \{\{M'\}_n\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{\{M'\}_n\})$. Allora $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{\{M'\}_n\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K))$. Per il Lemma 8.2.21, quest'ultimo è uguale a $\overline{\mathsf{Atoms}}(K)$.
2. $K \vdash n$: In questo caso, per la Proposizione 8.2.14 e il Corollario 8.2.17, $n \in \overline{\mathsf{Atoms}}(K)$. Per la Definizione 8.2.2 punto (3), $\overline{\mathsf{Atoms}}(K \cup \{\{M'\}_n\})$ è uguale a $\overline{\mathsf{Atoms}}(K \cup \{M'\})$. Per la regola (*SKDEC*), $K \vdash M'$. Allora, per induzione, $\overline{\mathsf{Atoms}}(K \cup \{M'\}) = \overline{\mathsf{Atoms}}(K)$.

case $M = \{\!|M'|\!\}_{k^+}$: Sia $K \vdash \{\!|M'|\!\}_{k^+}$. Distinguiamo due casi.

1. $K \vdash M'$: In questo caso $\overline{\mathsf{Atoms}}(K \cup \{\{\!|M'|\!\}_{k^+}\}) = \overline{\mathsf{Atoms}}(K)$, per la Definizione 8.2.2 punto (5).
2. $K \nvdash M'$: Per il Lemma 8.2.19, il messaggio $\{\!|M'|\!\}_{k^+} \in \overline{\mathsf{Atoms}}(K)$. Per il Lemma 8.2.22, $\overline{\mathsf{Atoms}}(K \cup \{\{\!|M'|\!\}_{k^+}\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{\{\!|M'|\!\}_{k^+}\})$. Allora $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{\{\!|M'|\!\}_{k^+}\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K))$. Per il Lemma 8.2.21, quest'ultimo è uguale a $\overline{\mathsf{Atoms}}(K)$.

case $M = h(M')$: Sia $K \vdash h(M')$. Distinguiamo due casi.

1. $K \vdash M'$: In questo caso $\overline{\mathsf{Atoms}}(K \cup \{h(M')\}) = \overline{\mathsf{Atoms}}(K)$, per la Definizione 8.2.2 punto (6).
2. $K \nvdash M'$: Per il Lemma 8.2.20, il messaggio $h(M') \in \overline{\mathsf{Atoms}}(K)$. Per il Lemma 8.2.22, $\overline{\mathsf{Atoms}}(K \cup \{h(M')\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{h(M')\})$. Allora $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{h(M')\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K))$. Per il Lemma 8.2.21, quest'ultimo è uguale a $\overline{\mathsf{Atoms}}(K)$. □

8.2.2 Modello Simbolico

Per rendere possibile la verifica automatica delle proprietà di sicurezza definiamo un modello che sia finito e che sia in grado di rappresentare esattamente le informazioni contenute nel modello concreto e lo chiamiamo *modello simbolico*. A tal fine introduciamo una procedura in grado di calcolare le sostituzioni per cui una run possa diventare una traccia. Dimostriamo che la nuova procedura termina per ogni run, garantendo così che il modello sia finito, e che restituisca effettivamente le sostituzioni per cui una run sia una traccia. In questo modo siamo sicuri di raffinare il modello ottenuto mediante la semantica simbolica eliminando i controesempi spuri. Costruiamo il modello simbolico, a partire dal grafo simbolico delle transizioni, determinando quali cammini del grafo simbolico hanno una controparte nel modello concreto. L'insieme delle tracce, ottenuto analizzando ogni run del grafo simbolico, costituisce il modello simbolico.

Definizione 8.2.24 *Il* modello di una run s, $\mathsf{Model}_{at}(\epsilon, s)$, *è l'insieme delle sostituzioni calcolate attraverso la procedura in Figura 8.1.*

Intuitivamente, questa procedura stabilisce che una traccia concatenata con una run vuota è ancora una traccia senza dover introdurre vincoli. Per calcolare le sostituzioni Q tali per cui una traccia t concatenata con una run non vuota s sia ancora una traccia, si deve tener conto della prima azione della sequenza s. Se la sequenza s è della forma $!\langle M\rangle \cdot s'$, cioè la prima azione di s è un'azione di output, la soluzione è data dalle sostituzioni Q, calcolate a partire da s', che rendono $(t \cdot s)$ una traccia. Se la sequenza s è della forma $?\langle M\rangle \cdot s'$, cioè la prima azione di s è un'azione di input, eseguiamo la procedura $\mathsf{Constraints}$, cioè calcoliamo le sostituzioni R che rendono il messaggio M derivabile dall'insieme $\overline{\mathsf{Atoms}}(O(t))$. In base al risultato di questa procedura, distinguiamo due casi. Se la soluzione calcolata è la sostituzione vuota, il risultato di Model_{at} è dato dalle sostituzioni Q, calcolate a partire da s', che rendono $(t \cdot s)$ una traccia. Se la soluzione di $\mathsf{Constraints}$ è diversa dalla sostituzione vuota,

```
Model_at(t,s) =
    case s of
      ϵ              ⇒ {[ ]}
      !⟨M⟩ · s′ ⇒ Model_at(t·!⟨M⟩,s′)
      ?⟨M⟩ · s′ ⇒ for R ∈ Constraints(Atoms(O(t)),M)
                      union
                            if R = [ ]
                            then Model_at(t·?⟨M⟩,s′)
                            else
                              for R′ ∈ Model_at(ϵ,(t · s)[R])
                              union { R@R′ }
```

Figura 8.1. Procedura Model_{at} per i modelli simbolici

il risultato di Model_{at} è dato dalle sostituzioni $R@R'$ tali che $(t \cdot s)[R][R']$ è una traccia dove $R' \in \mathsf{Model}_{at}(\epsilon,(t \cdot s)[R])$.
In Figura 8.2 è presentata una procedura alternativa per calcolare i modelli simbolici di una run, necessaria per semplificare la dimostrazione della completezza del modello.

```
Model_at(t,s) =
    case s of
      ϵ              ⇒ {[ ]}
      s′·!⟨M⟩   ⇒ Model_at(t,s′)
      s′·?⟨M⟩   ⇒ for R′ ∈ Model_at(t,s′)
                      union for R_0 ∈ Constraints(Atoms(O(t · s′)[R′]),M[R′])
                            union if R_0 = [ ]
                                  then {R′}
                                  else for R_1 ∈ Model_at(ϵ,(t · s)[R′@R_0])
                                       union { R′@R_0@R_1 }
```

Figura 8.2. Procedura Model_{at} per i modelli simbolici

Le prossime due proposizioni sono fondamentali per dimostrare che il modello simbolico è finito e quindi utilizzabile per una verifica automatica delle proprietà di sicurezza dei protocolli. I loro enunciati esprimono, rispettivamente, che la procedura Model_{at} termina e calcola le sostituzioni per cui una run è una traccia del modello simbolico.

Proposizione 8.2.25 *Per ogni traccia t e run s, il programma* $\mathsf{Model}_{at}(t,s)$ *termina.*

Dim. Sia t una traccia qualsiasi e s una run di lunghezza p con n variabili. Dimostriamo la terminazione del programma $\mathsf{Model}_{at}(t,s)$ di Figura 8.1 per induzione sulla struttura di s, cioè un ordine lessicografico che tiene conto del numero di variabili in s e della lunghezza di s.

Caso base : Se s ha lunghezza 0, allora $s = \epsilon$. In questo caso la procedura $\mathsf{Model}_{at}(t,\epsilon)$ termina.

Passo induttivo : Supponiamo che s sia una run di lunghezza $p > 0$ con n variabili. Distinguiamo i casi sulla prima azione di s.

case $s = !\langle M\rangle \cdot s'$: Per definizione di traccia, $t{\cdot}!\langle M\rangle$ è una traccia. Per induzione, dato che $|SVars(s)| = |SVars(s')|$ e la lunghezza di s' è minore di quella di s, $\mathsf{Model}_{at}(t{\cdot}!\langle M\rangle,s')$ termina. Per definizione di Model_{at}, $\mathsf{Model}_{at}(t{\cdot}!\langle M\rangle,s') = \mathsf{Model}_{at}(t,!\langle M\rangle \cdot s')$. Allora $\mathsf{Model}_{at}(t,!\langle M\rangle \cdot s')$ termina.

case $s = ?\langle M\rangle \cdot s'$: In questo caso $\mathsf{Model}_{at}(t,s)$ richiama $\mathsf{Constraints}(\overline{\mathsf{Atoms}}(O(t)),M)$. Per il Teorema 7.2.10, la procedura Constraints termina e restituisce le sostituzioni R tali che $\overline{\mathsf{Atoms}}(O(t))[R] \vdash M[R]$. Distinguiamo i casi sul risultato della procedura Constraints:

1. $R = [\ ]$: Per definizione di traccia, $t{\cdot}?\langle M\rangle$ è una traccia. Per induzione, dato che $|SVars(s)| = |SVars(s')|$ e la lunghezza di s' è minore di quella di s, $\mathsf{Model}_{at}(t{\cdot}?\langle M\rangle,s')$ termina. Per definizione di Model_{at}, $\mathsf{Model}_{at}(t{\cdot}?\langle M\rangle,s') = \mathsf{Model}_{at}(t,?\langle M\rangle \cdot s')$. Allora $\mathsf{Model}_{at}(t,?\langle M\rangle \cdot s')$ termina.
2. $R \neq [\ ]$: Per definizione di traccia, $(t{\cdot}?\langle M\rangle)[R]$ è una traccia. Per induzione, dato che $|SVars((t \cdot s)[R])| < |SVars(t \cdot s)|$, $\mathsf{Model}_{at}(\epsilon,(t \cdot s)[R])$ termina. Per definizione di Model_{at}, $\mathsf{Model}_{at}(\epsilon,(t \cdot s)[R])$ è uguale a $\mathsf{Model}_{at}(t,?\langle M\rangle \cdot s')$. Allora $\mathsf{Model}_{at}(t,?\langle M\rangle \cdot s')$ termina. □

Proposizione 8.2.26 *Per ogni traccia t, run s e per ogni* $R \in \mathsf{Model}_{at}(t,s)$*, la run* $(t \cdot s)[R]$ *è una traccia.*

Dim. Sia t una traccia qualsiasi e s una run di lunghezza p, con n variabili. Diamo una dimostrazione per induzione sulla struttura di s, cioè un ordine lessicografico che tiene conto del numero delle variabili contenute in s e della lunghezza della run.

Caso base : Se s ha lunghezza 0, allora $s = \epsilon$. In questo caso $\mathsf{Model}_{at}(t,\epsilon) = [\ ]$ e $t = t \cdot \epsilon$. Allora $t \cdot \epsilon$ è una traccia.

Passo induttivo : Supponiamo che s sia una run di lunghezza $p > 0$ con n variabili. Distinguiamo i casi sulla prima azione di s.

case $s = !\langle M\rangle \cdot s'$: Per definizione di Model_{at}, $R \in \mathsf{Model}_{at}(t{\cdot}!\langle M\rangle,s')$. Per induzione, dato che $|SVars(s)| = |SVars(s')|$ e la lunghezza di s' è minore della lunghezza di s, $(t{\cdot}!\langle M\rangle \cdot s')[R] = (t \cdot s)[R]$ è una traccia.

case $s = ?\langle M\rangle \cdot s'$: In questo caso $\mathsf{Model}_{at}(t,s)$ calcola $\mathsf{Constraints}(\overline{\mathsf{Atoms}}(O(t)),M)$. Per il Teorema 7.2.10, la procedura Constraints restituisce le sostituzioni R' tali che $\overline{\mathsf{Atoms}}(O(t))[R'] \vdash M[R']$. Distinguiamo due casi sul risultato della procedura Constraints:

1. $R' = [\ ]$: Per definizione di Model_{at}, $R \in \mathsf{Model}_{at}(t{\cdot}?\langle M\rangle,s')$. Per induzione, dato che $|SVars(s)| = |SVars(s')|$ e la lunghezza di s' è minore della lunghezza di s, $(t{\cdot}?\langle M\rangle \cdot s')[R] = (t \cdot s)[R]$ è una traccia.

2. $R' \neq [\,]$: Per definizione di Model_{at}, la sostituzione $R = R'@R''$ dove $R'' \in \mathsf{Model}_{at}(\epsilon,(t \cdot s)[R'])$. Per induzione, dato che $|SVars((t \cdot s)[R'])| < |SVars(t \cdot s)|$, $(t{\cdot}?\langle M\rangle \cdot s')[R'][R''] = (t \cdot s)[R'][R'']$ è una traccia. □

Corollario 8.2.27 *Una run s è una traccia se e solo se* $[\,] \in \mathsf{Model}_{at}(\epsilon,s)$.

Dim. Deriva direttamente dalla Proposizione 8.2.26 e dalla definizione di Model_{at}. □

Abbiamo dimostrato che esiste una procedura che calcola le sostituzioni per cui una run risulta una traccia. Per la verifica delle proprietà di sicurezza non è importante che una particolare sequenza di interazioni rispetti tali proprietà, ma ogni possibile esecuzione del protocollo deve soddisfarle. Estendiamo la procedura Model_{at} in modo da analizzare ogni run del grafo simbolico, costruendo così un modello finito in grado di rappresentare esattamente le informazioni contenute nel modello concreto.
Dato che abbiamo limitato le chiavi condivise a nomi, calcoliamo un modello in cui le sequenze di interazioni rispettino tale vincolo, determinando l'insieme delle variabili che occorrono nella sequenza come chiavi condivise e sostituendole con i nomi usati nel processo che modella il protocollo.

Definizione 8.2.28 *La funzione KVars è una funzione da messaggi a variabili. Diamo di seguito la definizione formale.*

$$
\begin{array}{lcl}
KVars(M) & = & \emptyset \quad M \in \mathcal{V} \cup \mathcal{N} \cup \mathcal{T} \cup \mathcal{K}^+ \cup \mathcal{K}^- \\
KVars((M,N)) & = & KVars(M) \cup KVars(N) \\
KVars(\{M\}_N) & = & \begin{cases} KVars(M) & se \quad N \notin \mathcal{V} \\ KVars(M) \cup \{N\} & se \quad N \in \mathcal{V} \end{cases} \\
KVars(\{\!|M|\!\}_k) & = & KVars(M) \\
KVars(h(M)) & = & KVars(M)
\end{array}
$$

Quindi la funzione *KVars* restituisce l'insieme delle variabili usate come chiavi condivise in un messaggio. Per ottenere sequenze di interazioni che rispettano i limiti imposti alle chiavi, estendiamo la definizione precedente in modo da calcolare l'insieme delle variabili che occorrono come chiavi condivise nei messaggi di una sequenza di interazioni, e quindi sostituiamo tali variabili con i nomi utilizzati dal processo che modella il protocollo attraverso un'istanziazione delle variabili di tipo brute force.

Definizione 8.2.29 *La funzione KSVars è una funzione da sequenze di interazioni a variabili. Data una sequenza di interazione s, KSVars(s) restituisce l'insieme delle variabili che occorrono come chiavi condivise nei messaggi della sequenza.*

Definizione 8.2.30 *Dato un processo chiuso P, definiamo l'insieme* $\mathsf{Run}(P)$ *come*

$$
\mathsf{Run}(P) = \left\{ s[\rho] \;\middle|\; \begin{array}{l} \epsilon\ ;\ P \rightsquigarrow^*\ s\ ;\ Q, \\ \rho : KSVars(s) \rightarrow \mathsf{Name}(P), \\ s[\rho] \text{ ha messaggi con chiavi condivise atomiche} \end{array} \right\}
$$

dove $\mathsf{Name}(P)$ *è l'insieme dei nomi usati in P.*

Introduciamo ora il modello simbolico utilizzato per verificare le proprietà di sicurezza di un protocollo. Questo modello è costruito calcolando, per ogni run ottenuta mediante la semantica simbolica, i vincoli per i quali il comportamento risulta computazionalmente valido.

Definizione 8.2.31 *Il* modello simbolico *di un processo chiuso P è l'insieme*

$$\mathsf{Model}_{at}(P) = \{s[R] \mid s \in \mathsf{Run}(P),\ R \in \mathsf{Model}_{at}(\epsilon, s)\}$$

8.2.3 Rispettabilità

Per mostrare che il modello simbolico è corretto e completo è necessario trovare uno strumento che consenta di mettere in relazione gli elementi del modello simbolico con quelli del modello concreto e quindi di verificare se i comportamenti del modello simbolico abbiano una controparte in quello concreto. La differenza tra le sequenze di interazioni nei due modelli consiste nel fatto che in quelle del modello simbolico occorrono variabili. Risulta evidente che tale strumento possa assumere la forma di sostituzioni ground.

Esempio 8.2.32 *Sia* $\mathsf{Model}(\epsilon, !\langle\{a\}_k\rangle \cdot ?\langle\{x\}_y\rangle) = \{[\,], [x \mapsto a, y \mapsto k]\}$. *Intuitivamente, in questo modello, la sostituzione vuota codifica le soluzione* ρ *con* $\rho(x) = M$ *e* $\rho(y) = N$ *tale che* $\{a\}_k \vdash M$ *e* $\{a\}_k \vdash N$, *mentre la sostituzione* $[x \mapsto a, y \mapsto k]$ *codifica la soluzione* ρ *con* $\rho(x) = a$ *e* $\rho(y) = k$.

Introduciamo il concetto di rispettabilità che serve per determinare le sostituzioni ground per cui una traccia del modello simbolico può diventare una sequenza di interazioni del modello concreto.

Definizione 8.2.33 *Sia s una run e sia* ρ *una sostituzione ground. Diciamo che* ρ rispetta *s se, per ogni variabile x in s, il giudizio* $O(s_x)[\rho] \vdash x[\rho]$ *è derivabile, dove* s_x *denota il più grande prefisso di s che non contiene x.*

Intuitivamente, una sostituzione ground rispetta una run se, data una sequenza di interazioni, siamo in grado di dedurre i messaggi sostituiti alle variabili, attraverso lo stato di conoscenza precedente l'introduzione delle variabili, dove la prima occorrenza di una variabile deve essere in un'azione di input. La nozione appena introdotta è molto simile a quella utilizzata da Boreale [18] per determinare se una sostituzione ground soddisfa una traccia simbolica e quindi per dimostrare la consistenza delle tracce.

Definizione 8.2.34 *Sia s una run. Diciamo che s è* rispettabile *se esiste una sostituzione ground* ρ *che la rispetta.*

Esempio 8.2.35 *Sia* $s = !\langle\{\!|a|\!\}_{k^+}\rangle \cdot !\langle b\rangle \cdot ?\langle\{x\}_n\rangle$ *una run e* $\rho = [x \mapsto b]$ *una sostituzione ground. Allora* ρ *rispetta s, ma* $s[\rho]$ *non è una traccia ground in quanto* $\{\{\!|a|\!\}_{k^+}, b\} \not\vdash \{b\}_n$.

L'esempio appena visto evidenzia come la nozione di rispettabilità è più debole di quella di traccia ground e quindi come non è sufficiente per determinare quali run simboliche hanno una controparte nel modello concreto. Comunque, per ogni traccia del modello simbolico esiste una sostituzione ground che la rispetta, e la combinazione della sequenza di interazione e della sostituzione che la rispetta restituisce una traccia del modello concreto. In questo modo la rispettabilità fornisce uno strumento per passare dal modello simbolico a quello concreto.
Con la seguente proposizione dimostriamo una relazione tra l'insieme di atomi, calcolato a partire dallo stato di conoscenza di una traccia, e una sostituzione ground che rispetta la traccia stessa. In particolare, illustriamo come l'insieme degli atomi è invariante sotto sostituzione.

Proposizione 8.2.36 *Per ogni traccia t e sostituzione ground ρ tale che $t[\rho]$ è una traccia ground, se ρ rispetta t, allora $\overline{\mathsf{Atoms}}(O(t)[\rho]) = \overline{\mathsf{Atoms}}(O(t))[\rho]$.*

Dim. Sia t una traccia e ρ una sostituzione ground che rispetta t. Ordiniamo le variabili di t in base alla loro occorrenza nella traccia. Per il Lemma 8.2.8, esiste una sequenza $F_0,...,F_n$ in forma normale, in cui $F_0 = \mathsf{Atoms}(O(t))$ e $F_n = \mathsf{Atoms}(\overline{\mathsf{Atoms}}(O(t)))$. Sia $F_0,...,F_m$ il prefisso normale di $F_0,...,F_n$. Allora $F_m = \mathsf{Atoms}(K_m)$ e $K_m = \{x_1,...,x_l\} \cup \overline{\mathsf{Atoms}}(O(t))$. Applichiamo la sequenza di regole con cui è stata ottenuta $F_0,...,F_m$ a $O(t)[\rho]$. Sia $E_0,...,E_m$ la nuova sequenza dove $E_0 = \mathsf{Atoms}(O(t)[\rho])$ e $E_m = \mathsf{Atoms}(\{x_1[\rho],...,x_l[\rho]\} \cup \overline{\mathsf{Atoms}}(O(t))[\rho])$. Allora $\overline{\mathsf{Atoms}}(O(t)[\rho]) = \overline{\mathsf{Atoms}}(\{x_1[\rho],...,x_l[\rho]\} \cup \overline{\mathsf{Atoms}}(O(t))[\rho])$. Diamo una dimostrazione per induzione sul numero di variabili in $X = \{x_1,...,x_l\}$.

Caso base : Se $|X| = 0$, allora $\overline{\mathsf{Atoms}}(O(t)[\rho]) = \overline{\mathsf{Atoms}}(O(t))$ in quanto le sostituzioni sono funzioni da variabili a messaggi. Inoltre, per lo stesso motivo, $\overline{\mathsf{Atoms}}(O(t)) = \overline{\mathsf{Atoms}}(O(t))[\rho]$. Allora $\overline{\mathsf{Atoms}}(O(t)[\rho]) = \overline{\mathsf{Atoms}}(O(t))[\rho]$.

Passo induttivo : Supponiamo $|X| = n$ con $n > 0$. Sia x_i la variabile di indice maggiore e t' il più grande prefisso di t in cui non occorre x_i. Per il Lemma 8.1.3, t' è una traccia e, per definizione di rispettabilità, ρ rispetta t'. Per ipotesi induttiva, $\mathsf{Atoms}(\{x_1[\rho],...,x_i[\rho]\} \cup \overline{\mathsf{Atoms}}(O(t))[\rho]) = \mathsf{Atoms}(\{x_i[\rho]\} \cup \overline{\mathsf{Atoms}}(O(t)[\rho]))$. Per definizione di rispettabilità, $O(t')[\rho] \vdash x_i[\rho]$. Dato che $O(t')[\rho] \subseteq O(t)[\rho]$, $O(t)[\rho] \vdash O(t')[\rho]$. Per la Proposizione 8.2.14, $\overline{\mathsf{Atoms}}(O(t)[\rho]) \vdash O(t')[\rho]$. Per la Proposizione 6.2.9, $\overline{\mathsf{Atoms}}(O(t)[\rho]) \vdash x_i[\rho]$. Allora, per la Proposizione 8.2.23, $\mathsf{Atoms}(\{x_i[\rho]\} \cup \overline{\mathsf{Atoms}}(O(t)[\rho])) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(O(t)[\rho]))$. Quest'ultimo, per il Lemma 8.2.21, è uguale a $\overline{\mathsf{Atoms}}(O(t)[\rho])$. Allora $\overline{\mathsf{Atoms}}(O(t)[\rho]) = \overline{\mathsf{Atoms}}(O(t))[\rho]$. □

Le ipotesi della proposizione precedente sono fondamentali per la validità della stessa e, infatti, tale risultato non può essere esteso a qualsiasi insieme di messaggi e a qualsiasi sostituzione ground come mostra il seguente esempio.

Esempio 8.2.37 *Sia $K = \{\{a\}_n,x\}$ un insieme di messaggi e $\rho = [x \mapsto n]$ una sostituzione ground. Allora $\overline{\mathsf{Atoms}}(K)[\rho] = \{\{a\}_n\}$, mentre $\overline{\mathsf{Atoms}}(K[\rho]) = \{a,n\}$.*

Le prossime due proposizioni sono fondamentali per dimostrare la correttezza del modello, cioè per provare come ogni traccia del modello simbolico ha una controparte nel modello concreto. La prima afferma che per ogni traccia esiste una sostituzione ground che la rispetta. La seconda proposizione dimostra che una traccia a cui è applicata una sostituzione che la rispetta è una sequenza di interazione del modello concreto.

