UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

VERIFICA FORMALE DEL PROTOCOLLO CRITTOGRAFICO DIFFIE-HELLMAN TRAMITE L'UTILIZZO DI SPIN E TAMARIN

Progetto del corso di Sistemi Operativi Dedicati

Autori

Mattia Sbattella
Jacopo Coloccioni

ANNO ACCADEMICO 2024-2025

Indice

1 Introduzione **1**
- 1.1 Scaletta . . . 1
- 1.2 Introduzione alla verifica formale . . . 1
  - 1.2.1 Modellazione formale dei sistemi . . . 2
  - 1.2.2 Specifiche formali . . . 2
  - 1.2.3 Esplorazione dello spazio degli stati . . . 2
  - 1.2.4 Problematiche . . . 3
- 1.3 Osservazioni sul significato della modellazione formale . . . 3

2 Metodologia e strumenti **5**
- 2.1 Diffie-Hellman . . . 5
  - 2.1.1 Algoritmo . . . 6
  - 2.1.2 Principi di sicurezza . . . 7
  - 2.1.3 Attacco Man-in-the-Middle . . . 7
  - 2.1.4 Prevenzione degli attacchi MITM . . . 8
- 2.2 Il modello di Dolev-Yao . . . 8
  - 2.2.1 Analisi e sintesi dei messaggi . . . 8
  - 2.2.2 Motivazioni per l'utilizzo nel nostro progetto . . . 9
- 2.3 SPIN e Promela . . . 9
  - 2.3.1 iSpin . . . 10
  - 2.3.2 SPIN e Promela nel nostro progetto . . . 10
- 2.4 Tamarin . . . 11
  - 2.4.1 Funzionamento . . . 11
  - 2.4.2 Motivazioni dell'integrazione . . . 11

3 Verifica formale con Spin **13**
- 3.1 Diffie-Hellmann con MITM . . . 13
- 3.2 Codice . . . 14

3.2.1 Parametri e `inline` . . . 14
3.2.2 Canali e variabili globali . . . 15
3.2.3 Processo `Alice` . . . 15
3.2.4 Processo `Bob` . . . 15
3.2.5 Processo `Intruder` . . . 16
3.2.6 Processo `init` e verifica . . . 16
3.3 Risultato . . . 17
3.4 Conclusione . . . 18

4 Verifica formale con Tamarin **19**
4.1 Diffie-Hellman con MITM . . . 19
4.1.1 1. Dichiarazioni iniziali . . . 19
4.1.2 2. Avvio del protocollo da parte di Alice . . . 19
4.1.3 3. Intercettazione dell'attaccante (prima fase) . . . 20
4.1.4 4. Risposta di Bob . . . 20
4.1.5 5. Intercettazione dell'attaccante (seconda fase) . . . 20
4.1.6 6. Conclusione di Alice (con chiave sbagliata) . . . 21
4.1.7 7. Conclusione di Bob (anch'egli con chiave sbagliata) . . . 21
4.1.8 8. Attaccante calcola le chiavi condivise . . . 21
4.1.9 9. Lemmi di violazione . . . 22
4.1.10 10. Fine della teoria . . . 23
4.2 Risultati della verifica formale . . . 23
4.2.1 Interpretazione e significato simbolico delle variabili nei grafi Tamarin . . . 23
4.2.2 Lemma di violazione dell'autenticazione . . . 24
4.2.3 Lemma di violazione della segretezza . . . 26
4.3 Considerazioni e sviluppi futuri . . . 28

5 Conclusioni **29**
5.1 Spin vs. Tamarin . . . 29
5.2 Conclusioni . . . 29
5.3 Sviluppi futuri . . . 30

Bibliografia **32**

Elenco delle figure

2.1 Algoritmo Diffie-Hellman 6
2.2 Diffie-Hellman attacco MITM 7

3.1 Violazione di `assert(keyA_A == keyB_B)` in SPIN 17
3.2 Schema degli scambi in iSPIN 17

4.1 Violazione lemma di autenticazione 24
4.2 Violazione lemma di sicurezza 26

CAPITOLO 1

Introduzione

Il seguente progetto è stato realizzato per la materia di Sistemi Operativi Dedicati. In questo corso era stato introdotto Spin come strumento di validazione formale per sistemi concorrenti. Nel nostro progetto è stato approfondito l'argomento estendendo l'utilizzo a sistemi crittografici, in particolare analizzando il protocollo Diffie-Hellman per lo scambio di chiavi simmetriche.