Proposizione 8.2.38 *Per ogni traccia t, esiste una sostituzione ground ρ tale che ρ rispetta t.*

Dim. Sia t una traccia di lunghezza p. La dimostrazione è data per induzione sulla lunghezza di t.

Caso base : Se t ha lunghezza 0, $t = \epsilon$. In questo caso t non contiene variabili. Allora ogni sostituzione ground ρ rispetta t.

Passo induttivo : Supponiamo che t sia una traccia di lunghezza $p > 0$. Per la Proposizione 8.1.3, ogni prefisso di t è una traccia. Sia t' il prefisso di t di lunghezza $p - 1$. Per induzione esiste una sostituzione ground ρ che rispetta t'. Dimostriamo che esiste una sostituzione ground che rispetta t, distinguendo i casi sull'ultima azione di t.

case $t = t' \cdot !\langle M \rangle$: In questo caso, per la definizione di traccia, in M non occorrono variabili che non occorrono in $O(t')$. Allora ρ rispetta t.

case $t = t' \cdot ?\langle M \rangle$: Sia ρ' una sostituzione tale che, per ogni variabile x in M ma non in $O(t')$, $O(t')[\rho] \vdash x[\rho']$. Consideriamo la sostituzione ground $\overline{\rho} = \rho @ \rho'$. Per definizione di rispettabilità, $\overline{\rho}$ rispetta t. □

A questo punto non rimane che dimostrare come ogni traccia del modello simbolico, a cui è applicata una sostituzione ground che la rispetta, è una traccia del modello concreto e quindi come ogni elemento del modello simbolico è un'astrazione di qualche traccia di quello concreto. Questo risultato è ottenuto provando che, dato un messaggio M, un insieme di messaggi K e una sostituzione ground ρ, se $K \vdash M$ e, per ogni variabile x in M ma non in K, $K[\rho] \vdash x[\rho]$, allora il giudizio $K[\rho] \vdash M[\rho]$ è derivabile. Il motivo di ciò è che, nella derivazione di $K \vdash M$, l'unica regola utilizzabile per dedurre le variabili in M ma non in K, è la regola (*VAR*).

Lemma 8.2.39 *Sia M un messaggio, K un insieme di messaggi, ρ una sostituzione che rende sia $K[\rho]$ che $M[\rho]$ ground. Se $K \vdash M$ e, per ogni variabile x in M ma non in K, $K[\rho] \vdash x[\rho]$, allora $K[\rho] \vdash M[\rho]$.*

Dim. Sia M un messaggio, K un insieme di messaggi, $K' = \overline{\mathsf{Atoms}}(K)$ e ρ una sostituzione ground. Per la Proposizione 8.2.14, un messaggio è derivabile da un insieme K se e solo se è derivabile da $\overline{\mathsf{Atoms}}(K)$. Diamo una dimostrazione per induzione sulla complessità di M.

Caso base : Se M ha complessità 0, allora $M \in \mathcal{V} \cup \mathcal{N} \cup \mathcal{T} \cup \mathcal{K}^+ \cup \mathcal{K}^-$. Se il messaggio M è un token, un nome, una chiave pubblica o una chiave privata il lemma è banalmente verificato perché queste entità non contengono variabili. Se M è una variabile, il lemma vale per ipotesi.

Passo induttivo : Supponiamo che M abbia complessità $p > 0$. Distinguiamo i casi sugli operatori di alto livello.

case $M = (M_1,M_2)$: Per ipotesi, $K' \vdash (M_1,M_2)$. Allora $K' \vdash M_1$ e $K' \vdash M_2$. Sia x una variabile in M ma non in K. Allora x è in M_1 o in M_2. Inoltre, $\overline{\mathsf{Atoms}}(K[\rho]) \vdash x[\rho]$. Allora, per induzione, $\overline{\mathsf{Atoms}}(K[\rho]) \vdash M[\rho]$.

case $M = \{M'\}_n$: Per ipotesi, $K' \vdash \{M'\}_n$. Sia x una variabile in M ma non in K. Per la Proposizione 8.2.15, esiste un albero di prova di $K' \vdash \{M'\}_n$ in cui non sono utilizzate regole di eliminazione. L'unica regola costruttiva applicabile è (*SKENC*). Quindi $K' \vdash M'$ e $K' \vdash n$. Dato che i nomi non contengono variabili, x è in M'. Inoltre, $\overline{\mathsf{Atoms}}(K[\rho]) \vdash x[\rho]$. Per il Lemma 8.2.11 e il Corollario 8.2.17, $\overline{\mathsf{Atoms}}(K[\rho]) \vdash n$. Allora, per induzione, $\overline{\mathsf{Atoms}}(K[\rho]) \vdash M[\rho]$.

case $M = \{\!|M'|\!\}_{k^+}$: Per ipotesi, $K' \vdash \{\!|M'|\!\}_{k^+}$. Sia x una variabile in M ma non in K. Per la Proposizione 8.2.15, esiste un albero di prova di $K' \vdash \{\!|M'|\!\}_{k^+}$ in cui non sono utilizzate regole di eliminazione. L'unica regola costruttiva applicabile è (*PKENC*). Quindi $K' \vdash M'$. Dato che le chiavi pubbliche non contengono variabili, x è in M'. Inoltre, $\overline{\mathsf{Atoms}}(K[\rho]) \vdash x[\rho]$. Allora, per induzione, $\overline{\mathsf{Atoms}}(K[\rho]) \vdash M[\rho]$.

case $M = h(M')$: Per ipotesi, $K' \vdash h(M')$. Sia x una variabile in M ma non in K. Per la Proposizione 8.2.15, esiste un albero di prova di $K' \vdash h(M')$ in cui non sono utilizzate regole di eliminazione. L'unica regola costruttiva applicabile è (*HASH*). Quindi $K' \vdash M'$ e x è in M'. Inoltre, $\overline{\mathsf{Atoms}}(K[\rho]) \vdash x[\rho]$. Allora, per induzione, $\overline{\mathsf{Atoms}}(K[\rho]) \vdash M[\rho]$. □

Proposizione 8.2.40 *Per ogni traccia t e per ogni sostituzione ground ρ che rispetta t, la run $t[\rho]$ è una traccia.*

Dim. Sia t una traccia di lunghezza p e ρ una sostituzione ground che rispetta t. La dimostrazione è data per induzione sulla lunghezza di t.

Caso base : Se t ha lunghezza 0, $t = \epsilon$. Per definizione di traccia ground, t è una traccia ground.

Passo induttivo : Supponiamo che t sia una traccia di lunghezza $p > 0$. Per la Proposizione 8.1.3, i prefissi di t sono tracce. Sia t' il prefisso di t di lunghezza $p - 1$. Per definizione di rispettabilità, ρ rispetta t'. Per induzione, $t'[\rho]$ è una traccia. Dimostriamo che $t[\rho]$ è una traccia, distinguendo i casi sull'ultima azione di t.

case $t = t' \cdot !\langle M \rangle$: In questo caso non ci sono condizioni aggiuntive da soddisfare. Per definizione di traccia, $t[\rho]$ è una traccia.

case $t = t' \cdot ?\langle M \rangle$: In questo caso, per definizione di traccia, $t[\rho]$ è una traccia se il messaggio $M[\rho]$ è deducibile da $O(t')[\rho]$. Per definizione di traccia $O(t') \vdash M$ e, per definizione di rispettabilità, per ogni variabile x in M ma non in $O(t')$, $O(t')[\rho] \vdash x[\rho]$. Per il Lemma 8.2.39, $O(t')[\rho] \vdash M[\rho]$. Allora $t[\rho]$ è una traccia. □

L'implicazione inversa della proposizione appena dimostrata non vale e quindi non è vero che ogni sostituzione ground, che rende una traccia del modello simbolico in una del modello concreto, rispetta la sequenza di interazioni del modello simbolico come si vede nel seguente esempio.

Esempio 8.2.41 *Sia* $t = !\langle \{\!|a|\!\}_{k^+} \rangle \cdot ?\langle \{\!|x|\!\}_{k^+} \rangle$ *una traccia e* $\rho = [x \mapsto a]$ *una sostituzione ground. Allora* $t[\rho]$ *è una traccia ground, ma* ρ *non rispetta* t *in quanto* $\{\!|a|\!\}_{k^+}\}[\rho] \not\vdash x[\rho]$.

L'esempio successivo evidenzia che, se non limitiamo l'analisi a processi chiusi, il modello che otteniamo non è corretto in quanto esistono tracce del modello simbolico che non hanno una controparte nel modello concreto.

Esempio 8.2.42 *Notiamo che* $t = !\langle \{x\}_n \rangle \cdot ?\langle x \rangle$ *è una traccia in quanto il giudizio* $\{x\}_k \vdash x$ *è derivabile, ma non esiste alcuna sostituzione ground* ρ *tale che* $\{x\}_k[\rho] \vdash x[\rho]$ *è derivabile.*

Per provare la completezza del modello dobbiamo verificare che ogni traccia ground sia una concretizzazione del modello simbolico. Per ottenere un tale risultato utilizziamo la seguente proposizione il cui enunciato esprime come le sostituzioni ground che rendono derivabile un giudizio sono istanze dei vincoli calcolati attraverso la procedura Constraints.

Lemma 8.2.43 *Sia* M *un messaggio,* K *un insieme di messaggi e* ρ *una sostituzione ground tale che* $M[\rho]$ *è ground e* $K[\rho]$ *è ground. Se* $K[\rho]$ *è atomico e* $K[\rho] \vdash M[\rho]$*, allora esiste* $R \in \mathsf{Realise}(K,M)$ *tale che* ρ *è un'istanza di* R.

Dim. Sia M un messaggio, K un insieme di messaggi e ρ una sostituzione ground. La dimostrazione è data per induzione su un ordine lessicografico che tiene conto delle variabili in $\{M\} \cup K$ e della complessità di M. Per la Proposizione 8.2.15, esiste un albero di prova di $K[\rho] \vdash M[\rho]$ in cui non sono utilizzate regole di eliminazione.

Caso base : In questo caso $M \in \mathcal{V} \cup \mathcal{T} \cup \mathcal{N} \cup \mathcal{K}^+ \cup \mathcal{K}^-$. Per ipotesi $K[\rho] \vdash M[\rho]$. Se $M \in \mathcal{V} \cup \mathcal{T} \cup \mathcal{K}^+$, per definizione di Realise, $\mathsf{Realise}(K,M) = \{[\,]\}$. Se $M \in \mathcal{N} \cup \mathcal{K}^-$, per il Corollario 8.2.17, $M \in K[\rho]$. Per definizione di Realise, $\mathsf{Realise}(K,M) = \{[\,]\}$. In ogni caso ρ è un'istanza della sostituzione vuota.

Passo induttivo : Supponiamo che in $\{M\} \cup K$ occorrano p variabili e M abbia complessità $q > 0$. Distinguiamo i casi sugli operatori di alto livello.

- $M = (M_1,M_2)$: Per ipotesi $K[\rho] \vdash (M_1[\rho],M_2[\rho])$, allora $K[\rho] \vdash M_1[\rho]$ e $K[\rho] \vdash M_2[\rho]$. Per induzione esiste $R_1 \in \mathsf{Realise}(K,M_1)$ e una sostituzione ground ρ_1 tale che $\rho = R_1@\rho_1$. Allora $K[R_1][\rho_1] \vdash M_2[R_1][\rho_1]$. Per induzione, dato che $|Vars(M_2[R_1]) \cup IVars(K[R_1])| < |Vars(M) \cup IVars(K)|$, esiste la sostituzione $R_2 \in \mathsf{Realise}(K[R_1],M_2[R_1])$ e una sostituzione ground ρ_2 tale che $\rho_1 = R_2@\rho_2$. Per definizione di Realise, $R_1@R_2 \in \mathsf{Realise}(K,M)$. Allora $\rho = R_1@R_2@\rho_2$.
- $M = \{M'\}_n$: Per ipotesi $K[\rho] \vdash \{M'[\rho]\}_n$. Distinguiamo due casi:

1. $K[\rho] \vdash n$: In questo caso $K[\rho] \vdash M'[\rho]$. Per induzione, esiste una sostituzione $R \in \mathsf{Realise}(K,M')$ e una sostituzione ground ρ' tale che $\rho = R@\rho'$. Inoltre, se $K[\rho]$ è atomico, $n \in K[R]$. Allora $\mathsf{Realise}(K[R],n) = \{[\ ]\}$. Per definizione di $\mathsf{Realise}$, $R \in \mathsf{Realise}(K,M)$. Allora $\rho = R@\rho'$.
2. $K[\rho] \nvdash n$: Per il Lemma 8.2.18, $M[\rho] \in K[\rho]$. Quindi esiste $N \in K$ tale che $N = \{N'\}_n$ e $M'[\rho] = N'[\rho]$. Allora esiste $R \in \mathsf{Realise}(K,M)$ tale che $R = mgu\{M' = N'\}$. Per definizione di most general unifier, $\rho = R@\rho'$.

$M = \{\!|M'|\!\}_{k^+}$: Per ipotesi $K[\rho] \vdash \{\!|M'[\rho]|\!\}_{k^+}$. Distinguiamo due casi.

1. $K[\rho] \vdash M'[\rho]$: Per induzione esiste $R \in \mathsf{Realise}(K,M')$ tale che $\rho = R@\rho'$. Per definizione di $\mathsf{Realise}$, $R \in \mathsf{Realise}(K,M)$.
2. $K[\rho] \nvdash M'[\rho]$: Per il Lemma 8.2.19, $M[\rho] \in K[\rho]$. Quindi esiste $N \in K$ tale che $N = \{\!|N'|\!\}_{k^+}$ e $M'[\rho] = N'[\rho]$. Allora esiste $R \in \mathsf{Realise}(K,M)$ tale che $R = mgu\{M' = N'\}$. Per definizione di most general unifier, $\rho = R@\rho'$.

$M = h(M')$: Per ipotesi $K[\rho] \vdash h(M'[\rho])$. Distinguiamo due casi.

1. $K[\rho] \vdash M'[\rho]$: Per induzione esiste $R \in \mathsf{Realise}(K,M')$ tale che $\rho = R@\rho'$. Per definizione di $\mathsf{Realise}$, $R \in \mathsf{Realise}(K,M)$.
2. $K[\rho] \nvdash M'[\rho]$: Per il Lemma 8.2.20, $M[\rho] \in K[\rho]$. Quindi esiste $N \in K$ tale che $N = h(M')$ e $M'[\rho] = N'[\rho]$. Allora esiste $R \in \mathsf{Realise}(K,M)$ tale che $R = mgu\{M' = N'\}$. Per definizione di most general unifier, $\rho = R@\rho'$. □

Proposizione 8.2.44 *Sia M un messaggio, K un insieme di messaggi e ρ una sostituzione ground tale che $M[\rho]$ è ground e $K[\rho]$ è ground. Se $K[\rho]$ è atomico e $K[\rho] \vdash M[\rho]$, allora esiste $R \in \mathsf{Constraints}(K,M)$ tale che ρ è un'istanza di R.*

Dim. La proposizione si dimostra banalmente iterando il Lemma 8.2.43. □

Lemma 8.2.45 *Sia M un messaggio, K un insieme atomico di messaggi e ρ una sostituzione ground tale che $M[\rho]$ e $K[\rho]$ siano ground. Se $K[\rho]$ è atomico, i giudizi $K \vdash M$ e $K[\rho] \vdash M[\rho]$ sono derivabili e la sostituzione vuota è l'unica sostituzione calcolata da $\mathsf{Constraints}(K,M)$, allora, per ogni variabile x in M ma non in K, il giudizio $K[\rho] \vdash x[\rho]$ è derivabile.*

Dim. Sia M un messaggio, K un insieme atomico di messaggi e ρ una sostituzione ground. Diamo la dimostrazione per induzione sulla complessità di M.

Caso base : Se M ha complessità 0, allora $M \in \mathcal{V} \cup \mathcal{T} \cup \mathcal{N} \cup \mathcal{K}^+ \cup \mathcal{K}^-$. Se M è un nome, un token, una chiave pubblica o una chiave privata, il lemma è banalmente verificato perchè queste entità non contengono variabili. Se M è una variabile il risultato vale per ipotesi.

Passo induttivo : Supponiamo che M abbia complessità $p > 0$. Distinguiamo i casi sugli operatori di alto livello.

$M = (M_1,M_2)$: In questo caso $K \vdash M_i$ e $K[\rho] \vdash M_i[\rho]$ con $i = 1,2$. Sia x una variabile in M ma non in K. Allora x è in M_1 o in M_2. Per definizione di Constraints, se Constraints$(K,M) = \{[\,]\}$, Constraints$(K,M_1) = \{[\,]\}$ e Constraints$(K,M_2) = \{[\,]\}$. Per induzione $K[\rho] \vdash x[\rho]$.

$M = \{M'\}_n$: Sia x una variabile in M ma non in K. Per la Proposizione 8.2.15, esiste un albero di prova di $K \vdash \{M'\}_n$ in cui non sono utilizzate regole di eliminazione. L'unica regola costruttiva applicabile è la regola (*SKENC*). Quindi $K \vdash M'$ e $K \vdash n$. Dato che i nomi non contengono variabili, x è in M'. Per il Lemma 8.2.11 e il Corollario 8.2.17, $K[\rho] \vdash n$. Allora $K[\rho] \vdash M'[\rho]$. Per definizione di Constraints, Constraints$(K,M') = \{[\,]\}$. Per induzione $K[\rho] \vdash x[\rho]$.

$M = \{\!|M'|\!\}_{k^+}$: Sia x una variabile in M ma non in K. Per la Proposizione 8.2.15, esiste un albero di prova di $K \vdash \{\!|M'|\!\}_{k^+}$ in cui non sono utilizzate regole di eliminazione. L'unica regola costruttiva applicabile è la regola (*PKENC*). Quindi $K \vdash M'$. Dato che le chiavi pubbliche non contengono variabili, x è in M'. Poiché Constraints$(K,M) = \{[\,]\}$, non esiste in K un messaggio N tale che $N[\rho] = M[\rho]$. Allora l'unica regola costruttiva per derivare $K[\rho] \vdash \{\!|M'|\!\}_{k^+}[\rho]$ è (*PKENC*) e quindi $K[\rho] \vdash M'[\rho]$. Inoltre, per la definizione di Constraints, Constraints$(K,M') = \{[\,]\}$. Per induzione $K[\rho] \vdash x[\rho]$.

$M = h(M')$: Sia x una variabile in M ma non in K. Per la Proposizione 8.2.15, esiste un albero di prova di $K \vdash h(M')$ in cui non sono utilizzate regole di eliminazione. L'unica regola costruttiva applicabile è la regola (*HASH*). Quindi $K \vdash M'$ e x è in M'. Poiché Constraints$(K,M) = \{[\,]\}$, non esiste in K un messaggio N tale che $N[\rho] = M[\rho]$. Allora l'unica regola costruttiva per derivare $K[\rho] \vdash h(M')[\rho]$ è (*HASH*) e quindi $K[\rho] \vdash M'[\rho]$. Inoltre, per la definizione di Constraints, Constraints$(K,M') = \{[\,]\}$. Per induzione $K[\rho] \vdash x[\rho]$. □

8.2.4 Correttezza e Completezza

Per essere sicuri che il modello calcolato dalla procedura Model_{at} sia utilizzabile per la verifica delle proprietà di sicurezza, dimostriamo formalmente la correttezza e la completezza del modello simbolico. Per correttezza si intende che ogni traccia del modello simbolico ha una controparte nel modello concreto, mentre per completezza si intende che ogni esecuzione che trova posto nel modello concreto ha una controparte nel modello simbolico. In questo modo, se una traccia del modello simbolico viola le proprietà di sicurezza, allora il protocollo modellato non le soddisfa. Inoltre, le tracce "cattive" consentono di individuare i possibili attacchi.

Teorema 8.2.46 *Sia s una run. Per ogni $R \in \mathsf{Model}_{at}(\epsilon,s)$ la traccia $s[R]$ è rispettabile e, per ogni sostituzione ground ρ, che rispetta $s[R]$ abbiamo che la run ground $s[R][\rho]$ è una traccia ground.*

Dim. Per la Proposizione 8.2.26, $s[R]$ è una traccia con $R \in \mathsf{Model}_{at}(\epsilon,s)$. Per la Proposizione 8.2.38, per ogni traccia esiste una sostituzione ground ρ che la rispetta. Per la Proposizione 8.2.40, se $s[R]$ è una traccia ed esiste una sostituzione ground ρ che rispetta

$s[R]$, $s[R][\rho]$ è una traccia. □

Abbiamo provato come le tracce simboliche sono un'astrazione di qualche traccia ground, cioè che ogni traccia del modello simbolico ha una controparte in quello concreto. Allora è immediato verificare che il modello simbolico è corretto.

Corollario 8.2.47 ***(Correttezza)*** *Sia P un processo. Per ogni traccia $t \in \mathsf{Model}_{at}(P)$ esiste una sostituzione ground ρ tale che $t[\rho] \in \mathsf{Trace}(P)$.*

A questo punto, per provare che il modello simbolico possa essere utilizzato per una verifica automatica delle proprietà di sicurezza e che i risultati ottenuti valgano anche nel modello concreto, rimane da dimostrare la completezza del modello simbolico. A tale scopo, proviamo che le traccie ground sono una concretizzazione di qualche traccia del modello simbolico.

Teorema 8.2.48 *Sia t una traccia e s una run. Per ogni sostituzione ground ρ tale che ρ rispetta t e $(t \cdot s)[\rho]$ è una traccia ground, esiste una sostituzione $R \in \mathsf{Model}_{at}(t,s)$ e una sostituzione ground ρ' tale che $\rho = R@\rho'$, $(t \cdot s)[R]$ è una traccia e ρ' rispetta $(t \cdot s)[R]$.*

Dim. Sia s una run di lunghezza p con n variabili e ρ una sostituzione ground tale che $s[\rho]$ è una traccia ground. Diamo una dimostrazione per induzione sulla struttura di s, cioè un ordine lessicografico che tiene conto delle variabili che occorrono in s e della lunghezza di s, utilizzando la procedura in Figura 8.2.

Caso base : Se $s = \epsilon$, allora $\mathsf{Model}_{at}(\epsilon,s) = \{[\,]\}$. In questo caso $\rho = \rho'$ e quindi, per ipotesi, t è una traccia e ρ rispetta t.

Passo induttivo : Supponiamo che s sia una run di lunghezza $p > 0$ con n variabili. Per definizione di traccia ground, ogni prefisso di una traccia ground è una traccia ground. Sia $s'[\rho]$ il prefisso di $s[\rho]$ di lunghezza $p - 1$. Distinguiamo i casi sull'ultima azione di s.

- **case** $s = s' \cdot !\langle M \rangle$: Per ipotesi $(t \cdot s' \cdot !\langle M \rangle)[\rho]$ è una traccia ground e ρ rispetto t. Per definizione di traccia ground, $(t \cdot s')[\rho]$ è una traccia ground. Allora, per induzione, esiste $R \in \mathsf{Model}_{at}(t,s')$ e una sostituzione ground ρ' tale che $\rho = R@\rho'$, $(t \cdot s')[R]$ è una traccia e ρ' rispetta $(t \cdot s')[R]$. Per definizione di Model_{at}, $\mathsf{Model}_{at}(t,s') = \mathsf{Model}_{at}(t,s)$. Allora, per definizione di traccia, $(t \cdot s)[R]$ è una traccia e, per definizione di rispettabilità, ρ' rispetta $(t \cdot s)[R]$.
- **case** $s = s' \cdot ?\langle M \rangle$: Per ipotesi $(t \cdot s' \cdot ?\langle M \rangle)[\rho]$ è una traccia ground e ρ rispetto t. Per definizione di traccia ground, $(t \cdot s')[\rho]$ è una traccia ground. Allora, per induzione, esiste $R \in \mathsf{Model}_{at}(t,s')$ e una sostituzione ground ρ' tale che $\rho = R@\rho'$, $(t \cdot s')[R]$ è una traccia e ρ' rispetta $(t \cdot s')[R]$. Inoltre, dato che $(t \cdot s' \cdot ?\langle M \rangle)[\rho]$ è una traccia ground, $O(t \cdot s')[\rho] \vdash M[\rho]$ e quindi $O(t \cdot s')[R][\rho'] \vdash M[R][\rho']$. Per la Proposizione 8.2.14, $\overline{\mathsf{Atoms}}(O(t \cdot s')[R][\rho']) \vdash M[R][\rho']$. Per la Proposizione 8.2.36, $\overline{\mathsf{Atoms}}(O(t \cdot s')[R])[\rho'] \vdash M[R][\rho']$. Allora, per la Proposizione

8.2.44, esiste $R_0 \in \mathsf{Constraints}(\overline{\mathsf{Atoms}}(O(t \cdot s')[R]), M[R])$ e una sostituzione ground ρ'' tale che $\rho' = R_0@\rho''$. Distinguiamo due casi.

1. Esiste $R_0 \neq [\,]$: Per definizione di rispettabilità, ρ'' rispetta ϵ e, per ipotesi, $(t \cdot s)[R@R_0][\rho'']$ è una traccia ground. Allora, per induzione, dato che $|SVars(s[R@R_0])| \leq |SVars(s[R])|$, esiste $R_1 \in \mathsf{Model}_{at}(\epsilon, (t \cdot s)[R@R_0])$ e una sostituzione ground ρ''' tali che $\rho'' = R_1@\rho'''$, $(t \cdot s)[R@R_0@R_1]$ è una traccia e ρ''' rispetta $(t \cdot s)[R@R_0@R_1]$. Per definizione di Model_{at}, $R@R_0@R_1 \in \mathsf{Model}_{at}(t,s)$. Allora $\rho = R@R_0@R_1@\rho'''$, $(t \cdot s)[R@R_0@R_1]$ è una traccia e ρ''' rispetta $(t \cdot s)[R@R_0@R_1]$.
2. $R_0 = [\,]$ è l'unica soluzione: Per definizione di $\overline{\mathsf{Model}}_{at}$, $R \in \mathsf{Model}_{at}(t,s)$ e $\rho = R@\rho'$. Dato che $(t \cdot s')[R]$ è una traccia e $\overline{\mathsf{Atoms}}(O(t \cdot s')[R]) \vdash M[R]$, $(t \cdot s)[R]$ è una traccia. Inoltre, dato che $(t \cdot s)[\rho]$ è una traccia ground, $\overline{\mathsf{Atoms}}(O(t \cdot s')[\rho]) \vdash M[\rho]$. Per il Lemma 8.2.45, per ogni variabile x in $M[R]$ ma non in $O(t \cdot s')[R]$, $\overline{\mathsf{Atoms}}(O(t \cdot s')[R])[\rho'] \vdash x[\rho']$. Allora, per definizione di rispettabilità, ρ' rispetta $(t \cdot s)[R]$. □

Abbiamo provato così come ogni traccia ground è una concretizzazione di qualche traccia del modello simbolico, cioè che ogni traccia del modello concreto ha una controparte in quello simbolico. Allora è immediato verificare che il modello simbolico è completo.

Corollario 8.2.49 ***(Completezza)*** *Sia P un processo. Per ogni traccia ground $t \in \mathsf{Trace}(P)$ esiste una traccia $s \in \mathsf{Model}_{at}(P)$ e una sostituzione ground ρ tale che $t = s[\rho]$.*

8.3 Caso Generale: Messaggi Arbirari come Chiavi Condivise

Per analizzare protocolli nel "mondo reale" è spesso necessario estendere il modello in modo da considerare chiavi condivise composte. In un tipico scenario di key exchange, due agenti si scambiano un segreto, e ognuno deriva la chiave simmetrica condivisa codificando attraverso una funzione hash le parti di un segreto condiviso con nonce e altri dati. Un esempio di questo è dato dal calcolo della *master key* nel SSL 3.0 handshake protocol [38]. Estendiamo così l'analisi al caso in cui le chiavi utilizzate dagli algoritmi crittografici simmetrici siano messaggi arbitrari.
Questa sezione segue quanto fatto per il caso in cui le chiavi condivise sono ristrette a nomi, ripercorrendone fedelmente i passi e mettendo in luce le differenze e le similitudini tra i due casi. Osserviamo subito che, senza i vincoli imposti alle chiavi condivise, non è più necessaria un'istanziazione brute force delle variabili per dimostrare la completezza del metodo, rendendo così la dimensione del modello ottenuto più compatta. La grammatica utilizzata in questo caso è quella presentata in Definizione 6.1.2 in cui le chiavi condivise possono essere qualsiasi messaggio.

8.3.1 Atomi

Introduciamo il concetto di atomo anche nel caso generale per cercare di sfruttare i risultati ottenuti in precedenza. Gli atomi sono introdotti per semplificare l'insieme dei messaggi

che costituiscono la conoscenza dell'ambiente senza, però, perdere informazioni. Intuitivamente, gli atomi sono gli elementi irriducibili di un insieme ottenuti scomponendo i messaggi in accordo con le regole di inferenza del sistema deduttivo, ed eliminando i messaggi che appartengono alla conoscenza iniziale dell'ambiente. La nuova definizione si differenzia da quella data per il caso particolare, per la regola utilizzata per scomporre i messaggi codificati a chiave condivisa. Prima si sfruttava il fatto che un nome è derivabile dall'insieme degli atomi se e solo se appartiene all'insieme stesso, ora si considerano messaggi arbitrari, per cui le chiave condivise potrebbero essere deducibili dall'insieme degli atomi senza appartenere all'insieme stesso. Analogamente a quanto fatto nel caso in cui è posto un limite alle chiavi condivise, definiamo le funzione Atoms e $\overline{\mathsf{Atoms}}$, necessarie per calcolare un insieme atomico da un qualsiasi insieme di messaggi.

Definizione 8.3.1 *La funzione* Atoms *è una funzione da insiemi di messaggi a insiemi di messaggi. Diamo di seguito la defiizione formale.*

$$
\begin{array}{lllll}
(1) & \mathsf{Atoms}(\{M\} \cup K) & = & \mathsf{Atoms}(K) & \textit{se } M \in \mathcal{V} \cup \mathcal{T} \cup \mathcal{K}^{+} \\
(2) & \mathsf{Atoms}(\{(M,N)\} \cup K) & = & \mathsf{Atoms}(\{M,N\} \cup K) & \\
(3) & \mathsf{Atoms}(\{\{M\}_N\} \cup K) & = & \mathsf{Atoms}(\{M\} \cup K) \textit{ se } K \vdash N & \\
(4) & \mathsf{Atoms}(\{[\![M]\!]_{k^+}, k^-\} \cup K) & = & \mathsf{Atoms}(\{M, k^-\} \cup K) & \\
(5) & \mathsf{Atoms}(\{[\![M]\!]_{k^+}\} \cup K) & = & \mathsf{Atoms}(K) \textit{ se } K \vdash M & \\
(6) & \mathsf{Atoms}(\{h(M)\} \cup K) & = & \mathsf{Atoms}(K) \textit{ se } K \vdash M & \\
(7) & \mathsf{Atoms}(K) & = & K \quad \textit{altrimenti} &
\end{array}
$$

Analizziamo ora le regole appena introdotte. La prima afferma che le variabili, i token e le chiavi pubbliche non appartengono all'insieme degli atomi. Queste entità, infatti, sono derivabili da qualsiasi insieme di messaggi e quindi non è indispensabile memorizzarle. Le successive tre regole sono equivalenti, rispettivamente, all'applicazione delle regole ($PROJ_i$), ($SKDEC$) e ($PKDEC$) di Tabella 6.1. In particolare, la regola (3) esprime come l'ambiente può decriptare un messaggio solo se la chiave utilizzata per la codifica è deducibile dalla sua conoscenza. Le regole (5) e (6) affermano che l'ambiente può cercare di decriptare un messaggio codificato attraverso un algoritmo crittografico asimmetrico o una funzione hash provando a cifrare tutti i messaggi da lui conosciuti. L'ultima regola viene applicata quando non è possibile fare alcun matching e restituisce l'insieme degli atomi.

Definizione 8.3.2 *Sia K un insieme di messaggi. L'insieme atomico di messaggi associato a K è calcolato nel seguente modo:*

$$
\overline{\mathsf{Atoms}}(K) = \begin{cases} K & \textit{se } \mathsf{Atoms}(K) = K \\ \overline{\mathsf{Atoms}}(\mathsf{Atoms}(K)) & \textit{altrimenti} \end{cases}
$$

Come detto per il caso precedente, la notazione adottata si è resa necessaria per non confondere l'insieme atomico risultante con gli insiemi utilizzati per calcolare tale insieme.
Verifichiamo ora alcune proprietà della funzione $\overline{\mathsf{Atoms}}$ utilizzate nel seguito del capitolo per dimostrare gli obiettivi della nostra tesi. È nostra intenzione mostrare come la funzione $\overline{\mathsf{Atoms}}$, dato in input un insieme di messaggi K, restituisce un insieme di messaggi

contenente la stessa informazione dell'insieme di partenza. Osserviamo, inoltre, come l'insieme atomico $\overline{\mathsf{Atoms}}(K)$ è minimale, cioè come ogni messaggio deducibile o appartiene all'insieme o può essere ottenuto per composizione dai messaggi dell'insieme.
Analizzando le regole della Definizione 8.3.1, evidenziamo alcune proprietà che caratterizzano le sequenze utilizzate per calcolare gli insiemi di atomi. Queste proprietà sono in parte utili per dimostrare alcuni risultati, già visti per il caso in cui le chiavi condivise sono ristrette a nomi, e in parte hanno lo scopo di fornire un'idea intuitiva più completa del funzionamento della funzione $\overline{\mathsf{Atoms}}$.

Osservazione 8.3.3 *Possiamo vedere l'insieme delle regole date nella Definizione 8.3.1, come un sistema di riscrittura da sinistra a destra dove le condizioni delle regole sono date dal matching con gli operandi. Pertanto, se* $\overline{\mathsf{Atoms}}(K) = K'$*, allora esiste una sequenza finita di espressioni* E_0,...,E_n *dove ogni espressione* E_i *è della forma* $\mathsf{Atoms}(K_i)$ *per qualche* K_i*, e ogni espressione* E_{i+1} *è ricavata da* E_i *applicando le regole del sistema di riscrittura. In particolare, abbiamo che* $E_0 = \mathsf{Atoms}(K)$ *e* $E_n = \mathsf{Atoms}(K')$*.*

Anche quando le chiavi condivise sono qualsiasi messaggio è conveniente utilizzare le nozioni di sequenza in forma normale e di prefisso normale presentate, rispettivamente, in Definizione 8.2.6 e in Definizione 8.2.7. Come fatto per il caso particolare, dimostriamo che ciò non comporta vincoli aggiuntivi all'analisi. L'idea è costruire una sequenza di espressioni in forma normale ritardando l'applicazione della regola (1) dell'Osservazione 8.3.3 nel caso in cui il matching sia fatto con una variabile e ciò è possibile in quanto tale regola non introduce nuovi messaggi negli insiemi associati alle espressioni ottenute. La dimostrazione del seguente lemma risulta analoga a quella data per il caso particolare e per questo viene omessa.

Lemma 8.3.4 *Sia K un insieme di messaggi e* $K' = \overline{\mathsf{Atoms}}(K)$*. Esiste una sequenza* F_0,...,F_n *in forma normale in cui* $F_0 = \mathsf{Atoms}(K)$ *e* $F_n = \mathsf{Atoms}(K')$*.*

Dim. Si veda il Lemma 8.2.8. □

La prossima proposizione mostra alcuni risultati illustrati nel seguito di questa tesi e il suo enunciato esprime come, dato un insieme K, l'insieme $\overline{\mathsf{Atoms}}(K)$ contiene la stessa informazione di K. Per provare questo risultato, sfrutteremo il fatto che i messaggi appartenenti ad $\overline{\mathsf{Atoms}}(K)$ sono deducibili da K e che gli elementi di K sono deducibili da $\overline{\mathsf{Atoms}}(K)$. Questi risultati intermedi sono ottenuti studiando i passi eseguiti dal sistema di riscrittura per calcolare l'insieme degli atomi.

Lemma 8.3.5 *Sia* $E = \mathsf{Atoms}(K_i)$ *e* $F = \mathsf{Atoms}(K_j)$*. Se F è ottenuta da E applicando una regola del sistema di riscrittura dell'Osservazione 8.3.3, allora* $K_i \vdash K_j$ *e* $K_j \vdash K_i$*.*

Dim. La dimostrazione è data per casi sulla regola del sistema di riscrittura applicata per passare da K_i a K_j.

Regola (1) : In questo caso $K_i = K_j \cup \{M\}$ con $M \in \mathcal{V} \cup \mathcal{T} \cup \mathcal{K}^+$. Allora $K_i \vdash K_j$ perché $K_j \subset K_i$ e $K_j \vdash K_i$ perché le variabili, i token e le chiavi pubbliche sono derivabili da qualsiasi insieme di messaggi.

Regola (2) : In questo caso $K_i = K_j \setminus \{M,N\} \cup \{(M,N)\}$. Abbiamo che $K_i \vdash M$ e $K_i \vdash N$, e che $K_j \vdash (M,N)$, allora $K_i \vdash K_j$ e $K_j \vdash K_i$.

Regola (3) : In questo caso $K_i = K_j \setminus \{M\} \cup \{\{M\}_N\}$. Abbiamo che $K_i \vdash M$ e $K_i \vdash N$ e che $K_j \vdash \{M\}_N$, allora $K_i \vdash K_j$ e $K_j \vdash K_i$.

Regola (4) : In questo caso $K_i = K_j \setminus \{M,k^-\} \cup \{\{\!|M|\!\}_{k^+},k^-\}$. Abbiamo che $K_i \vdash M$ e che $K_j \vdash \{\!|M|\!\}_{k^+}$, allora $K_i \vdash K_j$ e $K_j \vdash K_i$.

Regola (5) : In questo caso $K_i = K_j \cup \{\{\!|M|\!\}_{k^+}\}$ con $K_j \vdash M$. Allora $K_i \vdash K_j$ perché $K_j \subset K_i$ e che $K_j \vdash K_i$ perché $K_j \vdash \{\!|M|\!\}_{k^+}$.

Regola (6) : In questo caso $K_i = K_j \cup \{h(M)\}$ con $K_j \vdash M$. Allora $K_i \vdash K_j$ perché $K_j \subset K_i$ e che $K_j \vdash K_i$ perché $K_j \vdash h(M)$. □

Il risultato appena raggiunto mostra come i passi del sistema di riscrittura preservano il contenuto informativo degli insiemi a loro associati e, estendendolo, si prova che $K \vdash \overline{\mathsf{Atoms}}(K)$ e $\overline{\mathsf{Atoms}}(K) \vdash K$.

Lemma 8.3.6 *Sia K un insieme di messaggi. Allora* $K \vdash \overline{\mathsf{Atoms}}(K)$ *e* $\overline{\mathsf{Atoms}}(K) \vdash K$.

Dim. Sia K un insieme di messaggi e $K' = \overline{\mathsf{Atoms}}(K)$. Consideriamo una sequenza di espressioni $E_0,...,E_n$ della forma $\mathsf{Atoms}(K_i)$ per qualche K_i ottenuta applicando le regole del sistema di riscrittura dell'Osservazione 8.3.3 dove $E_0 = \mathsf{Atoms}(K)$ e $E_n = \mathsf{Atoms}(K')$. Il risultato si ottiene dal Lemma 8.3.5 per induzione sulla lunghezza della sequenza ottenuta applicando le regole del sistema di riscrittura. □

Abbiamo dimostrato che dall'insieme di messaggi K si deducono tutti i messaggi che appartengono a $\overline{\mathsf{Atoms}}(K)$ e che da $\overline{\mathsf{Atoms}}(K)$ si deducono tutti quelli di K. Questo risultato è utile per provare che K e $\overline{\mathsf{Atoms}}(K)$ hanno lo stesso contenuto informativo.

Proposizione 8.3.7 *Sia M un messaggio e K un insieme di messaggi. Allora il giudizio* $K \vdash M$ *è derivabile sse* $\overline{\mathsf{Atoms}}(K) \vdash M$ *è derivabile.*

Dim. Deriva direttamente dal Lemma 8.3.6 e dalla Proposizione 6.2.9. □

Quindi, l'insieme $\overline{\mathsf{Atoms}}(K)$, analogamente al caso in cui le chiavi condivise sono ristrette a nomi, contiene la stessa informazione dell'insieme K. Infatti, dai due insiemi è possibile dedurre gli stessi messaggi, e questo consente di semplificare la struttura delle derivazioni in quanto le regole di eliminazione sono implicite nella funzione Atoms come si vede nella proposizione seguente.

Proposizione 8.3.8 *Sia M un messaggio e K un insieme atomico di messaggi. Se $K \vdash M$, allora esiste un albero di prova che non utilizza regole di eliminazione.*

Dim. Sia M un messaggio, K un insieme atomico di messaggi e Γ un albero di prova di profondità p del giudizio $K \vdash M$. Dimostriamo il risultato della proposizione per induzione sulla profondità dell'albero di prova.

Caso base : Se Γ ha profondità 1, allora l'albero è ottenuto applicando un assioma del sistema deduttivo.

Passo induttivo : Supponiamo che Γ abbia profondità $p > 1$. Allora Γ è costituito da sottoalberi di profondità minore. Per induzione, questi ultimi possono essere sostituiti da alberi costruiti utilizzando solo regole costruttive, cioè assiomi e regole di introduzione. Dimostriamo che esiste un albero di prova del giudizio $K \vdash M$ in cui non vengono utilizzate regole di eliminazione, distinguendo i casi sull'ultima regola applicata in Γ. Se l'ultima regola applicata è una regola di introduzione il risultato vale. Se l'ultima regola applicata è una regola di eliminazione, distinguiamo i seguenti casi.

regola ($PROJ_i$) : Consideriamo solo il caso della regola ($PROJ_1$), in quanto il caso della regola ($PROJ_2$) è analogo. Sia Γ l'albero di prova del giudizio $K \vdash M$ e l'ultima regola applicata sia la regola ($PROJ_1$). Per ipotesi induttiva, nella derivazione del giudizio $K \vdash (M,N)$ non sono applicate regole di eliminazione, e, per definizione di atomo, (M,N) non appartiene a K.

$$(PROJ_1)\ \dfrac{(PAIR)\ \dfrac{\dfrac{\Gamma_1}{K \vdash M} \quad \dfrac{\Gamma_2}{K \vdash N}}{K \vdash (M,N)}}{K \vdash M}$$

Allora l'albero cercato, costruito solo con regole costruttive, è

$$\dfrac{\Gamma_1}{K \vdash M}$$

regola ($SKDEC$) : Sia Γ l'albero di prova del giudizio $K \vdash M$ e l'ultima regola applicata sia la regola ($SKDEC$). Per ipotesi induttiva, nella derivazione dei giudizi $K \vdash \{M\}_N$ e $K \vdash N$ non sono applicate regole di eliminazione. Per definizione di atomo, $\{M\}_N$ non appartiene a K.

$$(SKDEC)\ \dfrac{(SKENC)\ \dfrac{\dfrac{\Gamma_1}{K \vdash M} \quad \dfrac{\Gamma_2}{K \vdash N}}{K \vdash \{M\}_N} \quad \dfrac{\Gamma_2}{K \vdash N}}{K \vdash M}$$

Allora l'albero cercato, costruito solo con regole costruttive, è

$$\dfrac{\Gamma_1}{K \vdash M}$$

regola (*PKDEC*) : Sia Γ l'albero di prova del giudizio $K \vdash M$ e l'ultima regola applicata sia la regola (*PKDEC*). Per ipotesi induttiva, nella derivazione dei giudizi $K \vdash \{\!|M|\!\}_{k^+}$ e $K \vdash k^-$ non sono applicate regole di eliminazioni. Allora l'unica regola costruttiva per derivare $K \vdash k^-$ è la regola (*AX*) e quindi $k^- \in K$. Per definizione di atomo, $\{\!|M|\!\}_{k^+}$ non appartiene a K.

$$\begin{array}{l}(PKENC)\\(PKDEC)\end{array} \dfrac{\dfrac{\Gamma_1}{K \vdash M} \qquad \dfrac{\Gamma_2}{K \vdash k^-}}{K \vdash M}$$

Allora l'albero cercato, costruito solo con regole costruttive, è

$$\frac{\Gamma_1}{K \vdash M}$$

□

A questo punto è interessante notare la relazione tra le derivazioni semplici, introdotte nel capitolo precedente, e il risultato appena raggiunto.

Proposizione 8.3.9 *Sia M un messaggio e K un insieme atomico di messaggi. Allora una derivazione di $K \vdash M$ è semplice se e solo se non utilizza regole di eliminazione.*

I risultati appena raggiunti consentono di stabilire la relazione tra i messaggi deducibili da un insieme di atomi e i messaggi che appartengono all'insieme stesso. In particolare, mostriamo come un nome, o una chiave privata, appartiene ad un insieme di atomi se e solo se esiste una derivazione dell'entità dall'insieme stesso.

Corollario 8.3.10 *Sia ζ un nome o una chiave privata e K un insieme atomico di messaggi. Allora $K \vdash \zeta$ sse $\zeta \in K$.*

Dim. Sia ζ un nome o una chiave privata e K un insieme di messaggi. Per la Proposizione 8.3.8, se $K \vdash \zeta$, esiste un albero di prova che non usa regole di eliminazione. Allora l'unica regola costruttiva per derivare $K \vdash \zeta$ è la regola (*AX*) e quindi $\zeta \in K$. Se $\zeta \in K$, allora $K \vdash \zeta$ utilizzando la regola (*AX*). □

La prossima proposizione mostra che l'insieme degli atomi è minimale e questo risultato è ottenuto stabilendo le condizioni di appartenenza di un messaggio ad un insieme atomico. I prossimi tre lemmi studiano l'appartenenza all'insieme $\overline{\mathsf{Atoms}}(K)$ di messaggi ottenuti attraverso operatori di ordine superiore. Dimostriamo solo il primo, che studia il caso dei messaggi ottenuti mediante un algoritmo crittografico simmetrico, in quanto gli altri sono analoghi a quelli presentati nel caso precedente.

Lemma 8.3.11 *Siano M e N messaggi, K un insieme di messaggi e $K \vdash \{M\}_N$. Allora $\{M\}_N \in \overline{\mathsf{Atoms}}(K)$ sse $K \nvdash N$.*

Dim. ($\Rightarrow$) Sia $\{M\}_n \in \overline{\mathsf{Atoms}}(K)$. Supponiamo per assurdo che $K \vdash N$. Per definizione di atomo, $\{M\}_N \notin \overline{\mathsf{Atoms}}(K)$. Allora $K \nvdash N$.

($\Leftarrow$) Se $K \vdash \{M\}_N$ e $K \nvdash N$, per la Proposizione 8.3.7, $\overline{\mathsf{Atoms}}(K) \vdash \{M\}_N$ e $\overline{\mathsf{Atoms}}(K) \nvdash N$. Per la Proposizione 8.3.8, esiste un albero di prova di $\overline{\mathsf{Atoms}}(K) \vdash \{M\}_N$ che utilizza solo regole di introduzione o assiomi. L'unica regola costruttiva applicabile è la regola (AX). Allora $\{M\}_N \in \overline{\mathsf{Atoms}}(K)$. □

Lemma 8.3.12 *Sia k^+ una chiave pubblica, M un messaggio, K un insieme di messaggi e $K \vdash \{\!|M|\!\}_{k^+}$. Allora $\{\!|M|\!\}_{k^+} \in \overline{Atoms}(K)$ sse $K \nvdash M$.*

Dim. Si veda il Lemma 8.2.19. □

Lemma 8.3.13 *Sia M un messaggio e K un insieme di messaggi e $K \vdash h(M)$. Allora $h(M) \in \overline{Atoms}(K)$ sse $K \nvdash M$.*

Dim. Si veda il Lemma 8.2.20. □

Mostriamo ora che la funzione $\overline{\mathsf{Atoms}}$ restituisce un insieme di messaggi irriducibili, e quindi che la funzione $\overline{\mathsf{Atoms}}$ è idempotente come mostra il seguente lemma. Questo risultato risulta fondamentale per provare come l'insieme ottento mediante la funzione $\overline{\mathsf{Atoms}}$ è minimale.