1.1 Scaletta

Nel seguente capitolo verrà introdotto e affrontato il concetto di **verifica formale**. Il secondo capitolo getta le base metodologica per capire il lavoro svolto nei capitoli 3 e 4. Si parte dalla descrizione del protocollo Diffie–Hellman. Verranno spiegati gli strumenti utilizzati come Spin e Tamarin, sotto il modello simbolico di Dolev–Yao. Nel capitolo 3 si analizza lo scenario di attacco man-in-the-middle con Spin, raffinando l'analisi nel capitolo 4 con Tamarin. L'ultimo capitolo contiene le conclusioni e gli sviluppi futuri del progetto.

1.2 Introduzione alla verifica formale

La *verifica formale* è un insieme di tecniche matematicamente fondate che permettono di dimostrare la correttezza di sistemi complessi rispetto a specifiche formali. Essa si contrappone alla verifica empirica, come il testing o la simulazione, in quanto non si limita ad analizzare un sottoinsieme finito di comportamenti, ma considera *tutti* i comportamenti possibili del sistema, fornendo quindi una garanzia molto più forte.

1.2.1 Modellazione formale dei sistemi

Alla base della verifica formale vi è la necessità di esprimere il comportamento del sistema in una forma rigorosa e matematica, detta *modello formale*. Tale modello è tipicamente costituito da:

- una descrizione delle variabili di stato e delle transizioni ammissibili;
- regole o vincoli che determinano l'evoluzione del sistema;
- una semantica operazionale o assiomatica che definisce il significato del modello.

La modellazione può essere effettuata attraverso linguaggi logici (come la logica temporale o modelli di tipo algebrico), automi finiti, sistemi a stati finiti o strutture più complesse come reti di Petri o processi concorrenti.

1.2.2 Specifiche formali

Parallelamente al modello del sistema, è necessario formalizzare le *proprietà desiderate* del sistema stesso, ovvero ciò che si vuole verificare. Queste proprietà sono espresse in linguaggi formali, come la *logica temporale lineare* (LTL), la *logica computazionale dei rami* (CTL), oppure come *lemmi* o *asserzioni* da dimostrare.

Le proprietà possono riguardare:

- **Correttezza funzionale**: il sistema produce risultati corretti.
- **Sicurezza**: certi stati indesiderati non sono mai raggiunti.
- **Liveness**: certi eventi desiderati accadono sempre eventualmente.
- **Assenza di deadlock**: il sistema non entra mai in uno stato di stallo.

1.2.3 Esplorazione dello spazio degli stati

Una volta costruiti il modello e le specifiche, lo strumento di verifica esplora lo *spazio degli stati*, ovvero l'insieme di tutte le configurazioni possibili del sistema. Questa esplorazione può avvenire in modo:

- **Esplicito**, generando e visitando uno per uno gli stati;
- **Simbolico**, utilizzando strutture compatte che rappresentano insiemi di stati;
- **Deduttivo**, riducendo la verifica a una dimostrazione logica tramite sistemi di inferenza.

Lo scopo è determinare se, lungo tutte le possibili esecuzioni (o *tracce*), le proprietà formali sono rispettate. Se così non è, lo strumento fornisce un *controesempio*, utile per identificare errori nel modello o nel progetto del sistema.

1.2.4 Problematiche

La verifica formale offre garanzie forti e matematicamente fondate. Tuttavia, presenta anche delle sfide:

- **Complessità dello stato**: lo spazio degli stati cresce esponenzialmente con le variabili e i componenti (state explosion problem).
- **Difficoltà di modellazione**: la costruzione di modelli accurati e completi richiede competenze specialistiche.

1.3 Osservazioni sul significato della modellazione formale

Una riflessione importante emersa durante il lavoro con SPIN (e analogamente con Tamarin) riguarda la natura della modellazione formale rispetto alla simulazione di attacchi specifici. È naturale pensare, in fase iniziale, che uno strumento formale venga "programmato" per verificare un singolo scenario di attacco alla volta (come il classico attacco *man-in-the-middle* mostrato nelle sezioni successive). Tuttavia, l'approccio è in realtà molto più generale e potente.