Lemma 8.3.14 *La funzione $\overline{Atoms}$ è idempotente.*

Dim. Si veda il Lemma 8.2.21. □

Dimostriamo che l'insieme, ottenuto applicando la funzione $\overline{\mathsf{Atoms}}$ ad un insieme di messaggi, è minimale, cioè ogni messaggio deducibile o appartiene o è ottenuto per composizione dai messaggi dell'insieme degli atomi. Per provare questo risultato, utilizziamo il seguente lemma il quale afferma che calcolare $\overline{\mathsf{Atoms}}(K \cup \{M\})$ equivale a calcolare $\overline{\mathsf{Atoms}}(K)$ e poi applicare nuovamente la funzione $\overline{\mathsf{Atoms}}$ all'insieme ottenuto unito con il messaggio M.

Lemma 8.3.15 *Sia M un messaggio e K un insieme di messaggi. Allora $\overline{Atoms}(K \cup \{M\}) = \overline{Atoms}(\overline{Atoms}(K) \cup \{M\})$.*

Dim. Si veda il Lemma 8.2.22. □

Intuitivamente, questo risultato mostra come non sia importante l'ordine con cui la funzione $\overline{\mathsf{Atoms}}$ considera i messaggi dell'insieme, e risulta fondamentale per dimostrare che un insieme di atomi è minimale.

Proposizione 8.3.16 *Sia M un messaggio e K un insieme di messaggi. Per $K \vdash M$ derivabile, $\overline{Atoms}(K \cup \{M\}) = \overline{Atoms}(K)$.*

Dim. Sia M un messaggio e K un insieme di messaggi. Diamo una dimostrazione per induzione sulla complessità di M.

Caso base : Se M ha complessità 0, allora $M \in \mathcal{V} \cup \mathcal{N} \cup \mathcal{T} \cup \mathcal{K}^- \cup \mathcal{K}^+$. Se M è una variabile, un token o una chiave pubblica, allora $\overline{\mathsf{Atoms}}(K \cup \{M\}) = \overline{\mathsf{Atoms}}(K)$, per la Definizione 8.3.1 punto (1). Se M è un nome o una chiave privata, per la Proposizione 8.3.7 e il Corollario 8.3.10, $M \in \overline{\mathsf{Atoms}}(K)$. Per il Lemma 8.3.15, $\overline{\mathsf{Atoms}}(K \cup \{M\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{M\})$. Allora $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{M\})$ è uguale a $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K))$. Per il Lemma 8.3.14, $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K)) = \overline{\mathsf{Atoms}}(K)$.

Passo induttivo : Supponiamo che M abbia complessità $p > 0$. Distinguiamo i casi sugli operatori di alto livello.

case $M = (M_1, M_2)$: Per la Definizione 8.3.1 punto (2), $\overline{\mathsf{Atoms}}(K \cup \{(M_1, M_2)\})$ è uguale a $\overline{\mathsf{Atoms}}(K \cup \{M_1\} \cup \{M_2\})$. Per il Lemma 8.3.15, quest'ultimo equivale a $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K \cup \{M_1\}) \cup \{M_2\})$. Per la regola (*PAIR*), $K \vdash M_1$ e $K \vdash M_2$. Per induzione, dato che $K \vdash M_1$, $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K \cup \{M_1\}) \cup \{M_2\}) = \overline{\mathsf{Atoms}}(K \cup \{M_2\})$. Per induzione, dato che $K \vdash M_2$, $\overline{\mathsf{Atoms}}(K \cup \{M_2\}) = \overline{\mathsf{Atoms}}(K)$.

case $M = \{M'\}_N$: Sia $K \vdash \{M'\}_N$. Distinguiamo due casi.

1. $K \nvdash N$: In questo caso, per il Lemma 8.3.11, $\{M'\}_N \in \overline{\mathsf{Atoms}}(K)$. Per il Lemma 8.3.15, $\overline{\mathsf{Atoms}}(K \cup \{\{M'\}_N\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{\{M'\}_N\})$. Allora $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{\{M'\}_N\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K))$. Per il Lemma 8.3.14, quest'ultimo è uguale a $\overline{\mathsf{Atoms}}(K)$.
2. $K \vdash N$: Per la Definizione 8.3.1 punto (3), $\overline{\mathsf{Atoms}}(K \cup \{\{M'\}_N\})$ è uguale a $\overline{\mathsf{Atoms}}(K \cup \{M'\})$. Per la regola (*SKDEC*), $K \vdash M'$. Allora, per induzione, $\overline{\mathsf{Atoms}}(K \cup \{M'\}) = \overline{\mathsf{Atoms}}(K)$.

case $M = \{\!|M'|\!\}_{k^+}$: Sia $K \vdash \{\!|M'|\!\}_{k^+}$. Distinguiamo due casi.

1. $K \vdash M'$: In questo caso $\overline{\mathsf{Atoms}}(K \cup \{\{\!|M'|\!\}_{k^+}\}) = \overline{\mathsf{Atoms}}(K)$, per la Definizione 8.3.1 punto (5).
2. $K \nvdash M'$: Per il Lemma 8.3.12, il messaggio $\{\!|M'|\!\}_{k^+} \in \overline{\mathsf{Atoms}}(K)$. Per il Lemma 8.3.15, $\overline{\mathsf{Atoms}}(K \cup \{\{\!|M'|\!\}_{k^+}\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{\{\!|M'|\!\}_{k^+}\})$. Allora $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{\{\!|M'|\!\}_{k^+}\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K))$. Per il Lemma 8.3.14, quest'ultimo è uguale a $\overline{\mathsf{Atoms}}(K)$.

case $M = h(M')$: Sia $K \vdash h(M')$. Distinguiamo due casi.

1. $K \vdash M'$: In questo caso $\overline{\mathsf{Atoms}}(K \cup \{h(M')\}) = \overline{\mathsf{Atoms}}(K)$, per la Definizione 8.3.1 punto (6).
2. $K \nvdash M'$: Per il Lemma 8.3.13, il messaggio $h(M') \in \overline{\mathsf{Atoms}}(K)$. Per il Lemma 8.3.15, $\overline{\mathsf{Atoms}}(K \cup \{h(M')\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{h(M')\})$. Allora $\overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K) \cup \{h(M')\}) = \overline{\mathsf{Atoms}}(\overline{\mathsf{Atoms}}(K))$. Per il Lemma 8.3.14, quest'ultimo è uguale a $\overline{\mathsf{Atoms}}(K)$. □

8.3.2 Modello Simbolico

Come già evidenziato, per la verifica automatica delle proprietà di sicurezza abbiamo bisogno di introdurre un modello che risponda a precisi requisiti: sia finito e sia in grado di rappresentare esattamente le informazioni contenute nel modello concreto.

Introduciamo una procedura in grado di calcolare le sostituzioni per cui una run possa diventare una traccia. A tal fine unifichiamo i messaggi che occorrono nelle azioni di input con i messaggi che appartengono alla conoscenza dell'ambiente. Iterando questa procedura si controlla se esiste una sostituzione per cui una run simbolica sia un'astrazione di qualche traccia ground. Questa procedura si distingue, da quella data per il caso delle chiavi condivise limitate a nomi, per la funzione che calcola l'insieme di atomi. Dimostriamo che la nuova procedura termina per ogni run, garantendo così che il modello sia finito, e che restituisce effettivamente le sostituzioni per cui una run sia una traccia.
Il modello simbolico è costruito dal grafo simbolico delle transizioni, determinando quali cammini del grafo simbolico hanno una controparte nel modello concreto. L'insieme delle tracce, ottenuto analizzando ogni run del grafo simbolico, costituisce il modello simbolico.

Definizione 8.3.17 *Il* modello di una run s, $\mathsf{Model}(\epsilon,s)$, *è l'insieme delle sostituzioni calcolate attraverso la procedura in Figura 8.3.*

```
Model(t,s) =
    case s of
      ϵ            ⇒ {[ ]}
      !⟨M⟩ · s′ ⇒ Model(t·!⟨M⟩,s′)
      ?⟨M⟩ · s′ ⇒ for R ∈ Constraints(Atoms(O(t)),M)
                        union
                              if R = [ ]
                              then Model(t·?⟨M⟩,s′)
                              else
                                for R′ ∈ Model(ϵ,(t · s)[R])
                                union { R@R′ }
```

Figura 8.3. Procedura Model per i modelli simbolici

Intuitivamente, questa procedura stabilisce che una traccia concatenata con una run vuota è ancora una traccia senza dover introdurre vincoli. Se invece vogliamo calcolare le sostituzioni Q tali per cui una traccia t concatenata con una run non vuota s sia ancora una traccia, dovremo tener conto della prima azione della sequenza s. Se la sequenza s è della forma $!\langle M\rangle \cdot s'$, cioè la prima azione di s è un'azione di output, la soluzione è data dalle sostituzioni Q, calcolate a partire da s', che rendono $(t \cdot s)$ una traccia. Se la sequenza s è della forma $?\langle M\rangle \cdot s'$, cioè la prima azione di s è un'azione di input, eseguiamo la procedura Constraints, cioè calcoliamo le sostituzioni R che rendono il messaggio M derivabile dall'insieme $\overline{\mathsf{Atoms}}(O(t))$. In base al risultato di questa procedura, distinguiamo due casi. Se la soluzione calcolata è la sostituzione vuota, il risultato di Model è dato dalle sostituzioni Q, calcolate a partire da s', che rendono $(t \cdot s)$ una traccia. Se la soluzione di Constraints è diversa dalla sostituzione vuota, il risultato di Model è dato dalle sostituzioni $R@R'$ tali che $(t \cdot s)[R][R']$ è una traccia dove $R' \in \mathsf{Model}(\epsilon,(t \cdot s)[R])$. In A.3.1 è presentato il codice ML di tale procedura.
Diamo in Figura 8.4 una procedura alternativa per calcolare i modelli simbolici di una run, che consente di semplificare la dimostrazione della completezza del modello.

```
Model(t,s) =
    case s of
     ϵ            ⇒ {[ ]}
     s'·!⟨M⟩      ⇒ Model(t,s')
     s'·?⟨M⟩      ⇒ for R' ∈ Model(t,s')
                       union for R0 ∈ Constraints(Atoms(O(t · s')[R']),M[R'])
                             union if R0 = [ ]
                                   then {R'}
                                   else for R1 ∈ Model(ϵ,(t · s)[R' @ R0])
                                        union { R' @ R0 @ R1 }
```

Figura 8.4. Procedura Model per i modelli simbolici

Le prossime due proposizioni sono indispensabili per dimostrare che il modello simbolico è finito e quindi utilizzabile per una verifica automatica delle proprietà di sicurezza dei protocolli. I loro enunciati esprimono, rispettivamente, che la procedura Model termina e calcola le sostituzioni per cui una run è una traccia del modello simbolico.

Proposizione 8.3.18 *Per ogni traccia t e run s, il programma* Model(t,s) *termina.*

Dim. Sia t una traccia qualsiasi e s una run di lunghezza p con n variabili. Dimostriamo la terminazione del programma Model(t,s) di Figura 8.3 per induzione sulla struttura di s, cioè un ordine lessicografico che tiene conto del numero delle variabili in s e della lunghezza di s.

Caso base : Se s ha lunghezza 0, allora $s = \epsilon$. In questo caso la procedura Model(t,ϵ) termina.

Passo induttivo : Supponiamo che s sia una run di lunghezza $p > 0$ con n variabili. Distinguiamo i casi sulla prima azione di s.

case $s = !\langle M\rangle \cdot s'$: Per definizione di traccia, $t\cdot!\langle M\rangle$ è una traccia. Per induzione, dato che $|SVars(s)| = |SVars(s')|$ e la lunghezza di s' è minore di quella di s, Model$(t\cdot!\langle M\rangle,s')$ termina. Per definizione di Model, Model$(t\cdot!\langle M\rangle,s')$ = Model$(t,!\langle M\rangle \cdot s')$. Allora Model$(t,!\langle M\rangle \cdot s')$ termina.

case $s = ?\langle M\rangle \cdot s'$: In questo caso Model$(t,s)$ richiama Constraints$(\overline{\text{Atoms}}(O(t)),M)$. Per il Teorema 7.2.10, la procedura Constraints termina e restituisce le sostituzioni R tali che Atoms$(O(t))[R] \vdash M[R]$. Distinguiamo i casi sul risultato della procedura Constraints:

1. $R = [\ ]$: Per definizione di traccia, $t\cdot?\langle M\rangle$ è una traccia. Per induzione, dato che $|SVars(s)| = |SVars(s')|$ e la lunghezza di s' è minore di quella di s, Model$(t\cdot?\langle M\rangle,s')$ termina. Per definizione di Model, Model$(t\cdot?\langle M\rangle,s')$ = Model$(t,?\langle M\rangle \cdot s')$. Allora Model$(t,?\langle M\rangle \cdot s')$ termina.
2. $R \neq [\ ]$: Per definizione di traccia, $(t\cdot?\langle M\rangle)[R]$ è una traccia. Per induzione, dato che $|SVars((t \cdot s)[R])| < |SVars(t \cdot s)|$, Model$(\epsilon,(t \cdot s)[R])$ termina. Per definizione di Model, Model$(\epsilon,(t\cdot s)[R])$ è uguale a Model$(t,?\langle M\rangle\cdot s')$. Allora Model$(t,?\langle M\rangle \cdot s')$ termina. □

Proposizione 8.3.19 *Per ogni traccia t, run s e per ogni $R \in \mathsf{Model}(t,s)$, la run $(t \cdot s)[R]$ è una traccia.*

Dim. Sia t una traccia qualsiasi e s una run di lunghezza p, con n variabili. Diamo una dimostrazione per induzione sulla struttura di s, cioè un ordine lessicografico che tiene conto del numero delle variabili contenute in s e della lunghezza della run.

Caso base : Se s ha lunghezza 0, allora $s = \epsilon$. In questo caso $\mathsf{Model}(t,\epsilon) = [\,]$ e $t = t \cdot \epsilon$. Allora $t \cdot \epsilon$ è una traccia.

Passo induttivo : Supponiamo che s sia una run di lunghezza $p > 0$ con n variabili. Distinguiamo i casi sulla prima azione di s.

case $s = !\langle M\rangle \cdot s'$: Per definizione di Model, $R \in \mathsf{Model}(t\cdot!\langle M\rangle, s')$. Per induzione, dato che $|SVars(s)| = |SVars(s')|$ e la lunghezza di s' è minore della lunghezza di s, $(t\cdot!\langle M\rangle \cdot s')[R] = (t \cdot s)[R]$ è una traccia.

case $s = ?\langle M\rangle \cdot s'$: In questo caso $\mathsf{Model}(t,s)$ calcola $\mathsf{Constraints}(\overline{\mathsf{Atoms}}(O(t)), M)$. Per il Teorema 7.2.10, la procedura Constraints restituisce le sostituzioni R' tali che $\overline{\mathsf{Atoms}}(O(t))[R'] \vdash M[R']$. Distinguiamo due casi sul risultato della procedura Constraints:

1. $R' = [\,]$: Per definizione di Model, $R \in \mathsf{Model}(t\cdot?\langle M\rangle, s')$. Per induzione, dato che $|SVars(s)| = |SVars(s')|$ e la lunghezza di s' è minore della lunghezza di s, $(t\cdot?\langle M\rangle \cdot s')[R] = (t \cdot s)[R]$ è una traccia.
2. $R' \neq [\,]$: Per definizione di Model, la sostituzione $R = R'@R''$ dove $R'' \in \mathsf{Model}(\epsilon,(t\cdot s)[R'])$. Per induzione, dato che $|SVars((t\cdot s)[R'])| < |SVars(t\cdot s)|$, $(t\cdot?\langle M\rangle \cdot s')[R'][R''] = (t \cdot s)[R'][R'']$ è una traccia. □

Corollario 8.3.20 *Una run s è una traccia se e solo se* $[\,] \in \mathsf{Model}(\epsilon,s)$.

Dim. Deriva direttamente dalla Proposizione 8.3.19 e dalla definizione di Model. □

Finora abbiamo dimostrato che esiste una procedura che calcola le sostituzioni per cui una run sia una traccia. Comunque, per la verifica delle proprietà di sicurezza non interessa studiare se una particolare sequenza di interazioni rispetta tali proprietà, ma si deve verificare che ogni possibile esecuzione del protocollo le soddisfi. Allora estendiamo la procedura Model in modo da analizzare ogni run del grafo simbolico, costruendo così un modello finito in grado di rappresentare esattamente le informazioni contenute nel modello concreto. A differenza del caso in cui le chiavi condivise sono limitate a nomi, non è più necessaria un'istanziazione brute force delle variabili. In questo modo il grafo simbolico transizioni risulta più compatto rendendo la fase di raffinamento più efficiente.

Definizione 8.3.21 *Il* modello simbolico *di un processo chiuso P è l'insieme:*

$$\mathsf{Model}(P) = \{s[R] \mid s \in \mathsf{Run}(P),\ R \in \mathsf{Model}(\epsilon,s)\}$$

dove l'insieme delle run del processo è definito come $Run(P) = \{s \mid \epsilon; P \rightsquigarrow^* s; Q\}$.

Il modello simbolico è quindi l'insieme delle tracce ottenute analizzando tutte le run simboliche del processo. Osserviamo che dobbiamo considerare tutte le possibili sequenze di interazioni del modello astratto e non restringere l'attenzione a run massimali.

Esempio 8.3.22 *Il modello del processo*

$$P = !\langle h(n)\rangle \,.\, ?(x) \,.\, if\ x = n\ then\ !\langle error\rangle \,.\, \mathbf{0}$$

è dato dalla chiusura dei prefissi della traccia $!\langle h(n)\rangle \cdot ?(x)$. *Questo processo ha solo una run massimale, cioè* $!\langle h(n)\rangle \cdot ?(n) \cdot !\langle error\rangle$, *con modello vuoto.*

8.3.3 Rispettabilità

Il modello simbolico per dirsi corretto e completo deve rappresentare tutti e soli i possibili comportamenti del modello concreto. È quindi necessario verificare se i comportamenti del modello simbolico abbiano una controparte in quello concreto. La differenza tra le sequenze di interazioni nei due modelli consiste nel fatto che in quelle del modello simbolico occorrono variabili e quindi risulta evidente che tale strumento possa assumere la forma di sostituzioni ground.
Il concetto di rispettabilità, di seguito definito, è necessario per determinare le sostituzioni ground per cui una traccia del modello simbolico può diventare una sequenza di interazioni del modello concreto, ed è analogo a quello presentato per il caso in cui le chiavi condivise sono ristrette a nomi.

Definizione 8.3.23 *Sia s una run e sia* ρ *una sostituzione ground. Diciamo che* ρ rispetta *s se, per ogni variabile x in s, il giudizio* $O(s_x)[\rho] \vdash x[\rho]$ *è derivabile, dove* s_x *denota il più grande prefisso di s che non contiene x.*

In altre parole, una sostituzione ground rispetta una run se, data una sequenza di interazioni, siamo in grado di dedurre i messaggi sostituiti alle variabili, attraverso lo stato di conoscenza precedente l'introduzione delle variabili. Ricordiamo che la prima occorrenza di una variabile deve essere in un'azione di input.

Definizione 8.3.24 *Sia s una run. Diciamo che s è* rispettabile *se esiste una sostituzione ground* ρ *che la rispetta.*

La nozione di rispettabilità non basta da sola a individuare le run simboliche che hanno una controparte nel modello concreto in quanto è più debole di quella di traccia ground. È da evidenziare tuttavia che ogni traccia del modello simbolico è rispettabile e la combinazione della sequenza di interazione e della sostituzione che la rispetta restituisce una traccia del modello concreto. In questo modo la rispettabilità fornisce uno strumento per passare dal modello simbolico al modello concreto.
La seguente proposizione mostra la relazione tra l'insieme di atomi, calcolato a partire dallo

stato di conoscenza di una traccia, e una sostituzione ground che rispetta la traccia. Come nel caso in cui le chiavi condivise sono limitate a nomi, l'idea è sfruttare che l'insieme degli atomi è minimale. Le ipotesi garantiscono che la sostituzione rispetta la traccia e quindi che siamo in grado di dedurre i messaggi sostituiti alle variabili, attraverso lo stato di conoscenza corrente. Allora, utilizzando la nozione si sequenza in forma normale, risulta facile provare che l'insieme degli atomi è invariante sotto sostituzione.

Proposizione 8.3.25 *Per ogni traccia t e sostituzione ground ρ tale che $t[\rho]$ è una traccia ground, se ρ rispetta t, allora $\overline{\mathsf{Atoms}}(O(t)[\rho]) = \overline{\mathsf{Atoms}}(O(t))[\rho]$.*

Dim. Il risultato si ottiene per induzione sul numero di variabili che occorrono nella traccia similmente alla Proposizione 8.2.36. □

Le prossime due proposizioni sono fondamentali per dimostrare la correttezza del modello ottenuto mediante la funzione Model. La prima afferma che per ogni traccia esiste una sostituzione ground che la rispetta. La seconda proposizione mostra che una traccia a cui è applicata una sostituzione che la rispetta è una sequenza di interazione del modello concreto. Le loro dimostrazioni sono analoghe a quelle date per il caso in cui le chiavi condivise sono ristrette a nomi e per tale motivo le omettiamo.

Proposizione 8.3.26 *Per ogni traccia t, esiste una sostituzione ground ρ tale che ρ rispetta t.*

Dim. Il risultato si ottiene per induzione sulla lunghezza della traccia t similmente al Lemma 8.2.38. □

Dimostriamo ora come ogni traccia del modello simbolico, a cui è applicata una sostituzione ground che la rispetta, è una traccia del modello concreto. Questo risultato è ottenuto provando che, dato un messaggio M, un insieme di messaggi K e una sostituzione ground ρ, se $K \vdash M$ e, per ogni variabile x in M ma non in K, $K[\rho] \vdash x[\rho]$, allora il giudizio $K[\rho] \vdash M[\rho]$ è derivabile. Questo risultato vale in quanto nella derivazione di $K \vdash M$ l'unica regola utilizzabile per dedurre le variabili in M ma non in K, è la regola (*VAR*). Il seguente lemma è analogo al Lemma 8.2.39 e per tale motivo ci limitiamo a provare solo il caso in cui il messaggio è il risultato di un algoritmo crittografico simmetrico di codifica.

Lemma 8.3.27 *Sia M un messaggio, K un insieme di messaggi, ρ una sostituzione che rende sia $K[\rho]$ che $M[\rho]$ ground. Se $K \vdash M$ e, per ogni variabile x in M ma non in K, $K[\rho] \vdash x[\rho]$, allora $K[\rho] \vdash M[\rho]$.*

Dim. Sia M un messaggio, K un insieme di messaggi, $K' = \overline{\mathsf{Atoms}}(K)$ e ρ una sostituzione ground. Il risultato si ottiene per induzione sulla complessità del messaggio M e la dimostrazione è analoga a quella del Lemma 8.2.39. L'unico caso da verificare è quando il messaggio M è della forma $\{M'\}_N$. Per ipotesi, $K' \vdash \{M'\}_N$. Sia x una variabile in M ma non in K. Per la Proposizione 8.3.8, esiste un albero di prova di $K' \vdash \{M'\}_N$ in cui non sono utilizzate regole di eliminazione. L'unica regola costruttiva applicabile è (*SKENC*). Quindi $K' \vdash M'$ e $K' \vdash N$ e x è in M' o in N. Allora, per induzione, $\overline{\mathsf{Atoms}}(K[\rho]) \vdash M[\rho]$. □

Proposizione 8.3.28 *Per ogni traccia t e per ogni sostituzione ground ρ che rispetta t, la run $t[\rho]$ è una traccia.*

Dim. Il risultato si ottiene per induzione sulla lunghezza della traccia t similmente al Lemma 8.2.40. □

L'implicazione inversa della proposizione appena dimostrata non vale, cioè non è vero che ogni sostituzione ground, che rende una traccia del modello simbolico in una del modello concreto, rispetta la sequenza di interazioni del modello simbolico.
La completezza del modello è provata verificando che ogni traccia ground è una concretizzazione del modello simbolico e tale risultato è ottenuto utilizzando la seguente proposizione il cui enunciato esprime come le sostituzioni ground che rendono derivabile un giudizio sono istanze dei vincoli calcolati attraverso la procedura Constraints.

Lemma 8.3.29 *Sia M un messaggio, K un insieme di messaggi e ρ una sostituzione ground tale che $M[\rho]$ è ground e $K[\rho]$ è ground. Se $K[\rho]$ è atomico e $K[\rho] \vdash M[\rho]$, allora esiste $R \in \mathsf{Realise}(K,M)$ tale che ρ è un'istanza di R.*

Dim. Sia M un messaggio, K un insieme di messaggi e ρ una sostituzione ground. Il risultato si ottiene per induzione su un ordine lessicografico che tiene conto delle variabili in $\{M\} \cup K$ e della complessità di M, e la dimostrazione è analoga a quella del Lemma 8.2.43. L'unico caso da verificare è quando il messaggio M è della forma $\{M_1\}_{M_2}$. Per ipotesi, $K[\rho] \vdash \{M_1[\rho]\}_{M_2[\rho]}$. Distinguiamo due casi.