Quando si utilizza un verificatore formale come Tamarin, non si codifica uno specifico attacco, ma si descrive il protocollo in modo astratto, insieme a un modello generico delle capacità dell'avversario (nel nostro caso di tipo Dolev-Yao). Questo processo include:

- La definizione delle regole comportamentali dei partecipanti onesti (es. Alice e Bob), specificando l'invio, la ricezione e l'elaborazione dei messaggi.
- L'introduzione delle assunzioni matematiche necessarie, come l'algebra di protocolli crittografici (come nel nostro caso Diffie-Hellman) e le proprietà delle funzioni di hash.
- La modellazione delle capacità dell'attaccante in termini generali: intercettare, modificare, rigenerare, ritardare o replicare messaggi.

A partire da questi elementi, lo strumento costruisce automaticamente uno *spazio degli stati*, ovvero un grafo delle possibili evoluzioni del sistema. Ogni nodo rappresenta una configurazione simbolica del protocollo, e ogni arco una transizione derivante dall'applicazione di una regola.

La forza della verifica formale risiede nel fatto che lo strumento esplora tutte le traiettorie possibili, non limitandosi ai percorsi esplicitamente codificati. In altri termini, anche se nel modello può essere presente un "blocco sequenziale" che rappresenta un possibile attacco, non ci si limita a quello: si verifica automaticamente tutte le permutazioni plausibili degli eventi, inclusi attacchi per replay, reorder, composizione e

scomposizione dei messaggi, e scenari non previsti manualmente.

Per questo motivo, i lemmi che si formulano (ad esempio quelli relativi ad autenticazione e segretezza) non vengono valutati solo rispetto a un caso particolare, ma rispetto a tutte le possibili esecuzioni del protocollo compatibili con le regole definite. L'eventuale controesempio che si produce rappresenta dunque un vero e proprio attacco dimostrato formalmente.

CAPITOLO 2

Metodologia e strumenti

In questo capitolo vengono affrontate in modo sistematico tute le metodologie e gli strumenti utilizzati per la verifica formale del protocollo Diffie–Hellman, partendo dai fondamenti matematici, passando per la modellazione del classico attacco man-in-the-middle e la formalizzazione simbolica secondo Dolev-Yao. Vengono impiegati due strumenti complementari: SPIN con il linguaggio Promela per l'analisi di concorrenza della logica e Tamarin Prover per la verifica simbolica avanzata di primitive crittografiche.

2.1 Diffie-Hellman

Lo scambio di chiavi (DH) è un metodo crittografico che consente a due entità di generare una chiave simmetrica condivisa, anche se comunicano tramite un canale pubblico e potenzialmente insicuro. Questo protocollo ha rivoluzionato la crittografia moderna, ponendo le basi per la crittografia a chiave pubblica. L'aspetto rivoluzionario di DH consiste nel fatto che non è più necessario scambiarsi fisicamente la chiave segreta, come accadeva con i sistemi crittografici tradizionali. Al contrario, le due parti possono negoziare una chiave segreta condivisa semplicemente scambiando alcuni messaggi pubblici, senza mai trasmettere la chiave stessa. L'algoritmo si basa su proprietà matematiche dei gruppi modulari e, in particolare, sulla difficoltà computazionale del problema del logaritmo discreto.

2.1.1 Algoritmo

Come possiamo vedere in Figura 2.1:

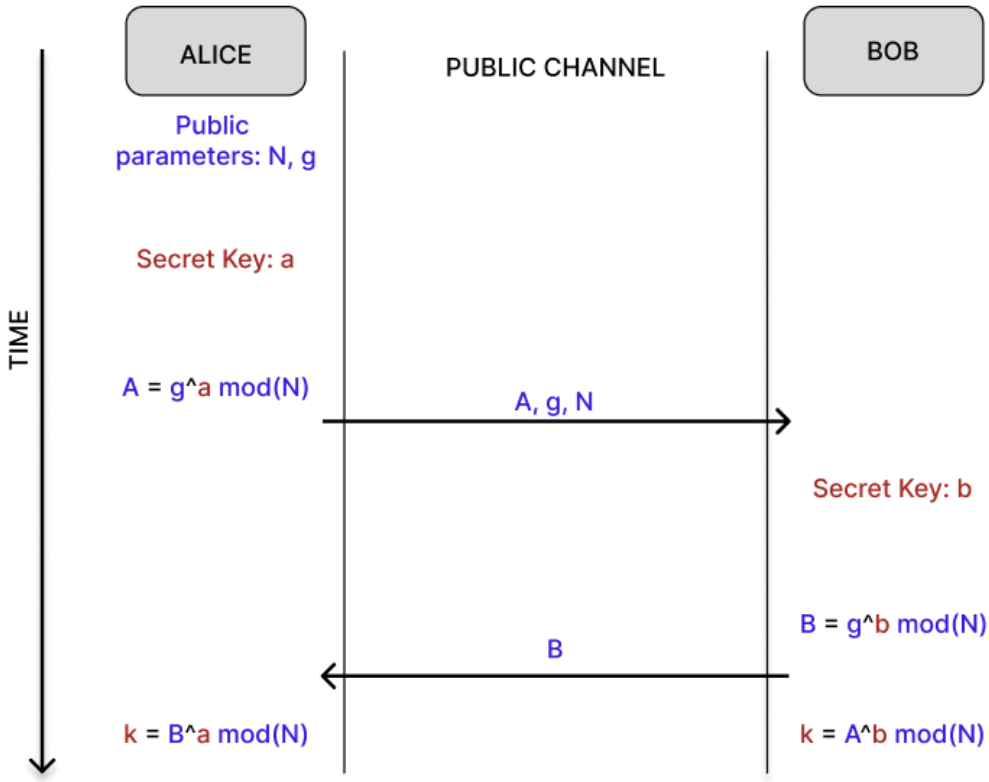

Figura 2.1: Algoritmo Diffie-Hellman

- Alice genera un numero primo molto grande N (circa 300 cifre decimali o 1024 bit) e una radice primitiva g, ossia una base che successivamente elevata a potenza, in aritmetica modulare, genera tutti i numeri che sono primi con N.
- Alice sceglie un numero segreto $a < N$, calcola $A = g^a \mod N$ e invia A, g ed N a Bob.
- Bob sceglie un numero segreto $b < N$, calcola $B = g^b \mod N$ e invia B ad Alice.
- Alice calcola la chiave segreta condivisa: $k = B^a \mod N$.
- Bob calcola la stessa chiave: $k = A^b \mod N$.

I numeri prodotti k sono lo stesso numero; si può quindi utilizzare questo numero o chiave per cifrare tutte le future comunicazioni.

2.1.2 Principi di sicurezza

Uno degli aspetti fondamentali per la sicurezza del protocollo Diffie–Hellman è che, anche se un eventuale attaccante dovesse intercettare i valori pubblici N, g, A e B, non sarebbe in grado di ricavare la chiave segreta $k = g^{ab} \bmod N$, poiché non conosce né il valore privato a scelto da Alice, né il valore b scelto da Bob.

Infatti, per poter risalire a a o b, l'attaccante dovrebbe calcolare:

$$a = \log_g A \quad \text{e} \quad b = \log_g B$$

ma questo implica risolvere il **problema del logaritmo discreto**, che è ritenuto computazionalmente proibitivo per numeri molto grandi. È proprio su questa asimmetria computazionale che si fonda la robustezza del protocollo DH.

2.1.3 Attacco Man-in-the-Middle

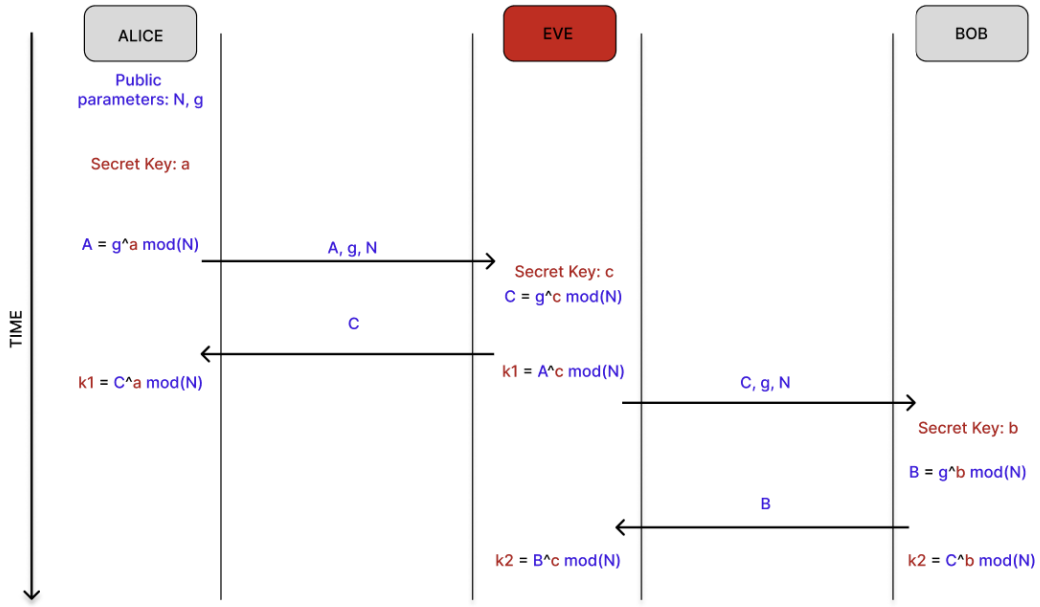

Figura 2.2: Diffie-Hellman attacco MITM

Come altri sistemi crittografici a chiave pubblica, anche il protocollo Diffie–Hellman è vulnerabile a un attacco chiamato *man-in-the-middle* (MITM).

Supponiamo, ad esempio, che un attaccante chiamato Eve intercetti il valore A inviato da Alice a Bob, come possiamo vedere in Figura 2.2. Fingendosi Bob, Eve genera un proprio valore B' e lo invia ad Alice. A questo punto:

- Alice calcola la chiave segreta condivisa con Eve: $k_{AC} = (B')^a \bmod N$.
- Eve calcola la stessa chiave: $k_{AC} = A^c \bmod N$, dove c è il segreto di Eve.
- Eve può ora leggere e (eventualmente) modificare tutti i messaggi che Alice invierà, credendo di parlare con Bob.
- Analogamente, Eve può eseguire lo stesso attacco verso Bob, fingendosi Alice.

In questo scenario, Eve è in grado di intercettare e leggere tutta la comunicazione tra i due interlocutori, rimanendo completamente invisibile.

2.1.4 Prevenzione degli attacchi MITM

Per prevenire attacchi di questo tipo, è necessario introdurre un meccanismo di **autenticazione delle parti**. In genere, questo si realizza facendo uso di un **ente certificatore** (Certification Authority), che garantisce l'identità degli interlocutori tramite *certificati digitali*.

Alice e Bob, prima di eseguire lo scambio DH, possono autenticarsi reciprocamente utilizzando firme digitali e chiavi pubbliche rilasciate da un'autorità fidata. In questo modo, ognuno può verificare che l'altro sia davvero chi dichiara di essere, impedendo a un terzo di inserirsi nella comunicazione.

2.2 Il modello di Dolev-Yao

Il modello di Dolev-Yao, proposto da Danny Dolev e Andrew Yao nel 1983, costituisce un riferimento fondamentale nell'analisi formale dei protocolli crittografici. Esso si configura come un modello simbolico, in cui la comunicazione tra le entità coinvolte avviene attraverso messaggi astratti, rappresentati come termini algebrici manipolabili secondo regole formali. L'assunzione centrale di questo modello è la presenza di un intruso estremamente potente, che ha il completo controllo della rete: può intercettare ogni messaggio, modificarne il contenuto, bloccarne la consegna o inviarne di nuovi a piacimento. Tale potere di manipolazione è massimo rispetto a qualsiasi modello di minaccia realistico, e per questo motivo il modello di Dolev-Yao è spesso considerato una base "pessimistica" ma conservativa per la verifica della sicurezza.