1. $K[\rho] \vdash M_2[\rho]$: In questo caso $K[\rho] \vdash M_1[\rho]$. Per induzione esiste $R_1 \in \mathsf{Realise}(K,M)$ e una sostituzione ground ρ_1 tale che $\rho = R_1@\rho_1$. Allora $K[R_1][\rho_1] \vdash M_2[R_1][\rho_1]$. Per induzione, dato che $|Var(M_2[R_1]) \cup IVar(K[R_1])| < |Var(M) \cup IVar(K)|$, esiste $R_2 \in \mathsf{Realise}(K[R_1],M_2[R_1])$ e una sostituzione ground ρ_2 tale che $\rho_1 = R_2@\rho_2$. Per definizione di Realise, $R_1@R_2 \in \mathsf{Realise}(K,M)$. Allora $\rho = R_1@R_2@\rho_2$.

2. $K[\rho] \not\vdash M_2[\rho]$: Per il Lemma 8.3.11, $M[\rho] \in K[\rho]$. Quindi esiste $N \in K$ tale che $N = \{N_1\}_{N_2}$ e $M[\rho] = N[\rho]$. Allora esiste $R \in \mathsf{Realise}(K,M)$ tale che $R = mgu\{M = N\}$. Per definizione di most general unifier, $\rho = R@\rho'$. □

Proposizione 8.3.30 *Sia M un messaggio, K un insieme di messaggi e ρ una sostituzione ground tale che $M[\rho]$ è ground e $K[\rho]$ è ground. Se $K[\rho]$ è atomico e $K[\rho] \vdash M[\rho]$, allora esiste $R \in \mathsf{Constraints}(K,M)$ tale che ρ è un'istanza di R.*

Dim. La proposizione si dimostra banalmente iterando il Lemma 8.3.29. □

Lemma 8.3.31 *Sia M un messaggio, K un insieme atomico di messaggi e ρ una sostituzione ground tale che $M[\rho]$ e $K[\rho]$ siano ground. Se $K[\rho]$ è atomico, i giudizi $K \vdash M$ e $K[\rho] \vdash M[\rho]$ sono derivabili e la sostituzione vuota è l'unica sostituzione calcolata da $\mathsf{Constraints}(K,M)$, allora, per ogni variabile x in M ma non in K, il giudizio $K[\rho] \vdash x[\rho]$ è derivabile.*

Dim. Sia M un messaggio, K un insieme di messaggi e ρ una sostituzione ground. Il risultato si ottiene per induzione sulla complessità del messaggio M e la dimostrazione è analoga a quella del Lemma 8.2.45. L'unico caso da verificare è quando il messaggio M è della forma $\{M'\}_N$. Sia x una variabile in M ma non in K. Per la Proposizione 8.3.8, esiste un albero di prova di $K \vdash \{M'\}_N$ in cui non sono utilizzate regole di eliminazione. L'unica regola costruttiva applicabile è la regola (*SKENC*). Quindi $K \vdash M'$ e $K \vdash N$. Allora x è in M' o in N. Allora $K[\rho] \vdash M'[\rho]$. Per definizione di Constraints, se $\mathsf{Constraints}(K,M) = \{[\,]\}$, $\mathsf{Constraints}(K,M') = \{[\,]\}$ e $\mathsf{Constraints}(K,N) = \{[\,]\}$. Per induzione $K[\rho] \vdash x[\rho]$. □

8.3.4 Correttezza e Completezza

Come per il caso in cui le chiavi condivise sono ristrette a nomi, si vuole avere la garanzia che il modello calcolato dalla procedura Model possa essere utilizzato per la verifica delle proprietà di sicurezza e a tale scopo dimostriamo la correttezza e la completezza del modello simbolico. Per correttezza si intende che ogni traccia del modello simbolico ha una controparte nel modello concreto, e per completezza si intende che ogni esecuzione che trova posto nel modello concreto ha una controparte nel modello simbolico. In questo modo, se viene trovata una traccia del modello simbolico che viola le proprietà di sicurezza, allora il protocollo che è stato modellato non le soddisfa. Inoltre tali tracce consentono di individuare i possibili attacchi.

Teorema 8.3.32 *Sia s una run. Per ogni $R \in \mathsf{Model}(\epsilon,s)$ la traccia $s[R]$ è rispettabile e, per ogni sostituzione ground ρ, che rispetta $s[R]$ abbiamo che la run ground $s[R][\rho]$ è una traccia ground.*

Dim. Per la Proposizione 8.3.19, $s[R]$ è una traccia con $R \in \mathsf{Model}(\epsilon,s)$. Per la Proposizione 8.3.26, per ogni traccia esiste una sostituzione ground ρ che la rispetta. Per la Proposizione 8.3.28, se $s[R]$ è una traccia ed esiste una sostituzione ground ρ che rispetta $s[R]$, $s[R][\rho]$ è una traccia. □

Dato che le tracce del modello simbolico sono un'astrazione di qualche traccia ground, è immediato verificare che il modello simbolico è corretto.

Corollario 8.3.33 ***(Correttezza)*** *Sia P un processo. Per ogni traccia $t \in \mathsf{Model}_{at}(P)$ esiste una sostituzione ground ρ tale che $t[\rho] \in \mathsf{Trace}(P)$.*

Ora, per provare che il modello simbolico possa essere utilizzato per una verifica automatica delle proprietà di sicurezza e che i risultati ottenuti valgano anche nel modello concreto, rimane da dimostrare che le traccie ground sono una concretizzazione di qualche traccia del modello simbolico.

Teorema 8.3.34 *Sia t una traccia e s una run. Per ogni sostituzione ground ρ tale che ρ rispetta t e $(t \cdot s)[\rho]$ è una traccia, esiste una sostituzione $R \in \mathsf{Model}(t,s)$ e una sostituzione ground ρ' tale che $\rho = R@\rho'$, $(t \cdot s)[R]$ è una traccia e ρ' rispetta $(t \cdot s)[R]$.*

Dim. Sia s una run di lunghezza p con n variabili e ρ una sostituzione ground tale che $s[\rho]$ è una traccia ground. Diamo una dimostrazione per induzione sulla struttura di s, cioè un ordine lessicografico che tiene conto delle variabili che occorrono in s e della lunghezza di s, utilizzando la procedura in Figura 8.4.

Caso base : Se $s = \epsilon$, allora $\mathsf{Model}(\epsilon,s) = \{[\,]\}$. In questo caso $\rho = \rho'$ e quindi, per ipotesi, t è una traccia e ρ rispetta t.

Passo induttivo : Supponiamo che s sia una run di lunghezza $p > 0$ con n variabili. Per definizione di traccia ground, ogni prefisso di una traccia ground è una traccia ground. Sia $s'[\rho]$ il prefisso di $s[\rho]$ di lunghezza $p-1$. Distinguiamo i casi sull'ultima azione di s.

case $s = s'\cdot!\langle M\rangle$: Per ipotesi $(t \cdot s'\cdot!\langle M\rangle)[\rho]$ è una traccia ground e ρ rispetto t. Per definizione di traccia ground, $(t \cdot s')[\rho]$ è una traccia ground. Allora, per induzione, esiste $R \in \mathsf{Model}(t,s')$ e una sostituzione ground ρ' tali che $\rho = R@\rho'$, $(t \cdot s')[R]$ è una traccia e ρ' rispetta $(t \cdot s')[R]$. Per definizione di Model, $\mathsf{Model}(t,s') = \mathsf{Model}(t,s)$. Allora, per definizione di traccia, $(t \cdot s)[R]$ è una traccia e, per definizione di rispettabilità, ρ' rispetta $(t \cdot s)[R]$.

case $s = s'\cdot?\langle M\rangle$: Per ipotesi $(t \cdot s'\cdot?\langle M\rangle)[\rho]$ è una traccia ground e ρ rispetto t. Per definizione di traccia ground, $(t \cdot s')[\rho]$ è una traccia ground. Allora, per induzione, esiste $R \in \mathsf{Model}(t,s')$ e una sostituzione ground ρ' tali che $\rho = R@\rho'$, $(t \cdot s')[R]$ è una traccia e ρ' rispetta $(t \cdot s')[R]$. Inoltre, dato che $(t \cdot s'\cdot?\langle M\rangle)[\rho]$ è una traccia ground, $\mathcal{O}(t \cdot s')[\rho] \vdash M[\rho]$ e quindi $\mathcal{O}(t \cdot s')[R][\rho'] \vdash M[R][\rho']$. Per la Proposizione 8.3.7, $\overline{\mathsf{Atoms}}(\mathcal{O}(t \cdot s')[R][\rho']) \vdash M[R][\rho']$. Per la Proposizione 8.3.25, $\overline{\mathsf{Atoms}}(\mathcal{O}(t \cdot s')[R])[\rho'] \vdash M[R][\rho']$. Allora, per la Proposizione 8.3.30, esiste $R_0 \in \mathsf{Constraints}(\overline{\mathsf{Atoms}}(\mathcal{O}(t \cdot s')[R]),M[R])$ e una sostituzione ground ρ'' tale che $\rho' = R_0@\rho''$. Distinguiamo due casi.

1. Esiste $R_0 \neq [\,]$: Per definizione di rispettabilità, ρ'' rispetta ϵ e, per ipotesi $(t \cdot s)[R@R_0][\rho'']$ è una traccia ground. Allora, per induzione, dato che $|SVars(s[R@R_0])| \leq |SVars(s[R])|$, esiste $R_1 \in \mathsf{Model}(\epsilon,(t \cdot s)[R@R_0])$ e una sostituzione ground ρ''' tali che $\rho'' = R_1@\rho'''$, $(t \cdot s)[R@R_0@R_1]$ è una traccia e ρ''' rispetta $(t \cdot s)[R@R_0@R_1]$. Per definizione di Model, $R@R_0@R_1 \in \mathsf{Model}(t,s)$. Allora $\rho = R@R_0@R_1@\rho'''$, $(t \cdot s)[R@R_0@R_1]$ è una traccia e ρ''' rispetta $(t \cdot s)[R@R_0@R_1]$.
2. $R_0 = [\,]$ è l'unica soluzione: Per definizione di Model, $R \in \mathsf{Model}(t,s)$ e $\rho = R@\rho'$. Dato che $(t \cdot s')[R]$ è una traccia e $\overline{\mathsf{Atoms}}(\mathcal{O}(t \cdot s')[R]) \vdash M[R]$, $(t \cdot s)[R]$ è una traccia. Inoltre, dato che $(t \cdot s)[\rho]$ è una traccia ground, $\overline{\mathsf{Atoms}}(\mathcal{O}(t \cdot s')[\rho]) \vdash M[\rho]$. Per il Lemma 8.3.31, per ogni variabile x in $M[R]$ ma non in $\mathcal{O}(t \cdot s')[R]$, $\overline{\mathsf{Atoms}}(\mathcal{O}(t \cdot s')[R])[\rho'] \vdash x[\rho']$. Allora, per definizione di rispettabilità, ρ' rispetta $(t \cdot s)[R]$. □

Avendo provato come ogni traccia ground ha una controparte in quello simbolico, è immediato verificare che il modello simbolico è completo.

Corollario 8.3.35 ***(Completezza)*** *Sia P un processo. Per ogni traccia ground $t \in \mathsf{Trace}(P)$ esiste una traccia $s \in \mathsf{Model}(P)$ e una sostituzione ground ρ tale che $t = s[\rho]$.*

Parte III

Definizione e Verifica di Protocolli di Autenticazione - Conclusioni

Introduzione

Concludiamo il nostro lavoro facendo il punto dei risultati conseguiti. Quest'ultima parte è divisa in tre capitoli.
Il *Capitolo 10* definisce la nozione di sicurezza che consente di stabilire la correttezza dei protocolli. Presenta, inoltre alcuni esempi di applicazione dell'analisi, definita nel nostro progetto, per fornire un metodo di lettura e interpretazione dei risultati ottenuti. In particolare, si sofferma su un protocollo elementare di autenticazione, proposto da Woo e Lam [96] per analizzare le proprietà di corrispondenza e segretezza, e sul protocollo di Needham-Schroeder [70] nella versione proposta da Lowe [56]. Confronta i risultati ottenuti con quelli dell'analisi effettuata da Schneider [83] sullo stesso protocollo per evidenziare l'importanza della definizione di autenticazione.
Il *Capitolo 11* confronta la nostra analisi con quelle presenti in letteratura, cercando di individuarne pregi e difetti.
Il *Capitolo 12* descrive i risultati raggiunti con questo progetto e dà alcuni spunti per estendere l'analisi.

Capitolo 9

Definizione di Autenticazione e Verifica di Protocolli

Prima di applicare l'analisi ai protocolli dobbiamo definire il significato di correttezza per protocolli di autenticazione, cioè quali siano i requisiti che danno certezza dell'identificazione degli agenti che partecipano al protocollo e della segretezza delle informazioni riservate trasmesse. Le proprietà dei protocolli da verificare sono diretta conseguenza della definizione di autenticazione e tali proprietà sono espresse mediante la struttura del protocollo.
Questo capitolo specifica la nozione di autenticazione, utilizzata nel nostro approccio, la quale consente di determinare la correttezza dei protocolli. Inoltre, presenta due esempi di applicazione della nostra analisi per la verifica delle proprietà di autenticazione di differenti protocolli.

9.1 Correspondence e Secrecy

Abbiamo sin qui notato come i vari ricercatori si sono sforzati di dare una risposta al concetto di autenticazione. Ognuno di essi ha proposto il suo punto di vista, formulando definizioni più o meno forti. Le analisi sia formali che informali dei protocolli sono consequenziali alla definizione di autenticazione. Occorre, quindi, tener presente che la correttezza del protocollo è legata alla definizione data, cioè agli obiettivi prefissati.
Ci sono due difetti negli approcci esistenti delle analisi formali. Il primo è un salto significativo tra la nozione formale e quella intuitiva di correttezza. La nozione formale di correttezza è spesso altamente specializzata e non sempre cattura interamente la nozione intuitiva che il progettista di un protocollo desidera. Per esempio, alcuni approcci [31, 55] adottano la segretezza come loro principale criterio di correttezza, ma non indirizzano l'analisi direttamente alla segretezza, mentre altri [22, 44] specificano la correttezza in termini di *state of belief* rendendo implicita la segretezza e questo salto può portare a potenziali fraintendimenti.
Il secondo difetto è che la nozione formale di correttezza, adottata in molti approcci esistenti, non è sufficientemente formalizzata e tale imprecisione può causare problemi nella fase di

verifica dei protocolli. Per di più, con la nozione di correttezza insufficientemente formalizzata, è difficile giudicare la generalità del metodo proposto.
Per risolvere questi difetti, proponiamo, seguendo l'approccio proposto da Woo e Lam [96], una formalizzazione della correttezza basata su due tipi di proprietà primitive, chiamate *correspondence* e *secrecy*. La proprietà di *correspondence* è indirizzata a obiettivi di autenticazione, mentre la proprietà di *secrecy* è rivolta a obiettivi di distribuzione delle chiavi. Informalmente le proprietà di sicurezza esprimono un legame tra il verificarsi di determinati eventi, come l'avvenuta autenticazione tra due agenti, e il passaggio sulla rete di dati particolari.
È nostra intenzione mantenere una netta separazione tra modello simbolico e la correttezza delle proprietà di sicurezza. Così se, per una particolare applicazione, sono necessarie nuove proprietà di correttezza (oltre a correspondence e a secrecy), possiamo facilmente definirle usando il modello simbolico. Diamo ora la nozione di correttezza per protocolli di autenticazione.

9.1.1 Correspondence

Un agente, che si vuole autenticare, deve essere sicuro di parlare con l'agente desiderato, cioè bisogna consentire agli agenti coinvolti nella comunicazione di ottenere sufficienti garanzie sull'identità degli altri agenti che partecipano al protocollo. Per l'analisi dei protocolli, questo obiettivo di alto livello deve essere tradotto in una proprietà più primitiva. Ricordiamo che i protocolli sono stati definiti come uno schema che stabilisce il formato e la sequenza di messaggi che due o più agenti, in accordo sul loro utilizzo, devono inviarsi. La proprietà di corrispondenza afferma che la comunicazione deve avvenire seguendo la sequenza di passi fissati dal protocollo. In particolare, quando un agente che si deve autenticare termina le sue operazioni, l'altro deve essere presente e adempiere alle proprie. Formalizziamo ora questa idea.

Definizione 9.1.1 *Una sequenza di interazioni non soddisfa la proprietà di corrispondenza se rappresenta un'esecuzione massimale del protocollo e non rispetta il formato e l'ordine dei messaggi fissati dal protocollo.*

Per determinare se un protocollo soddisfa la proprietà di corrispondenza non interessa studiare se una generica sequenza di interazioni soddisfa tale proprietà, ma si deve verificare che tutte le tracce del modello soddisfino tale proprietà.

Definizione 9.1.2 *Un protocollo Π non soddisfa la proprietà di corrispondenza se esiste una traccia $t \in \mathsf{Model}(P)$ che non soddisfa la proprietà di corrispondenza, dove P è il processo che modella il protocollo Π.*

Analizziamo così ogni traccia del modello calcolato confrontandola con le specifiche del protocollo. Se questo non soddisfa la proprietà, il metodo restituisce le traccie che violano le proprietà e con le quali possiamo facilmente ricostruire gli eventuali attacchi di un agente ostile. Inoltre, siamo garantiti che la verifica della proprietà sul modello, ottenuto applicando la procedura Model, termina in quanto abbiamo dimostrato che il modello è finito.

9.1.2 Secrecy

Tutti i protocolli crittografici basano il loro funzionamento sulla presenza di alcune informazioni segrete allo scopo di garantire la sicurezza delle comunicazioni. Tali informazioni sono note al più ai soli partecipanti ad una sessione del protocollo e la loro compromissione comporta la perdita della sicurezza nelle trasmissioni. La segretezza di sistemi crittografici è quindi quella parte della sicurezza riguardante il flusso di informazioni da una sorgente segreta a osservatori non autorizzati, in aggiunta alle capacità di questi osservatori ad estrarre conoscenze utili dalle informazioni trasmesse.
Un agente, che prende parte a un protocollo, vuole avere la sicurezza che nessun agente ostile possa venire a conoscenza dei suoi segreti. Per l'analisi dei protocolli, questo obiettivo di alto livello deve essere tradotto in una proprietà di correttezza più primitiva. La proprietà di segretezza afferma che informazioni riservate non devono essere accessibili ad un agente ostile. Per verificare se una sequenza di interazioni gode della proprietà di segretezza, si deve controllare che le informazioni riservate non siano deducibili dai messaggi che appartengono alla conoscenza iniziale dell'ambiente e dai messaggi spediti dagli agenti. Esprimiamo ora la proprietà di segretezza di un protocollo in termini delle tracce che genera.

Definizione 9.1.3 *Sia s una sequenza di interazioni e M un messaggio riservato. La sequenza s soddisfa la proprietà di segretezza se il giudizio $O(s) \vdash M$ non è derivabile.*

In altre parole, una sequenza di interazioni soddisfa la proprietà di segretezza se la combinazione delle sue azioni non rivela all'ambiente l'informazione riservata. Per determinare, però, se un protocollo soddisfa la proprietà di segretezza non interessa studiare se una generica sequenza di interazioni la soddisfa, ma si deve verificare che tutte le tracce del modello rispettino tale proprietà.

Definizione 9.1.4 *Un protocollo Π soddisfa la proprietà di segretezza se ogni traccia $t \in \mathsf{Model}(P)$ soddisfa la proprietà di segretezza, dove P è il processo che modella il protocollo Π.*

Un metodo per verificare la proprietà di segretezza è estendere il processo da analizzare in modo da considerare la presenza di osservatori nel modello. Calcoliamo $\mathsf{Model}(P|O)$, dove P è il processo che modella il protocollo Π e O è il processo che modella l'osservatore e la cui specifica nel calcolo è data dal processo

$$\begin{aligned} O &= ?(x) \,.\, O'[x] \\ O'[x] &= if\ x = M\ then\ !\langle error\rangle \,.\, \mathbf{0} \end{aligned}$$

dove M è l'informazione riservata. Se non esiste nessuna traccia nel modello ottenuto contenente l'azione $!\langle error\rangle$, allora il protocollo Π soddisfa la proprietà di segretezza.

9.2 Esempi

Questa sezione presenta due esempi di applicazione della nostra analisi per la verifica delle proprietà di autenticazione di differenti protocolli. Analizziamo prima un protocollo elementare di autenticazione unilaterale proposto da Woo e Lam [96] e successivamente verifichiamo se il protocollo di Needham-Schroeder [70], nella versione proposta da Lowe [56],

autentica correttamente due agenti che intendono attivare una sessione di comunicazione. Confrontiamo i risultati ottenuti con quelli dell'analisi effettuata da Schneider [83] sull'ultimo protocollo per mettere in evidenza l'importanza della definizione di autenticazione.
La verifica viene eseguita, percorrendo i passi presentati in questa tesi, nel rispetto del seguente schema.

1. Traduzione del protocollo in un processo espresso nel calcolo utilizzato.
2. Costruzione del modello simbolico tramite la procedura Model.
3. Verifica della proprietà di corrispondenza.
4. Verifica della proprietà di segretezza.

Per rendere più chiaro il modello, calcolato attraverso la funzione Model, esplicitiamo nelle tracce ottenute l'agente che compie l'azione, scrivendo $!_P\langle M\rangle$ e $?_P\langle M\rangle$ per indicare, rispettivamente, che l'agente P ha spedito o ricevuto il messaggio M.

9.2.1 Esempio 1: Protocollo di Autenticazione Unilaterale di Woo e Lam

Presentiamo un protocollo di autenticazione unilaterale, proposto da Woo e Lam [96], per verificare le proprietà di corrispondenza e segretezza attraverso la nostra analisi.
Il seguente protocollo di autenticazione unilaterale Π è definito per autenticare l'agente Q all'agente P e per distribuire una nuova chiave di sessione dall'agente P all'agente Q. Π è costituito dai seguenti due messaggi.

$$\begin{array}{lllll} \text{Messaggio 1} & P \rightarrow Q & : & p,\{k\}_{k_{PQ}} \\ \text{Messaggio 2} & Q \rightarrow P & : & q,\{k\}_{k_{PQ}} \end{array}$$

L'agente P invia un messaggio composto dal nome che lo identifica, p, e da una nuova chiave di sessione k codificata con la chiave condivisa k_{PQ}. L'agente Q risponde a P con un messaggio composto dal nome che lo identifica, q, e dalla chiave k codificata con la chiave condivisa k_{PQ}.
Dalla descrizione del protocollo risulta che l'unica informazione necessaria agli agenti per portare a termine correttamente una sessione di comunicazione è la chiave condivisa k_{PQ}. Questo protocollo è rappresentato dal processo $P|Q$, dove P e Q sono processi sequenziali che rappresentano il comportamento dei rispettivi agenti. Diamo la loro specifica nel calcolo:

$$\begin{array}{lcllcl} P & = & !\langle(p,\{k\}_{k_{PQ}})\rangle \,.\, ?(x) \,.\, P'[x] & Q & = & ?(x) \,.\, Q'[x] \\ P'[x] & = & \mathit{let}\ x = (q,y)\ \mathit{in}\ P''[y] & Q'[x] & = & \mathit{let}\ x = (p,y)\ \mathit{in}\ Q''[y] \\ P''[y] & = & \mathit{case}\ y\ \mathit{of}\ \{z\}_{k_{PQ}}\ \mathit{in}\ P'''[z] & Q''[y] & = & \mathit{case}\ y\ \mathit{of}\ \{z\}_{k_{PQ}}\ \mathit{in}\ Q'''[z] \\ P'''[z] & = & \mathit{if}\ z = k\ \mathit{then}\ \mathbf{0} & Q'''[z] & = & !\langle(q,\{z\}_{k_{PQ}})\rangle \,.\, \mathbf{0} \end{array}$$

In [96] è stato dimostrato che il protocollo non soddisfa la proprietà di corrispondenza; anche noi accreditiamo tale tesi con il metodo di analisi presentato. Assumendo che l'ambiente conosca il nome degli agenti che partecipano al protocollo, calcoliamo l'insieme delle sequenze di interazioni che costituiscono il modello simbolico attraverso la funzione Model$(P|Q)$. Riportiamo di seguito le sequenze di interazioni più significative ottenute:

$$
\begin{array}{ll}
(i) & !_P\langle(p,\{k\}_{k_{PQ}})\rangle \cdot ?_P\langle(q,\{k\}_{k_{PQ}})\rangle \\
(ii) & !_P\langle(p,\{k\}_{k_{PQ}})\rangle \cdot ?_Q\langle(p,\{k\}_{k_{PQ}})\rangle \cdot ?_P\langle(q,\{k\}_{k_{PQ}})\rangle \cdot !_Q\langle(q,\{k\}_{k_{PQ}})\rangle \\
(iii) & !_P\langle(p,\{k\}_{k_{PQ}})\rangle \cdot ?_Q\langle(p,\{k\}_{k_{PQ}})\rangle \cdot !_Q\langle(q,\{k\}_{k_{PQ}})\rangle \cdot ?_P\langle(q,\{k\}_{k_{PQ}})\rangle
\end{array}
$$

Osservando i risultati ottenuti, notiamo che la proprietà di corrispondenza non è verificata. Esistono, infatti, alcune tracce nel modello in cui l'agente P termina le azioni previste dal protocollo prima che l'agente Q spedisca il messaggio di risposta. In particolare, nella traccia (i) l'agente P esegue le sue azioni senza la partecipazione dell'agente Q. La sequenza (iii), invece, rispetta la corretta esecuzione del protocollo, ma non è sufficiente per affermare che il protocollo funzioni correttamente, in quanto la proprietà in analisi deve valere per tutte le tracce del modello. Allora il protocollo analizzato potrebbe essere soggetto ad un attacco. Osservando il seguente scenario, in cui viene esplicitato l'attaccante, verifichiamo come il protocollo risulti effettivamente non corretto. Indichiamo con Z l'agente ostile che interagisce con il protocollo. L'agente racchiuso tra parentesi è quello a cui l'agente ostile vuole sostituirsi.

$$
\begin{array}{lll}
\text{Messaggio 1} & P \rightarrow (Q)Z & : \quad p,\{k\}_{k_{PQ}} \\
\text{Messaggio 2} & (Q)Z \rightarrow P & : \quad q,\{k\}_{k_{PQ}}
\end{array}
$$

L'attacco al protocollo può essere simulato analizzando il processo $P|Q|Z$, dove P e Q sono i processi presentati prima, mentre Z è il processo che rappresenta il comportamento dell'agente ostile. La specifica del processo Z nel calcolo è data da:

$$
\begin{array}{lcl}
Z & = & ?(x) \,.\, Z'[x] \\
Z'[x] & = & \mathit{let}\ x = (p,y)\ \mathit{in}\ Z''[y] \\
Z''[y] & = & !\langle(q,y)\rangle \,.\, \mathbf{0}
\end{array}
$$

Il modello simbolico è dato dalle soluzioni di $\mathsf{Model}(P|Q|Z)$. Riportiamo di seguito le sequenze di interazioni più significative calcolate attraverso la nostra analisi.

$$
\begin{array}{ll}
(i) & !_P\langle(p,\{k\}_{k_{PQ}})\rangle \cdot ?_P\langle(q,\{k\}_{k_{PQ}})\rangle \\
(ii) & !_P\langle(p,\{k\}_{k_{PQ}})\rangle \cdot ?_Z\langle(p,\{k\}_{k_{PQ}})\rangle \cdot !_Z\langle(q,\{k\}_{k_{PQ}})\rangle \cdot ?_P\langle(q,\{k\}_{k_{PQ}})\rangle \\
(iii) & !_P\langle(p,\{k\}_{k_{PQ}})\rangle \cdot ?_Q\langle(p,\{k\}_{k_{PQ}})\rangle \cdot !_Q\langle(q,\{k\}_{k_{PQ}})\rangle \cdot ?_P\langle(q,\{k\}_{k_{PQ}})\rangle
\end{array}
$$

Dai risultati ottenuti si nota che l'attacco può essere portato con successo. Nella sequenza (i) l'agente P termina le sue azioni senza la partecipazione degli agenti Q e Z in quanto nel modello sono presenti attaccanti impliciti. Nella traccia (ii) l'agente ostile Z riesce a sostituirsi con successo all'agente Q, autenticandosi con l'agente P. La traccia (iii), invece, identifica il caso in cui l'agente Q riesce ad autenticarsi a P, rappresentando la corretta esecuzione del protocollo.

Comunque, il protocollo Π preserva la segretezza della chiave k, in quanto un agente ostile non è in grado di ottenere la chiave distribuita durante l'esecuzione di Π come si può facilmente verificare assumendo la presenza di osservatori nel medello. Questo risultato garantisce che la proprietà di segretezza è rispettata dal protocollo analizzato, ma assume poca importanza in quanto il protocollo non soddisfa la proprietà di corrispondenza. Il fallimento del protocollo Π è da attribuire alle informazioni che non devono essere capite per

essere utilizzate, infatti, l'agente ostile si limita a rispedire a P la chiave cifrata senza doverla decriptare.
Durante l'analisi del protocollo Π abbiamo assunto che l'ambiente conosca il nome degli agenti che partecipano al protocollo. Senza questa assunzione, la nostra analisi afferma che il protocollo funziona correttamente quando viene eseguito una prima volta, infatti, il modello non contiene alcuna traccia che viola le proprietà di sicurezza. Nelle sessioni successive, cioè quando gli agenti eseguono nuovamente il protocollo, l'ambiente conosce già i nomi dei partecipanti e si ricade nel caso da noi analizzato.

9.2.2 Esempio 2: Protocollo di Needham-Schroeder con chiave pubblica

Introduciamo una versione semplificata del protocollo di Needham-Schroeder con chiave pubblica proposta da Lowe [56]. Questo protocollo ha lo scopo di fornire mutua autenticazione fra le componenti di un sistema che intendono iniziare una sessione di comunicazione, per cui la correttezza è legata esclusivamente alla validità della proprietà di corrispondenza. Il protocollo adotta crittografia a chiave pubblica; ogni agente è quindi in possesso di una chiave segreta e di una chiave pubblica disponibile ad ogni altra componente del sistema.
La versione originale del protocollo di Needham-Schroeder [70] assume che gli agenti non conoscano la chiave pubblica degli altri, prevedendo uno scambio di messaggi fra gli agenti e un trust server S allo scopo di ottenere tali chiavi.

$$
\begin{array}{lcl}
\text{Messaggio 1} & A \rightarrow S & : \quad B \\
\text{Messaggio 2} & S \rightarrow A & : \quad [\![k_b^+,B]\!]_{k_s^-} \\
\text{Messaggio 3} & A \rightarrow B & : \quad \{\!|N_a,A|\!\}_{k_b^+} \\
\text{Messaggio 4} & B \rightarrow S & : \quad A \\
\text{Messaggio 5} & S \rightarrow B & : \quad [\![k_a^+,A]\!]_{k_s^-} \\
\text{Messaggio 6} & B \rightarrow A & : \quad \{\!|N_a,N_b|\!\}_{k_a^+} \\
\text{Messaggio 7} & B \rightarrow A & : \quad \{\!|N_b|\!\}_{k_b^+}
\end{array}
$$

Se gli agenti sono già in possesso delle chiavi pubbliche, il protocollo può essere così semplificato:

$$
\begin{array}{lcl}
\text{Messaggio 1} & A \rightarrow B & : \quad A,B,\{\!|N_a,A|\!\}_{k_b^+} \\
\text{Messaggio 2} & B \rightarrow A & : \quad B,A,\{\!|N_a,N_b,B|\!\}_{k_a^+} \\
\text{Messaggio 3} & A \rightarrow B & : \quad A,B,\{\!|N_b|\!\}_{k_b^+}
\end{array}
$$

Informalmente, il protocollo funziona come segue: tramite il primo messaggio A invia a B il nonce N_a codificato con la chiave pubblica di B. Con la ricezione del secondo messaggio, contenente il nonce N_a, A ha la garanzia che sia stato B ad inviare il messaggio, in quanto solo B è in possesso della chiave che consente di decodificare il primo messaggio. Inoltre, poiché B con il secondo messaggio, similmente a quanto fatto da A tramite il primo, ha inviato il nonce N_b codificato con la chiave pubblica di A, B ha la garanzia che sia stato A a generare il terzo messaggio che contiene N_b in quanto solo A è in possesso della chiave che consente di

decodificare il secondo messaggio. Dalla descrizione del protocollo, le uniche informazioni necessarie agli agenti per portare a termine correttamente una sessione di comunicazione sono le seguenti:

- L'insieme delle chiavi pubbliche di tutti gli agenti del sistema.
- La chiave privata dell'agente stesso.
- L'insieme dei nonce da utilizzare nelle varie sessioni di comunicazione.

Dato che un nonce è un valore generato in modo casuale, la probabilità che due sessioni di comunicazione utilizzino due nonce uguali è esponenzialmente bassa per cui si può assumere che due sessioni distinte utilizzano sempre nonce differenti.

Analisi Simbolica

Il protocollo di Needham-Schroeder è modellato dal processo $A|B$, dove A e B sono processi sequenziali che rappresentano il comportamento dei rispettivi agenti. Diamo la loro specifica nel calcolo.

$$
\begin{array}{lcl}
A & = & !\langle((a,b),\{\!|(N_a,a)|\!\}_{k_b^+})\rangle \,.\, ?(x_1) \,.\, A^{(i)}[x_1] \\
A^{(i)}[x_1] & = & \mathit{let}\ x_1 = ((b,a),x_2)\ \mathit{in}\ A^{(ii)}[x_2] \\
A^{(ii)}[x_2] & = & \mathit{case}\ x_2\ \mathit{of}\ \{\!|x_3|\!\}_{k_a^-}\ \mathit{in}\ A^{(iii)}[x_3] \\
A^{(iii)}[x_3] & = & \mathit{let}\ x_1 = ((N_a,x_4),b)\ \mathit{in}\ A^{(iv)}[x_4] \\
A^{(iv)}[x_4] & = & !\langle((a,b),\{\!|x_4|\!\}_{k_b^+})\rangle \,.\, \mathbf{0} \\
B & = & ?(y_1) \,.\, B^{(i)}[y_1] \\
B^{(i)}[y_1] & = & \mathit{let}\ y_1 = ((y_2,b),y_3)\ \mathit{in}\ B^{(ii)}[y_2,y_3] \\
B^{(ii)}[y_2,y_3] & = & \mathit{case}\ y_3\ \mathit{of}\ \{\!|y_4|\!\}_{k_b^-}\ \mathit{in}\ B^{(iii)}[y_2,y_4] \\
B^{(iii)}[y_2,y_4] & = & \mathit{let}\ y_4 = (y_5,y_6)\ \mathit{in}\ B^{(vi)}[y_2,y_5,y_6] \\
B^{(iv)}[y_2,y_5,y_6] & = & \mathit{if}\ y_2 = y_6\ \mathit{then}\ B^{(v)}[y_5,y_6] \\
B^{(v)}[y_5,y_6] & = & !\langle((b,y_6),\{\!|((y_5,N_b),b)|\!\}_{y_6})\rangle \,.\, ?(y_7) \,.\, B^{(vi)}[y_5,y_6,y_7] \\
B^{(vi)}[y_5,y_6,y_7] & = & \mathit{let}\ x_7 = ((y_8,b),y_9)\ \mathit{in}\ B^{(vii)}[y_5,y_6,y_8,y_9] \\
B^{(vii)}[y_5,y_6,y_8,y_9] & = & \mathit{if}\ y_6 = y_8\ \mathit{then}\ B^{(viii)}[y_5,y_8,y_9] \\
B^{(viii)}[y_5,y_8,y_9] & = & \mathit{case}\ y_9\ \mathit{of}\ \{\!|y_{10}|\!\}_{k_b^-}\ \mathit{in}\ B^{(ix)}[y_5,y_8,y_{10}] \\
B^{(ix)}[y_5,y_8,y_{10}] & = & \mathit{if}\ y_{10} = N_b\ \mathit{then}\ \mathbf{0}
\end{array}
$$

Assumiamo, come abbiamo fatto nell'esempio precedente, che l'ambiente conosca il nome che identifica gli agenti partecipanti al protocollo. Calcolando l'insieme delle tracce che costituiscono il modello simbolico, attraverso la funzione $\mathsf{Model}(A|B)$, vediamo che nessuna sequenza di interazioni trovata viola la proprietà di corrispondenza. Possiamo perciò concludere che il protocollo di Needham-Schroeder, per la nostra analisi, è in grado di fornire mutua autenticazione a due agenti che intendono iniziar una sessione di comunicazione.
Lowe [57] ha dimostrato come il protocollo di Needham-Schroeder con Public Key non è corretto se utilizza un algoritmo crittografico di tipo ECB [85], mostrando un attacco alla sicurezza del protocollo stesso. Ricordiamo che la nostra analisi assume crittografia perfetta e

si occupa di determinare attacchi solo alla struttura del protocollo. Supponiamo, quindi, che possano esistere meccanismi crittografici in grado di garantire la sicurezza di tale protocollo, lasciando, però, l'ultima parola agli esperti di crittografia.

Confronto con Schneider

Soffermiamoci un istante sull'importanza che assume la definizione di sicurezza durante la fase di verifica confrontando i risultati ottenuti con diverse tecniche formali di analisi. L'analisi effettuata da Schneider [83] sul protocollo di Needham-Schroeder a chiave pubblica, propone di verificare se il protocollo autentica correttamente due agenti che attivano una sessione di comunicazione. Tale verifica dimostra come, nel caso in cui la componente del sistema A abbia iniziato una sessione del protocollo di Needham-Schroeder con l'altra componente B, le due si autenticano correttamente solo se A è in grado di produrre l'ultimo messaggio del protocollo. Questa definizione di autenticazione non garantisce, a nostro avviso, la correttezza del protocollo, infatti, verificando l'originalità solo dell'ultimo messaggio, si controlla semplicemente che B autentichi A, ma non che B sia stato autenticato correttamente da A. Poiché l'analisi non prende in esame i passi precedenti del protocollo e in particolare non assume che N_a sia segreto, nel caso in cui il protocollo non potesse garantire l'autenticazione di B da parte di A, la verifica potrebbe non rivelare tale anomalia.
La verifica del protocollo, nel caso appena descritto, evidenzia l'errore nella definizione di correttezza schematizzando la modalità per attaccare il protocollo in questa situazione. Sia e il nome che identifica l'ambiente e k_e^+ e k_e^- la coppia di chiavi associata all'ambiente. Calcolando il modello simbolico del protocollo, attraverso la funzione Model, otteniamo, tra gli elementi del modello, la seguente sequenza di interazioni:

$$!_A\langle((a,b),\{\!|(N_a,a)|\!\}_{k_b^+})\rangle \cdot ?_B\langle((e,b),\{\!|(N_a,e)|\!\}_{k_b^+})\rangle \cdot !_B\langle(((b,e),\{\!|((N_a,N_b),b)|\!\}_{k_e^+})\rangle \cdot$$
$$?_A\langle(((b,a),\{\!|((N_a,N_b),b)|\!\}_{k_a^+})\rangle \cdot !_A\langle((a,b),\{\!|N_b|\!\}_{k_b^+})\rangle \cdot ?_B\langle((a,b),\{\!|N_b|\!\}_{k_b^+})\rangle$$

Analizzando la traccia di computazione calcolata, un agente ostile potrebbe interagire con il protocollo per disturbarne il corretto funzionamento. Supponendo, infatti, che l'ambiente riesca a ottenere N_a, esso potrebbe compromettere l'autenticazione di A con B tramite l'attacco che riportiamo di seguito senza per questo essere rilevato dalla verifica di Schneider.

Messaggio 1	$A \rightarrow I(B)$	:	$A,B,\{\!\|N_a,A\|\!\}_{k_b^+}$
Messaggio 1'	$I \rightarrow B$	:	$I,B,\{\!\|N_a,I\|\!\}_{k_b^+}$
Messaggio 2'	$B \rightarrow I$	:	$B,I,\{\!\|N_a,N_b,B\|\!\}_{k_i^+}$
Messaggio 2	$I(B) \rightarrow A$	:	$B,A,\{\!\|N_a,N_b,B\|\!\}_{k_a^+}$
Messaggio 3	$A \rightarrow B$	:	$A,B,\{\!\|N_b\|\!\}_{k_b^+}$

Quindi A è convinto di essersi autenticato con B, mentre, a causa della compromissione di N_a e all'ignara complicità di B, l'intruso è riuscito a celare la propria identità con quella di B attraverso un Man-in-the-Middle Attack.

Capitolo 10

Analogie e Differenze con le Tecniche Simboliche in Letteratura

Dopo aver presentato la nostra analisi e la sua potenzialità, vogliamo confrontarla con le tecniche simboliche presenti in letteratura mettendo in luce analogie e differenze.
I metodi automatici per la verifica delle proprietà dei protocolli di sicurezza sono molto spesso basati sulla generazione esplicita del modello del protocollo, infatti la loro analisi rientra nel model checking: dato un modello, che rappresenta il comportamento del protocollo, e una proprietà, si vuole decidere se il modello soddisfa tale proprietà. In questa tesi abbiamo proposto un'analisi per protocolli di autenticazione e fornito alcune condizioni sufficienti sotto cui la loro verifica può essere eseguita mediante un metodo simbolico. L'analisi di un protocollo consiste nello stabilire se un attaccante può generare una sua esecuzione che viola una delle sue condizioni di correttezza, cioè se esiste un'istanziazione delle variabili che verifica i vincoli imposti da una traccia che non soddisfa le proprietà di sicurezza.
Un protocollo può essere descritto come un sistema di processi concorrenti e l'analisi delle tracce generate può essere utilizzata per verificare le proprietà di autenticazione e segretezza. Il maggior inconveniente di questo approccio è dato dal numero infinito di tracce generate dall'esecuzione del protocollo, e ciò si trasmette al modello dell'ambiente, il cui comportamento risulta così imprevedibile. Perciò, piuttosto che cercare di descrivere questo comportamento con un processo specifico, si assume che la rete di comunicazione sia sotto il controllo dell'ambiente (come proposto nel modello di Dolev-Yao [31]). Quindi, un agente in attesa di un input potrebbe ricevere uno degli infiniti messaggi, che l'ambiente conosce e trasmette sulla rete, generando un'esplosione degli stati che rende il modello del protocollo, tipicamente un grafo di transizione degli stati, infinito (più precisamente a ramificazione infinita).
Molti approcci, basati sul model checking a stati finiti, seguono il modello degli attaccanti di Dolev-Yao. Questi metodi, imponendo un limite ai parametri critici (ad esempio, numero di chiavi, numero di accoppiamenti e codifiche nei messaggi), riducono il modello a una dimensione finita e lo esplorano per controllare se è raggiunto qualche stato insicuro. Un'esplorazione esaustiva dello spazio degli stati è così possibile mediante tecniche standard. In particolare, per rendere il model checking a stati finiti applicabile all'analisi dei protocolli

sono necessarie due condizioni:

(i) Un limite al numero di esecuzioni del protocollo.

(ii) Un limite al numero di possibili messaggi che l'attaccante può generare e spedire agli agenti onesti in qualunque momento.

Queste condizioni consentono di descrivere il protocollo come una macchina a stati finiti e di esprimere formalmente le proprietà di raggiungibilità, come segretezza e autenticazione, mediante qualche strumento come la logica temporale. Questi limiti, inoltre, devono essere scelti attentamente poiché, perfino quando sono giustificati e il modello è finito, il fattore di ramificazione delle azioni di input potrebbe causare l'esplosione del numero di stati e di tracce. Scartare una delle due condizioni porta a modelli infiniti: trascurare la condizione (i) porta all'indecidibilità dell'analisi dei protocolli a meno di severe restrizioni; piuttosto, è meno chiaro come estendere la condizione (ii) affinché sia preservata la decidibilità e l'efficienza del metodo. Per le osservazioni riportate, risulta evidente come sia molto difficile stabilire la completezza degli approcci basati sul model checking a stati finiti. Contrariamente a questi ultimi, il nostro metodo può analizzare l'intero spazio infinito degli stati generato da un numero finito di partecipanti. Tale metodo risulta efficiente, in quanto il modello simbolico è compatto e la procedura di raffinamento, utilizzata nell'analisi, è invocata solo su richiesta e su singole tracce simboliche. Comunque, affermazioni sull'efficienza devono essere generalmente prese con molta attenzione, dato che il problema dell'analisi dei protocolli è NP-hard [81].
Presentiamo ora un confronto con altre analisi simboliche basato sulle funzioni crittografiche supportate dai vari metodi e, nel caso di crittografia simmetrica, se esse sono o meno in grado di considerare chiavi composte. Questi aspetti consentono di stabilire la classe dei protocolli crittografici che i metodi sono in grado di analizzare. Ad esempio, Boreale e Buscemi [19] formalizzano solo crittografia simmetrica, ma in uno degli esempi da loro proposti esaminano il protocollo di Needham-Schroeder a chiave pubblica, trattando la crittografia asimmetrica nello stesso modo di quella simmetrica. A nostro avviso questo modo di procedere può causare un non corretto funzionamento dell'analisi, in quanto, durante lo sviluppo della nostra tesi, abbiamo osservato come la crittografia simmetrica e asimmetrica necessitano di assunzioni e formalizzazioni diverse. Nel più semplice modello basato su quello di Dolev-Yao, in grado di formalizzare crittografia simmetrica, si assume che tutte le chiavi condivise siano atomiche (costanti o variabili che possono essere istanziate solo con costanti). Questo semplifica molto la conoscenza dell'ambiente in quanto l'analisi di un termine risulta lineare sulla profondità della struttura del termine [73]. Per analizzare i protocolli utilizzati nel "mondo reale", però, spesso è necessario estendere il modello in modo che sia in grado di supportare chiavi condivise composte. In un tipico scenario di key exchange, due agenti, scambiandosi un segreto, derivano la chiave condivisa applicando la funzione hash alle parti del segreto condiviso insieme a nonce e ad altri dati (vedi il calcolo della master key nel SSL 3.0 handshake protocol [38]). Altro aspetto fondamentale per il confronto è dato dalla dimensione del modello ottenuto dall'analisi, che fornisce un parametro fondamentale per valutare l'efficienza del metodo.
La prima applicazione delle analisi simboliche spetta a Huima [53] che presenta una tecnica

decisionale per studiare la segretezza nei protocolli crittografici senza porre limiti alle operazioni degli attaccanti. Il metodo supporta crittografia sia simmetrica che asimmetrica e quindi la classe dei protocolli considerati è molto generale. I protocolli sono definiti utilizzando un formalismo basato sulle algebre di processo, simile allo spi-calculus non tipato. L'esecuzione di un protocollo genera un insieme di vincoli equazionali e la sua analisi consiste nel determinare l'esistenza di un'istanziazione delle variabili che soddisfa tutti i vincoli simultaneamente e quindi nel risolvere il problema dell'analisi della conoscenza. In [53] è fornita, però, solo una descrizione informale del sistema di riscrittura delle equazioni necessario per risolvere questi vincoli e, inoltre, manca una dimostrazione formale della correttezza e completezza del metodo.
Approcci più recenti basati su analisi simboliche sono descritti in [4, 5, 17, 18, 32]. Fiore e Abadi [32] basano il loro studio su [17], distinguendosi, però, per la fase di raffinamento del modello ottenuto mediante la semantica simbolica. Essi utilizzano una variante non tipata dello spi-calculus, che supporta crittografia a chiave simmetrica, e una free algebra. Il loro metodo analizza protocolli che utilizzano chiavi condivise composte, ma la completezza è provata solo quando le chiavi condivise sono atomiche. La semantica simbolica introdotta costruisce un grafo simbolico di computazione che simula il comportamento degli agenti onesti. I cammini del grafo rappresentano tutte le possibili esecuzioni del protocollo ed alcune di esse potrebbero violare le proprietà di sicurezza desiderate. Per determinare se esiste una traccia concreta corrispondente al cammino che non le soddisfa, viene fornito un algoritmo per decidere l'esistenza di un *realiser* per tutti i messaggi spediti dall'ambiente al processo. Un realiser è una sostituzione per variabili tale che ogni messaggio ricevuto è derivabile dalla conoscenza dell'ambiente. La nostra analisi si basa essenzialmente su questo metodo e il nostro apporto consiste nell'aver esteso il linguaggio in modo da poter analizzare protocolli che utilizzano algoritmi crittografici asimmetrici, e nell'aver dimostrato la completezza anche quando non sono posti vincoli alle chiave condivise.
Amadio e Lugiez [4, 5] si basano sull'analisi della raggiungibilità simbolica di Huima [53] e utilizzano un formalismo basato sulle algebre di processo non tipate simile allo spi-calculus per modellare i protocolli. Il loro metodo è in grado di supportare le funzioni hash, la crittografia asimmetrica e quella simmetrica, ma quest'ultima solo con il vincolo di considerare chiavi condivise atomiche. Similmente a Huima, la loro semantica genera vincoli equazionali, piuttosto che in forma di unifier, e, diversamente dal nostro approccio, l'esecuzione simbolica e il controllo della correttezza non sono tenute separate, avendo un impatto rilevante sulla dimensione del modello simbolico calcolato. Amadio e Lugiez codificano l'autenticazione attraverso la raggiungibilità, influendo così sulla complessità del loro metodo. Forniscono alcuni risultati sulla complessità del processo, e in particolare dimostrano che l'algoritmo decisionale è NP-hard. Analogamente a quando restringiamo le chiavi condivise a nomi, anche in questo approccio è necessario un metodo brute force per istanziare le variabili in posizione di chiave; tali variabili devono essere istanziate ad ogni possibile nome usato dai partecipanti, influendo così sulla dimensione del modello ottenuto. Questo inconveniente non è presente quando estendiamo l'analisi al caso in cui le chiavi condivise sono messaggi arbitrari, e nel metodo di Boreale [17, 18] il quale, però, propone un'analisi completa solo quando le chiavi condivise sono atomiche. La sua semantica simbolica è molto simile alla

nostra; in particolare, il messaggio ricevuto è rappresentato da una variabile libera i cui possibili valori sono vincolati durante l'esecuzione del protocollo. Il processo di raffinamento, invece, è diverso dal nostro e si basa sulla decomposizione dei termini simbolici in componenti irriducibili, e risulta non deterministico e quindi restituisce differenti forme risolte per la stessa traccia simbolica. Inoltre, in [17, 18] manca la motivazione per cui viene utilizzato tale procedimento rendendo così difficile capire il funzionamento dell'analisi. Boreale si differenzia dalla nostra analisi anche per aver unito la fase di verifica delle proprietà a quella della costruzione del modello adducendo motivi di efficienza in termini di memoria occupata. Noi abbiamo preferito mantenere separate queste due fasi, in quanto se, per qualche particolare applicazione, è necessario definire altre proprietà di sicurezza, queste sono verificate direttamente sul modello simbolico senza doverlo ricalcolare. In ogni caso, è sufficiente modificare la procedura Model per ottenere un metodo che si comporti nello stesso modo. In [20], Boreale e Buscemi propongono un algoritmo, per verificare se il protocollo rispetta le proprietà di sicurezza, in cui, calcolato il modello mediante la semantica simbolica, si verifica per ogni traccia generata se, ogniqualvolta occorre l'azione β nella traccia, l'azione α sia occorsa precedentemente nella traccia, dove l'ordine tra α e β è fissato dal protocollo. Ora, se si volesse verificare la proprietà di sicurezza, si dovrebbe calcolare nuovamente il modello. A nostro avviso la fase di unificazione, utilizzata per costruire il modello, è quella che risulta computazionalmente più gravosa e quindi conviene ripeterla il minor numero di volte. Per stabilire quale approccio sia più efficiente si dovrebbe confrontare la fase di raffinamento, ma Boreale non fornisce un algoritmo che descrive formalmente questa fase e il confronto risulta difficile da attuare. Come nella nostra analisi, quando il protocollo non soddisfa una proprietà, il metodo dà una soluzione per calcolare l'attacco, cioè una traccia che viola la proprietà.

Nuovi approcci simbolici sono presentati in [26] e [63] ed entrambi, a differenza del nostro metodo, non si basano sull'unificazione per costruire il modello simbolico. Comon, Cortier e Mitchell [26] adottano il modello di Dolev-Yao e definiscono un metodo in grado di analizzare protocolli che supportano crittografia asimmetrica e crittografia simmetrica con chiavi composte. La tecnica di decisione è ridotta ad un problema di insiemi di vincoli, a sua volta ridotto ad un problema di automata-theoretic, e l'algoritmo risultante impiega un tempo doppiamente esponenziale. La complessità è provata assumendo forti assunzioni sulla sintassi del protocollo. La prima è un rilassamento dei vincoli imposto al numero di sessioni, assumendo che solo un numero limitato di nuovi dati è generato in tutte le sessioni e ciò comporta un numero finito di sessioni o la mancanza nel protocollo di passi in cui sono generati nonce. Questo metodo, a differenza del nostro permette così di considerare sessioni multiple, ma per garantire la completezza e per stabilire la decidibilità del metodo si devono introdurre ulteriori restrizioni, ponendo dei limiti ai messaggi e non rappresentando così adeguatamene la realtà. La seconda assunzione stabilisce, infatti, che un agente può utilizzare solo una parte dei messaggi che riceve nei messaggi che spedisce. La tecnica proposta da Millen e Shmatikov [63] si focalizza sulle proprietà di raggiungibilità ed è basata sulla risoluzione dei vincoli. Ogni agente onesto del protocollo è specificato come un *semi-bundle* nel modello dello spazio dei ruoli del protocollo, dove per semi-bundle si intende un ruolo del protocollo parametrizzato con variabili. Utilizzare ruoli parametrizzati per rappresentare tracce simboliche di protocolli assume una chiara separazione tra il problema della riduzione

simbolica e quello dell'analisi della conoscenza. Quest'ultima è calcolata dalla procedura per risolvere il sistema di vincoli mediante una loro riduzione. Tale procedura, analogamente a quella da noi definita, termina, ed è corretta e completa perfino in presenza di chiavi composte. L'algoritmo così ottenuto risulta utile nella pratica in quanto può essere utilizzato per l'analisi di protocolli reali. Tuttavia, a nostro avviso, basarsi su proprietà di raggiungibilità per verificare la correttezza dei protocolli ha un impatto rilevante sulla complessità del metodo e rende difficile calcolare attacchi non conosciuti.

Capitolo 11

Conclusioni

Grande importanza, nell'ambito dei sistemi distribuiti, è data all'autenticazione tra le entità del sistema. Infatti, ogni partecipante esige l'assicurazione dell'identità delle altre parti. La risoluzione di questa problematica è fondamentale per il successo della sicurezza in un ambiente distribuito e la sua importanza è evidenziata, anche, dall'enorme attenzione che i problemi legati al processo di autenticazione hanno ricevuto in letteratura.
Molti sono stati i modelli presentati, legati ad impostazioni formali che si basano il più delle volte sulle algebre di processo. Questi approcci hanno il vantaggio di provare le specifiche di sicurezza richieste grazie all'utilizzo di strumenti di verifica dei modelli proposti, ma allo stesso tempo tendono ad allontanarsi dalle tecniche legate alla crittografia moderna. Impostazione diversa viene data dagli approcci legati alla teoria della complessità, che sono volti ad avvicinare le problematiche della sicurezza alle primitive basi della crittografia moderna. Sebbene queste metodologie abbiano un modello solido, hanno difficoltà a rendere la verifica delle specifiche semplice e pratica. Ogni approccio tenta di adeguare al proprio modello di sistema, definizioni appropriate di sicurezza e cerca di raggiungere una certa coerenza con il modello rappresentato. Tuttavia il risultato è spesso impreciso e la difficoltà di fornire adeguate definizioni è ben visibile in ogni approccio. Questa carenza è imputabile sia alla formalizzazione del modello, sia alla visione che ogni ricercatore ha del concetto di autenticazione.
Siamo giunti, perciò, alla conclusione che, proponendo definizioni adeguate, si possono risolvere problematiche diverse. Occorre, però, fare molta attenzione alle definizioni individuate e rapportarle correttamente agli intenti fissati e al modello di sistema considerato: la definizione di correttezza per protocolli, infatti, è legata agli scopi prefissati e al modello del sistema in analisi.

11.1 Risultati Ottenuti

L'obiettivo principale, che questo lavoro si è proposto, è definire un metodo automatico per verificare la correttezza di protocolli di autenticazione a stati infiniti, con l'intento di trovare eventuali errori nella progettazione dei protocolli stessi. Abbiamo affrontato perciò la verifica formale di protocolli di autenticazione, individuando i problemi che derivano e proponendo

una nuova tecnica formale per risolverli.
Abbiamo scelto di modellare i protocolli mediante le algebre di processo, in quanto ci è sembrato lo strumento più appropriato per ottenere modelli adatti ad una verifica automatica delle proprietà di sicurezza. In questo modo, il comportamento di un agente è descritto da un processo sequenziale, i protocolli sono ottenuti dalla composizione parallela di tali processi, mentre gli attaccanti sono modellati implicitamente nel protocollo.
La verifica automatica della correttezza di un protocollo, però, non può essere eseguita studiando le tracce di computazione ottenute attraverso la semantica classica delle algebre di processo. Infatti, poiché i messaggi conosciuti dall'ambiente sono infiniti, il modello ottenuto è infinito.
Attraverso una tecnica simbolica che opera in due fasi, abbiamo definito un modello finito in grado di rappresentare esattamente tutte le esecuzioni computazionalmente valide del protocollo. Prima abbiamo definito una riduzione simbolica di processi in cui gli input sono valutati formalmente, cioè viene introdotta una variabile nuova ad ogni azione di input; tale riduzione è anche in grado di calcolare i vincoli per cui il sistema si evolve. Il risultato non è ancora sufficiente: i modelli ottenuti sono si finiti, ma non tutti i suoi elementi rappresentano una storia valida del sistema in analisi. I processi, infatti, potrebbero ricevere messaggi di cui l'ambiente non ha conoscenza. Abbiamo risolto il problema definendo una procedura simbolica che analizza la conoscenza dell'ambiente, cioè un algoritmo in grado di stabilire quali messaggi possono essere dedotti a partire dallo stato di conoscenza dell'ambiente e dalla sua conoscenza iniziale. Abbiamo ottenuto così uno strumento per raffinare il modello astratto eliminando i controesempi spuri, cioè quei comportamenti che non hanno una controparte nel modello concreto. Combinando la riduzione simbolica con la procedura per l'analisi della conoscenza, abbiamo costruito modelli simbolici finiti, che rappresentano esattamente le esecuzioni simboliche computazionalmente valide del protocollo, e che sono adatti per una verifica automatica.
Altro aspetto fondamentale per definire una verifica della correttezza di protocolli è formalizzare la nozione di sicurezza per protocolli di autenticazione. Tali protocolli sono progettati per garantire l'identità degli agenti e lo scambio di nuove chiavi segrete di sessione per future comunicazioni. Abbiamo formalizzato le nozioni di autenticazione e di scambio di chiavi, utilizzando quelle proposte da Woo e Lam [96]. Questi hanno individuato due proprietà primitive: correspondence, che afferma che la comunicazione deve avvenire seguendo i passi fissati dal protocollo, e secrecy, che esprime come le informazioni riservate non devono essere deducibili dalla conoscenza dell'ambiente. Su queste due proprietà abbiamo basato le definizioni di correttezza adeguate al nostro modello.
La tecnica così definita è in grado di modellare protocolli che utilizzano la codifica basata su algoritmi crittografici sia simmetrici che asimmetrici e, a differenza di altri autori [4, 17] che si limitano a trattare il caso in cui le chiavi condivise sono limitate a nomi, è in grado di analizzare il caso in cui le chiavi condivise sono qualsiasi messaggio. Questa generalità risulta utile per modellare protocolli in cui le chiavi condivise possono essere qualsiasi messaggio e quindi la nostra tecnica produce risultati decidibili per una classe interessante di protocolli.
In questo progetto è stato sviluppato anche un prototipo che ha permesso di calcolare i modelli dei protocolli visti. Tale prototipo è stato implementato in ML [75, 93, 95].

11.2 Ricerche Future

Si possono intraprendere varie strade per estendere i risultati ottenuti in questo lavoro. Alcuni spunti per ricerche future sono i seguenti:

1. Estendere la grammatica considerando altri tipi di primitive crittografiche, come la firma a chiave privata.

2. Estendere l'analisi per verificare processi in cui è presente la replicazione e la restrizione, consentendo, per esempio, di studiare automaticamente sessioni multiple.

3. Formalizzare altre proprietà di sicurezza (oltre a corrispondenza e segretezza) per studiare altre tipologie di protocolli.

4. Rendere più efficienti gli algoritmi per l'analisi della conoscenza, utilizzando insiemi atomici di messaggi.

È nostra opinione che l'approccio proposto per l'analisi della conoscenza è abbastanza generale per affrontare altre primitive crittografiche.
La necessità di estendere l'analisi per verificare processi infiniti è evidente negli esempi presentati nel Capitolo 10. Tale estensione è necessaria per una simulazione automatica di sessioni multiple sia sequenziali che simultanee. Nel protocollo proposto da Woo e Lam, per esempio, non saremo costretti ad assumere che l'ambiente conosca il nome dei partecipanti alle comunicazioni, ma basterà eseguire più volte il protocollo attraverso la replicazione per scoprire con facilità una serie di attacchi che possono essere portati ai protocolli. Ora, per riconoscere attacchi, che si basano su sessioni multiple, bisogna conoscere a priori il numero di sessioni necessario per rompere il protocollo.
Nella presentazione della tesi abbiamo osservato che l'utilizzo di insiemi atomici di messaggi ci consente di raggiungere l'obiettivo senza dover applicare le regole di eliminazione dei sistemi deduttivi per l'analisi della conoscenza. È possibile quindi riscrivere il sistema deduttivo di Tabella 6.1 senza le regole suddette. Inoltre, non è più necessaria l'introduzione dei sistemi che utilizzano i sottotermini e i sottotermini attivi in quanto le derivazioni ottenute con le regole di inferenza riscritte sono semplici. A questo punto risulta evidente che anche il sistema deduttivo di Tabella 7.3 potrà essere riscritto tralasciando le regole di eliminazione. Quindi la fase di raffinamento potrà essere resa ancora più efficiente ridefinendo la procedura per l'analisi della conoscenza che si basa su tale sistema deduttivo.

Appendice A

Codice ML

```
type
   token_type = string;
type
   name_type  = string;
type
   key_type   = string;

fun
    isIn x nil = false
  | isIn x (a::t) = (x=a) orelse (isIn x t);

fun
    union nil l = l
  | union (a::t) l
     = let val l1 = union t l
       in if isIn a l
          then l1
          else a::l1
       end;

fun
    Union nil = nil
  | Union (a::t) = union a (Union t);

fun
    ForUnion l f = Union (map f l);

fun
    lista([],f)   = []
  | lista(x::b,f) = if (isIn x f)
                    then lista(b,f)
                    else x::lista(b,f);

fun
    ast_msg (skencr(M,N)) = (skencr(M,N))::(ast_msg M)
  | ast_msg (pkencr(M,N)) = (pkencr(M,N))::(ast_msg M)
  | ast_msg (p(M,N)) = (p(M,N))::(union (ast_msg M) (ast_msg N))
  | ast_msg _ = [];

fun
    ast K = ForUnion K ast_msg;
```

A.1 Codice ML per il controllo della conoscenza

```
datatype
   message = name of name_type
           | token of token_type
           | pukey of key_type
           | prkey of key_type
           | p of message * message
           | skencr of message * message
           | pkencr of message * key_type
           | h of message;
```

A.1.1 Codice ML della procedura Check

```
fun
  check (k,m,f) = not(isIn m f) andalso
                  ( analyse(k,m,f) orelse synthetise(k,m,f) )

  and

  analyse(k,m,f) =
             let val f1 = m::f
             in case m of
                    p(m1,m2)      => check(k,m1,f1) andalso check(k,m2,f1)
                  | skencr(m1,m2) => check(k,m1,f1) andalso check(k,m2,f1)
                  | pkencr(m1,m2) => check(k,m1,f1)
                  | h(m1)         => check(k,m1,f1)
                  |     _         => false
              end

  and

  synthetise(k,m,f) =
              case m of
                 token(t)  => true
               | pukey(pu) => true
               | _         =>  (isIn m k)
                               orelse
                                let val f1 = m::f
                                in let val a = ast(k)
                                   in let val b = lista(a,f1)
                                      in  proj(k,f1,m,b)
                                          orelse
                                           skdec(k,f1,m,b)
                                          orelse
                                           pkdec(k,f1,m,b)
                                       end
                                   end
                                end

  and
```

```
    proj(k,f,m,[])   = false
  | proj(k,f,m,n::b) =
                case n of
                  p(m1,m2) => if  m = m1 orelse m = m2
                              then  synthetise(k,p(m1,m2),f)
                                   orelse
                                    proj(k,f,m,b)
                               else proj(k,f,m,b)
                | _        => proj(k,f,m,b)

 and

    skdec(k,f,m,[])   = false
  | skdec(k,f,m,n::b) =
                case n of
                    encr(m1,m2) => if m = m1
                                   then  ( synthetise(k,skencr(m1,m2),f)
                                          andalso
                                           check(k,m2,f) )
                                         orelse skdec(k,f,m,b)
                                    else skdec(k,f,m,b)
                  | _           => skdec(k,f,m,b)

and

    pkdec(k,f,m,[])   = false
  | pkdec(k,f,m,n::b) =
                case n of
                   pkencr(m1,m2) => if m = m1
                                    then ( synthetise(k,pkencr(m1,m2),f)
                                          andalso
                                           check(k,prkey(m2),f) )
                                         orelse pkdec(k,f,m,b)
                                    else pkdec(k,f,m,b)
                 | _             => pkdec(k,f,m,b);
```

A.1.2 Codice ML della procedura Enumerate

```
datatype
   prooftree = tok of token_type
             | pub of key_type
             | ax of message
             | pair of prooftree * prooftree
             | projl of prooftree
             | projr of prooftree
             | skenc  of prooftree * prooftree
             | skdec of prooftree * prooftree
             | pkenc  of prooftree * key_type
             | pkdec of prooftree * prooftree
             | hash of prooftree;
```

```
fun
    enumerate(k,m,f) = if (isIn m f)
                       then []
                       else ( analyse(k,m,f) @ synthetise(k,m,f) )

    and

    analyse(k,m,f) =
            let val f1 = m::f
            in  case m of
                 p(m1,m2)      => let val a = enumerate(k,m1,f1)
                                  in let val b = enumerate(k,m2,f1)
                                     in if (a = [] orelse b = [])
                                        then []
                                        else ForUnion
                                               a
                                               ( fn R1 => ForUnion
                                                            b
                                                            ( fn R2 => [ pair(R1,R2) ] )
                                               )
                                     end
                                  end
               | skencr(m1,m2) => let val a = enumerate(k,m1,f1)
                                  in let val b = enumerate(k,m2,f1)
                                     in if (a = [] orelse b = [])
                                        then []
                                         else ForUnion
                                               a
                                               ( fn R1 => ForUnion
                                                            b
                                                            ( fn R2 => [ skenc(R1,R2) ] )
                                               )
                                     end
                                  end
               | pkencr(m1,pu) => let val a = enumerate(k,m1,f1)
                                  in if a = []
                                     then []
                                     else ForUnion
                                            a
                                            ( fn R => [ pkenc(R,pu) ] )
                                  end
               | h(m1)         => let val a = enumerate(k,m1,f1)
                                  in if a = []
                                     then []
                                     else ForUnion
                                            a
                                            ( fn R => [ hash(R) ] )
                                   end
               |  _            => []
            end


    and
```

```
synthetise(k,m,f) = synthetisebase(k,m,f) @ synthetiseproj(k,m,f)
                    @ synthetiseskdec(k,m,f) @ synthetisepkdec(k,m,f)

and

synthetisebase(k,m,f) = case m of
                            token(t) => [tok(t)]
                          | pukey(pu) => [pub(pu)]
                          |   _  => if (isIn m k)
                                    then [ax(m)]
                                    else []

and

synthetiseproj(k,m,f) =
                       let val f1 = m::f
                       in let val a = ast(k)
                          in let val b = lista(a,f1)
                             in if b = []
                                then []
                                else proj(k,m,f1,b)
                             end
                          end
                       end

and

  proj(k,m,f,[])   = []
| proj(k,m,f,n::b) =
              case n of
                 p(m1,m2) => if  m = m1
                             then let val a = synthetise(k,p(m1,m2),f)
                                  in if a = []
                                     then proj(k,m,f,b)
                                     else ForUnion
                                           a
                                           ( fn R => [ projl(R) ] )
                                  end
                               else if  m = m2
                                    then let val a = synthetise(k,p(m1,m2),f)
                                         in if a = []
                                            then proj(k,m,f,b)
                                            else ForUnion
                                                  a
                                                  ( fn R => [ projr(R) ] )
                                         end
                                    else proj(k,m,f,b)
               | _        => proj(k,m,f,b)

and

synthetiseskdec(k,m,f) =
                      let val f1 = m::f
                      in let val a = ast(k)
```

```
                          in let val b = lista(a,f1)
                             in if b = []
                                  then []
                                  else skdecr(k,m,f1,b)
                             end
                          end
                       end

   and

     skdecr(k,m,f,[])   = []
   | skdecr(k,m,f,n::b) =
         case n of
           skencr(m1,m2) => if  m = m1
                            then let val a = synthetise(k,skencr(m1,m2),f)
                                 in let val c = enumerate(k,m2,f)
                                    in if a = [] orelse c = []
                                       then skdecr(k,m,f,b)
                                       else ForUnion
                                              a
                                              ( fn R1 => ForUnion
                                                           c
                                                           ( fn R2 => [ skdec(R1,R2) ] ) )
                                    end
                                 end
                            else skdecr(k,m,f,b)
           | _           => skdecr(k,m,f,b)

and

   synthetisepkdec(k,m,f) =
                       let val f1 = m::f
                       in let val a = ast(k)
                          in let val b = lista(a,f1)
                             in if b = []
                                   then []
                                   else pkdecr(k,m,f1,b)
                             end
                          end
                       end

and

   pkdecr(k,m,f,[])   = []
 | pkdecr(k,m,f,n::b) =
         case n of
           pkencr(m1,m2) => if  m = m1
                            then let val a = synthetise(k,pkencr(m1,m2),f)
                                 in let val c = enumerate(k,prkey(m2),f)
                                    in if a = [] orelse c = []
                                       then pkdecr(k,m,f,b)
                                       else ForUnion
                                              a
                                              ( fn R1 => ForUnion
```

```
                                                          c
                                                          ( fn R2 => [ pkdec(R1,R2) ] ) )
                                    end
                                end
                            else pkdecr(k,m,f,b)
          | _            => pkdecr(k,m,f,b);
```

A.2 Codice ML per l'analisi della conoscenza

```
type
   var_type   = string;

datatype
   message = name of name_type
           | token of token_type
           | var of var_type
           | pukey of key_type
           | prkey of key_type
           | p of message * message
           | skencr of message * message
           | pkencr of message * key_type
           | h of message;

fun
    isSubterm M N
        =   (M=N)
        orelse
          case N of
            skencr(N1,N2) => (isSubterm M N1) orelse (isSubterm M N2)
          | pkencr(N1,N2) => (isSubterm M N1)
          | p(N1,N2)    => (isSubterm M N1) orelse (isSubterm M N2)
          | h(N1)       => (isSubterm M N1)
          | _ => false ;

exception ERROR;

fun
    tvSubst M (var x) N
      = ( case M of
            var y       => if x=y then N else (var y)
          | name a      => name a
          | token t     => token t
          | pukey pu    => pukey pu
          | prkey pu    => prkey pu
          | skencr(M1,M2) => skencr(tvSubst M1 (var x) N, tvSubst M2 (var x) N)
          | pkencr(M1,M2) => pkencr(tvSubst M1 (var x) N, M2)
          | p(M1,M2)    => p(tvSubst M1 (var x) N, tvSubst M2 (var x) N)
          | h(M1)       => h(tvSubst M1 (var x) N) )

  | tvSubst M _ N
        = raise ERROR ;
```

```
fun
    EvSubst E x M
        = map (fn (N1,N2) => (tvSubst N1 x M,tvSubst N2 x M)) E ;

exception NoMGU ;

fun
    mgu nil  = nil

  | mgu ( (var x, N)::E )
        = if N=(var x)
          then mgu E
          else if (isSubterm (var x) N)
               then raise NoMGU
               else union [(var x,N)] ( mgu(EvSubst E (var x) N) )

  | mgu ( (name a, N)::E )
        = ( case N of
            var x  => union [(var x,name a)] ( mgu(EvSubst E (var x) (name a)) )
          | name b => if a=b then mgu E else raise NoMGU
          | _      => raise NoMGU )

  | mgu ( (token t, N)::E )
        = ( case N of
            var x   => union [(var x,token t)] ( mgu(EvSubst E (var x) (token t)) )
          | token s => if t=s then mgu E else raise NoMGU
          | _       => raise NoMGU )

  | mgu ( (pukey pu, N)::E )
        = ( case N of
            var x   => union [(var x,pukey pu)] ( mgu(EvSubst E (var x) (pukey pu)) )
          | pukey s => if pu=s then mgu E else raise NoMGU
          | _ => raise NoMGU )

  | mgu ( (prkey pu, N)::E )
        = ( case N of
            var x   => union [(var x,prkey pu)] ( mgu(EvSubst E (var x) (prkey pu)) )
          | prkey s => if pu=s then mgu E else raise NoMGU
          | _ => raise NoMGU )

  | mgu ( (skencr(M1,M2), N)::E )
        = ( case N of
            var x         => if (isSubterm (var x) M1) orelse (isSubterm (var x) M2)
                             then raise NoMGU
                             else union
                                  [(var x,skencr(M1,M2))]
                                  ( mgu(EvSubst E (var x) (skencr(M1,M2))) )
          | skencr(N1,N2) => mgu( union [(M1,N1),(M2,N2)] E )
          | _             => raise NoMGU )

  | mgu ( (pkencr(M1,M2), N)::E )
        = ( case N of
              var x         => if (isSubterm (var x) M1)
                               then raise NoMGU
```

```
                              else union
                                   [(var x,h(M1))]
                                   ( mgu(EvSubst E (var x) (pkencr(M1,M2))) )
          | pkencr(N1,M2) => mgu( union [(M1,N1)] E )
          | _             => raise NoMGU )

  | mgu ( (p(M1,M2), N)::E )
       = ( case N of
           var x    => if (isSubterm (var x) M1) orelse (isSubterm (var x) M2)
                       then raise NoMGU
                       else union
                             [(var x,p(M1,M2))]
                             ( mgu(EvSubst E (var x) (p(M1,M2))) )
         | p(N1,N2) => mgu( union [(M1,N1),(M2,N2)] E )
         | _        => raise NoMGU )

   | mgu ( (h(M1), N)::E )
       = ( case N of
           var x => if (isSubterm (var x) M1)
                    then raise NoMGU
                    else union
                           [(var x,h(M1))]
                           ( mgu(EvSubst E (var x) (h(M1))) )
         | h(N1) => mgu( union [(M1,N1)] E )
         | _     => raise NoMGU ) ;

fun
    TvSubst K x N = map (fn M => tvSubst M x N) K ;

fun
    tVSubst M nil = M
  | tVSubst M ((x,N)::R) = tVSubst (tvSubst M x N) R ;

fun
    TVSubst K nil = K
  | TVSubst K ((x,M)::R) = TVSubst (TvSubst K x M) R ;
```

A.2.1 Codice ML della procedura Realise

```
fun
    Realise K M F
       = if (isIn M F)
         then nil
         else union
               (Analyse K M F)
               (Synthetise K M F)

 and

    Analyse K M F
      = let val F' = union [M] F
        in case M of
           p(M1,M2)      => ForUnion
                            ( Realise K M1 F' )
```

```
                              ( fn R1 => ForUnion
                                         ( Realise
                                           ( TVSubst K R1 )
                                           ( tVSubst M2 R1)
                                           ( TVSubst F' R1) )
                                         ( fn R2 => [ R1 @ R2 ] ) )
          | skencr(M1,M2) => ForUnion
                              ( Realise K M1 F' )
                              ( fn R1 => ForUnion
                                         ( Realise
                                           ( TVSubst K R1 )
                                           ( tVSubst M2 R1)
                                           ( TVSubst F' R1) )
                                         ( fn R2 => [ R1 @ R2 ] ) )
          | pkencr(M1,M2) => Realise  K M1 F'
          | h(M1)         => Realise  K M1 F'
          | _ => nil
    end

and

    Synthetise K M F
     = case M of
         var x    => if K=[] then nil else [ [] ]
       | token t  => [ [] ]
       | pukey pu => [ [] ]
       | _        => union
                     ( ForUnion K ( fn M' => [ mgu [(M,M')] ] handle NoMGU => nil ) )
                     ( let val F' = union [ M ] F
                       in ForUnion
                          ( AST K )
                          ( fn N => if (isIn N F')
                                    then [ nil ]
                                    else case N of
                                         p(N1,N2) => union
                                                     ( let val R = mgu [(M,N1)]
                                                       in ForUnion
                                                          ( Synthetise
                                                            ( TVSubst K R )
                                                            ( p( tVSubst M R , tVSubst N2 R ) )
                                                            ( TVSubst F' R ) )
                                                        ( fn R' => [ R@R' ] )
                                                       end handle NoMGU => nil )
                                                     ( let val R = mgu [(M,N2)]
                                                       in ForUnion
                                                          ( Synthetise
                                                            ( TVSubst K R )
                                                            ( tVSubst N R )
                                                            ( TVSubst F' R ) )
                                                          ( fn R' => [ R@R' ] )
                                                        end handle NoMGU => nil )
                                       | skencr(M',N') => ( let val R = mgu [ (M,M') ]
                                                            in ForUnion
                                                               ( Synthetise
```

```
                                                    ( TVSubst K R )
                                                    ( tVSubst N R )
                                                    ( TVSubst F' R ) )
                                                  ( fn R1 => let val R' = R@R1
                                                             in ForUnion
                                                               ( Realise
                                                                 ( TVSubst K R' )
                                                                 ( tVSubst N' R' )
                                                                 ( TVSubst F' R' ) )
                                                               ( fn R2 => [ R'@R2 ] )
                                                             end )
                                               end handle NoMGU => nil )
                                | pkencr(M',N') => ( let val R = mgu [ (M,M') ]
                                                     in ForUnion
                                                        ( Synthetise
                                                          ( TVSubst K R )
                                                          ( tVSubst N R )
                                                          ( TVSubst F' R ) )
                                                        ( fn R1 => let val R' = R@R1
                                                                   in ForUnion
                                                                      ( Realise
                                                                        ( TVSubst K R' )
                                                                        ( prkey N' )
                                                                        ( TVSubst F' R' ) )
                                                                      ( fn R2 => [ R'@R2 ] )
                                                                   end )
                                                     end handle NoMGU => nil )
                                   | _           => nil )
                        end ) ;
```

A.2.2 Codice ML della procedura Constraints

```
fun Constraints K M
  = ForUnion
      ( Realise K M nil )
      ( fn R => if R = []
                then [ [] ]
                else ( ForUnion
                         ( Constraints (TVSubst K R) (tVSubst M R) )
                         ( fn R' => [ R @ R' ] ) ) ) ;
```

A.3 Codice ML per i modelli simbolici

```
datatype

  action = output of message
         | input of message ;

fun
    TraceSubst t R =
        map
          ( fn a => case a of
```

```
                    output M => output(tVSubst M R)
                  | input M  => input(tVSubst M R) )
          t ;
```

A.3.1 Codice ML della procedura Model

```
fun
   Model s =
     let fun
            Model1 t s =
                case s of
                   nil   => [ [] ]
                 | a::s' => ( case a of
                                output M => Model1 ( t@[a] ) s'
                              | input M  => ForUnion
                                            ( Constraints
                                              ( ForUnion
                                                 t
                                                 ( fn a => case a of
                                                           output a' => [ a' ]
                                                         | _         => nil ) )
                                              M )
                                            ( fn R => if R = []
                                                      then Model1 (t@[a]) s'
                                                      else ForUnion
                                                           ( Model1 nil (TraceSubst (t@s) R) )
                                                           ( fn R' => [ R@R' ] ) ) )
     in Model1 nil s
     end ;
```

Bibliografia

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security.* ACM Press, 1997. Extended version available as Technical Report 414, University of Cambridge Computer Laboratory.

[2] M. Abadi and R. Needham. Prudent engineering practice for cryptography protocols. Tecnical Report RR 125, Digital Systems Research Center, June 1994.

[3] R. Amadio, W. Charatonik. On name generation and set-based analysis in Dolev-Yao model Tecnical Report RR 4379, INRIA, January 2002.

[4] R. Amadio and D. Lugiez. On the Reachability Problem in Cryptographic Protocols. In *Proc. CONCUR 2000*, volume 1877 of *Lecture Notes in Computer Science*, pages 380-394. Springer-Verlag, 2000.

[5] R. Amadio, D. Lugiez and V. Vanackere. On The Symbolic Reduction of Processes with Cryptographic Functions. Technical Report RR 4147, INRIA, Sophia-Antipolis, 2001. Under Revision for *Theoretical Computer Science*. Abstract appared in Proc. Logical Aspects of Cryptographic Protocol Verification, Electronic Notes in Theoretical computer Science, 55(1), 2001.

[6] J. P. Banatre and D. LeMetayer. Gamma and the Chemical Reaction Model: Ten Years After. In JM. Andreoli, C. Hankin and D. LeMetayer editors, *Coordination Programming: Mechanisms, Models and Semantics*, pages 3-41. Impirial College Press, 1996.

[7] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In Douglas R. Stinson Editor, Proc. CRYPTO 93, pages 232-249. Springer, 1994. Lecture Notes in Computer Science N. 773.

[8] M. Bellare and P. Rogaway. Provably secure session key distribution - the three party case. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 57-66, LAs Vegas, Nevada, 1995.

[9] M. Berger, K. Honda, N. Yoshida. Sequentiality and the π-calculus. TLCA01, LNCS 2044, pages 29-45, Springer, 2001.

[10] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Mlva, and M. Yung. Systematic Design of Two-Party Authentication Protocols. In J. Feigenbaum Editor, *Proc. CRYPTO91*, pages 44-61. Springer, 1992. Lecture Notes in Computer Science N. 576.

[11] M. Blum and S. Micali. Sequences of pseudo-random bits. *SIAM J. Computing*, 13(4):850-863, November 1984.

[12] C.Bodei. Security Issues in Process Calculi. Computer Science Department. University of Pisa. TD-2/00, March 2000.

[13] C. Bodei, P. Degano, F. Nielson and H. Riis Nielson. Control Flow Analysis for the π-calculus. In *Proceeding of CONCUR'98,* volume 1466 of *LNCS* , pages 84-98. Spring-Verlag, 1998.

[14] C. Bodei, P. Degano, F. Nielson and H. Riis Nielson. Security Analysis using Flow Logics. Current Trends in Theoretical Computer Science, pages 525-542, 2001.

[15] D. Bolignano. An approach to the Formal Verification of Cryptographic Protocol. In Clifford Neuman Editor, *3rd ACM Conference on Computer and Communications Security*, pages 106-118, New Delhi, India, March 1996.

[16] D. Bolignano. Towards the Formal Verification of Electronic Commerce Protocols. In PCSFW: Proceedings of the 10th Computer Security Foundations Workshop. IEEE Computer Society Press, 1997.

[17] M. Boreale Symbolic analysis of criptographic protocols in the spi-calculus. Manuscript available from http://dsiII.dsi.unifi.it/ boreale/papers.html, 2000.

[18] M. Boreale. Symbolic trace analysis of cryptographic protocols. Accepted to ICALP'01, March 2001.

[19] M. Boreale, M. Buscemi. Experimenting with STA, a Tool for Automatic Analysis of Security Protocols. A shorter version appears in *Proc. of SAC '02*, ACM Press, 2002.

[20] M. Boreale, M. Buscemi. A Framework for the Analysis of Security Protocol. To appear in *Proc. of CONCUR '02*.

[21] P. J. Broadfoot. Data indepence in the model checking of security protocols. PhD Thesis. University of Oxford. 2001.

[22] M. Burrows, M. Abadi, R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18-36, 1990.

[23] A. Cavalca. Tecnica dei Vincoli Negativi: un Nuovo Metodo di Analisi per Protocolli di Autenticazione. Tesi di Laurea. Università degli Studi di Bologna, Anno Accademico 1996-'97.

[24] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstract Refinement. *Computer Aided Verification (CAV)*, July 2000.

[25] E. Clarke, S. Jha and W. Marreno. Using State Space Exploration and a Natural Deduction Style Message Derivation Engine to Verify Security Protocols. In *Proc. IFIP Working Conference on Programming Comcepts and Methods (PROCOMET)*, 1998.

[26] M. Comon, V. Cortier and J. Mitchel. Tree automata with one memory, set constraints, and ping-pong protocols. In *Proc. ICALP*, pages 682-693. Springer Lecture Notes in Computer Science 2076, 2001.

[27] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest. Introduction to Algorithms. MIT Press, 2001.

[28] R. De Nicola and M. Henessy. Testing equivalences for processes. Theoretical Computer Science, 34:83-133, 1984.

[29] W. Diffie, M. Hellman. New Directions in Cryptography. In *IEEE Transactions on Information Theory*, val 22, pages 644-654, 1976.

[30] W. Diffie, P. C. Van Oorschot and M. J. Weiner. Authentication and Authenticated Key Exchange. *Designs, Codes, and Cryptography*, 2(2):107-125, June 1992.

[31] D. Dolev and A. Yao. On The Security of Public Key Protocols. *Transactions on Information Theory*, 29(2):198-208, 1983.

[32] M. Fiore, M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In 14th *Computer Security Foundations Workshop*. IEEE Computer Society Press, 2001.

[33] R. Focardi, R. Gorrieri. The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties. Tecnichal Report UBLCS-96-14, August 1996.

[34] R. Focardi, A. Ghelli and R. Gorrieri. Using Non Interference for the Analysis of Security Protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols* (H. Orman and C. Meadows Ed.) September 3-5, 1997, DIMACS Center, CoRE Building, Rutgers University.

[35] R. Focardi and F. Martinelli. A Uniform Approach for the Definition of Security Properties. In *proceedings of World Congress on Formal Methods (FM'99)*, Springer-Verlag LNCS 1708, pag. 794-813, Toulouse (France), September 1999.

[36] R. Focardi, R. Gorrieri and F. Martinelli. Message Authentication through Non Interference. In *proceedings of International Conference on Algebraic Methodology And Software Technology (AMAST 2000)*, LNCS 1816, pag. 258-272, Iowa City, Iowa, USA, May 2000.

[37] R. Focardi, R. Gorrieri and F. Martinelli. Non Interference for the Analysis of Cryptographic Protocols. In *proceedings of 27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, (U. Montanari, ed.), LNCS 1853, Geneve, Switzerland, July 2000.

[38] A. Freier, P. Karlton and P. Kocher. The SSL Protocol. Version 3.0. `http://home.netscape.com/eng/ssl3/`

[39] T. Genet and F. Klay. Rewriting for Cryptographic Protocols Verification. In *Proc. CADE, Springer Lecture Notes in Computer Science* 1831, 2000.

[40] R. Giacobazzi and E. Quintarelli. Incompleteness, Counterexamples and Refinements in Abstract Model-Checking . In P. Cousot Ed. *The 8th International Static Analysis Symposium SAS'01*, volume 2126 of *Lecture Notes in Computer Science*, pages 356-373, Springer-Verlag. La Sorbonne, Paris, 16-18 July, 2001.

[41] P. Gianbiagi. Security for Mobile Implementations of Security Protocols. SICS Tecnichal Report T2001:19, October 2001.

[42] O. Goldreich, S. Goldwasser and S. Micali. How to Costruct Random Functions. *Journal of the ACM*, 33(4):792-807, October 1984.

[43] S. Goldwasser, S. Micali. Probabilistic Encryption. *JCSS*, 28(2):270-299, April 1984.

[44] L. Gong, R. Needham, and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. In Symposium on Research in Security and Privacy, pages 234-248. IEEE Computer Society Press, May 1990.

[45] J. Goubault. A Metod for Automatic Cryptografic Protocols Verification In *Proc. FMPPTA, Springer-Verlag*, 2000.

[46] Joshua D. Guttman, F. Javier Thayer Fábrega. Authentication Tests. In *To appear in Security and Privacy Symposium'00*, 2000.

[47] Joshua D. Guttman. Security Goals: Packet Trajectories and Strand Spaces. In R. Gorrieri and R. Focardi, editors, *Foundations of Security Analysis and Design*, volume 2171 of *LNCS* Springer Verlag, 2001.

[48] M. Hennessy. Algebraic Theory of Processes. MIT Press, 1988.

[49] T. Hoare, C.A.R., Communicating Sequential Processes. Prentice Hall, 1985.

[50] K. Honda, V. Vasconcelos and N. Yoshisa. Secure Information Flow as Typed Process Behaviour. ESOP'00, LNCS 1782, pages 180-199, 2000.

[51] K. Honda and N. Yoshida. A uniform type structure for secure information flow. In *Symposium on Principles of Programming Languages*, pages 81-92, 2002.

[52] J. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Language, and Computation*. Addison-Wesley, Reading, Massachusett, 1979.

[53] A. Huima. *Analysis of cryptographic protocols via symbolic state space enumeration.* Research Report A54, Laboratory for Theoretical Computer Science, Helsinki University of Technology, August 1999.

[54] M. Hyland and L. Ong. On Full Abstraction for PCF: I, II and III. In *Information and Computation*, Volume 163, pages 285-408, December 2000.

[55] R. A. Kemmerer. Analyzing Encryption Protocols Using Formal Tecniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448-457, May 1989.

[56] G. Lowe. An Attack on the Needham-Schoeder Public Key Authentication Protocol. *Information Processing Letters*, 56(3):131-133, November 1995.

[57] G. Lowe. Breaking and Fixing the Needham Schroeder Public Key Protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147-166. Springer Verlag 1995.

[58] G. Lowe. A Hierarchy of Authentication Specifications. Technical Report 1996/33, Department of Mathematics and Computer Science, 1996.

[59] G. Lowe. Towards a Completeness Result for Model Checking of Security Protocols. *Journal of Computer Security,* Volume 7, Numbers 2, 3, pages 89-146, 1999. Earlier versions appeared in Proceedings of *11th IEEE Computer Security Foundations Workshop*, pages 96-105, 1998, and as Department of Mathematics and Computer Science, Technical Report 1998/6, University of Leicester, 1998.

[60] W. Mao and C. Boyd. Towards the Formal Analysis of Security Foundations Workshop VI, pages 147-158. IEEE Computer Society Press, 1993.

[61] C. Meadows. A System for the Specification and Verification of Key Management Protocols. In *1991 IEEE Computer Symposium on Research in Security e Privacy*, pages 182-195, May 1991.

[62] J. K. Millen, S. C. Clark and S. B. Freedman. The Interrogator: Protocol Security Analysis. *IEEE Transictions on Software Engineering*, SE-13(2):274-288, February 1987.

[63] J. K. Millen, V. Shmatikov. Constraint Solving for Bounded-process Cryptographic Protocol Analysis. In *Proc. of8th ACM Conference on Computer and Communication Security*, ACM Press, 2001.

[64] R. Milner. Communication and Concurrency. Prentice Hall, 1989.

[65] R. Milner. The polyadic π-calculus: a tutorial. University of Edinburgh, ECS-LFCS-91-180, 1991.

[66] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I and part II. *Information and Computation,* vol. 100, pp. 1-40 and pp. 41-77, September 1992.

[67] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *TCS*, 114:149:171, 1993.

[68] D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *Proc. Stastic Analysis Symposium, Springer Lect. Notes in Computer Science*, 1999.

[69] C. Myers, C. Clack, E. Poon. Programming with Standard ML. Prentice Hall, 1993.

[70] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Network of Computers. Communications of the ACM, 21(12):993-999, December 1978.

[71] D. M. Nessett. A Critique of the Burrows, Abadi and Needham Logic. In *Operating System Review*, volume 2, pages 35-38, April 1990.

[72] J. Parrow. An Introducion to the Pi-calculus. Handbook of Process Algebra, Bergstra and Ponse and Smolka, Elsevier, pages 479-543, 2001.

[73] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. In *Journal of Computer Security*, 6:85-128, 1998.

[74] L. C. Paulson. Foundations of Computer Science. Computer Laboratory, University of Cambridge, 2001.

[75] F. Pottier and V. Simonet. Information Flow Inference for ML. *Symposium on Principles of Programming Languages*, pages 319-330, 2002.

[76] A. John Power, Hayo Thielecke. Environments, Continuation Semantics and Indexed Categories. TACS, pages 391-414, 1997.

[77] C. Priami Enhanced Operational Semantics for Concurrency. PhD Thesis. TD-08/96, Dipartimento di Informatica, University of Pisa. 1996.

[78] U. Quintarelli e N. Zannone. Appunti tratti dal corso di Sicurezza e Crittografia. Università di Verona, Anno Accademico 2001-02.

[79] W. Reising. Petri Nets. *EATCS Monographs on Theoretical Computer Science*, Spinger Verlang, 1983.

[80] M. Roveri. *Meccanizzazione di Astrazione: Problematiche e Applicazioni*. Dipartimento di Informatica Sistemistica e Telematica, Università di Genova, Gennaio 1997.

[81] M. Rusinowich, M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In 14th *Computer Security Foundations Workshop.* IEEE Computer Society Press, 2001.

[82] D. Sangiorgi and D. Walker. The π-calculus: a Theory of Mobile Processes, Cambridge University Press, 2001.

[83] S. Schneider. Verifying Authentication Protocols with CSP. In *PCSFW: Proceedings of the 10th Computer Security Foundation Workshop*. IEEE Computer Society Press, 1997.

[84] S. Schneider. Formal Analysis of a Non-Repudiaton Protocol. In *Proceedings of CSFW'98*, pages 54-65. IEEE Press, 1998.

[85] B. Schneier. *Applied Cryptography Second Edition: Protocols, Algorithms, and Source in code C.* John Wiley & Sons, Inc., 1996.

[86] Scott D. Stoller. A Bound on Attacks on Authentication Protocols. To appear in Proceedings of the *2nd IFIP International Conference on Theoretical Computer Science,* 2002. An extended version appeared as Indiana University, Computer Science Dept., Technical Report 526, July 1999 (revised January 2001).

[87] D. P. Sidho. Authentication Protocols for Computer Network. newblock*CNIS*, 11(4):279-310, 1986.

[88] P. Simons. Basic of Computer Security: Cryptographic Protocol. Institute of Computer Science, Rheinische Friedrich-Wihelms-Universität Bonn, March 2000.

[89] E. Snekkens. Exploring the BAN Approact to Protocol Analysis. In *1991 IEEE Symposium on Research in Security and Privacy*, pages 171-181. IEEE Computer Society Press, 1991.

[90] W. Stallings. Network and Internetwork Security: Principles and Practice. Prentice Hall, 1995.

[91] M. J. Toussaint. Formal Verification of Probabilistic Properties in Cryptographic Protocols. *Lecture Notes in Computer Science*, 739, 1993.

[92] A.S. Troelstra, H. Schwichtemberg. Basic Proof Theory. Cambridge University Press 1996, 2000.

[93] D. Ullman. Element of ML Programming. Prentice Hall, 1993.

[94] C. H. West. General Tecnique for Communications Protocol Validation. IBM Journal of Research and Development, 22(4):393-404, July 1978.

[95] Å. Wikström. Functional Programming using Stadard ML. Prentice Hall, 1987.

[96] T. Woo and S. Lam. A Semantic Model for Authentication Protocols. In *Proc. Symposium on Research in Security and Privacy.* IEEE Computer Society Press, 1993.

[97] A. C. Yao. Theory and Application of Tradoor Functions. In *Proc. IEEE Symposium on Fundations of Computer Science*, pages 80-91, 1982.

[98] N. Yoshida, M. Berger, K. Honda. Normalisation in the π-calculus. LICS'01, pages 311-322, IEEE, 2001.

[99] N. Yoshida, K. Honda, M. Berger. Linearity and Bisimulation. To appear as MCS technical report, Leicester, 2001.

[100] S. Zdancewic and A. C. Myers, Secure Information Flow and CPS. ESOP01, LNCS 2028, pages 46-62, Springer, 2001.

[101] E. D. Zwicky, S. Cooper and D. Russell. *Building Internet Firewalls*, 2nd Edition. O'Reilly & Associates, Incorporated. June 2000