2.2.1 Analisi e sintesi dei messaggi

Il modello impone un vincolo cruciale: l'intruso non può infrangere le primitive crittografiche, che sono trattate come funzioni perfette, ovvero non invertibili senza la

chiave appropriata. Questa astrazione consente di separare l'analisi logica del protocollo dalla sicurezza computazionale degli algoritmi crittografici, permettendo una verifica sistematica delle proprietà strutturali e di confidenzialità. Le operazioni principali a disposizione dell'intruso sono due: *analisi* (`analz`) e *sintesi* (`synth`). Tramite l'analisi, l'intruso può decomporre un messaggio ricevuto, accedere ai suoi componenti (ad esempio nomi, nonce, chiavi pubbliche) e decifrarne il contenuto se possiede la relativa chiave di decrittazione. La sintesi consente invece di costruire nuovi messaggi utilizzando elementi già noti, combinandoli in forme arbitrarie e, eventualmente, cifrandoli con chiavi a sua disposizione. Questo schema permette la simulazione di un'ampia varietà di scenari di attacco, inclusi replay, man-in-the-middle, reflection e impersonificazione.

2.2.2 Motivazioni per l'utilizzo nel nostro progetto

L'efficacia del modello di Dolev-Yao è dimostrata storicamente: utilizzandolo, sono stati scoperti diversi attacchi a protocolli ritenuti sicuri, tra cui il celebre attacco di Gavin Lowe al protocollo Needham-Schroeder. L'adozione di questo modello nel nostro progetto risponde all'esigenza di analizzare il protocollo di Diffie–Hellman in un contesto realistico ma gestibile computazionalmente.

L'astrazione intrinseca del modello di Dolev-Yao è particolarmente vantaggiosa nel nostro caso, poiché il protocollo Diffie–Hellman si basa su operazioni matematicamente complesse, come l'esponenziazione modulare, la cui simulazione a basso livello risulterebbe onerosa e poco informativa ai fini della verifica logica. Possiamo dunque concentrare l'attenzione sulle proprietà fondamentali di segretezza e autenticazione, verificando che la chiave condivisa tra le parti non venga esposta in alcuna delle configurazioni esplorate dal modello, anche in presenza di un intruso con poteri massimi di osservazione e interferenza.

2.3 SPIN e Promela

SPIN (Simple PROMELA INterpreter) è uno strumento di verifica formale dei modelli di sistema concorrente, il cui scopo principale è quello di consentire la modellazione, la simulazione e la verifica automatica di sistemi a stati finiti, tipicamente protocolli di comunicazione e algoritmi distribuiti.

Il cuore di SPIN è il linguaggio Promela (Process Meta Language), un linguaggio di descrizione che consente di specificare il comportamento dei processi, la loro interazione attraverso canali di comunicazione e la gestione della concorrenza.

Uno dei punti di forza di SPIN è la sua capacità di esplorare lo spazio degli stati

del sistema specificato in Promela, al fine di verificare proprietà formali espresse in logica temporale, come ad esempio l'assenza di deadlock, la raggiungibilità di uno stato desiderato o il rispetto di invarianti di sicurezza. Il modello viene convertito in una rappresentazione intermedia ottimizzata e analizzato attraverso un algoritmo di model checking esplicito, capace di identificare automaticamente violazioni delle proprietà dichiarate.

2.3.1 iSpin

iSpin fornisce un'interfaccia grafica semplificata e interattiva per l'utilizzo del model checker SPIN. Nel contesto del nostro progetto, il modello **iSpin** rappresenta uno strumento fondamentale per analizzare la sicurezza logica del protocollo Diffie-Hellman.

Attraverso iSpin possiamo descrivere le interazioni tra le entità coinvolte nel protocollo e simulare visivamente le sequenze di scambio di chiavi.

2.3.2 SPIN e Promela nel nostro progetto

L'utilizzo di SPIN e del linguaggio Promela si è rivelato inizialmente utile per modellare il protocollo concorrente Diffie–Hellman, grazie alla possibilità di descrivere in modo esplicito i processi (es. Alice, Bob, Intruder) e i canali di comunicazione, nonché di simulare lo scenario non deterministico di man-in-the-middle, senza doverci preoccupare di implementare i dettagli di basso livello del networking.

Tuttavia, questa impostazione si è rivelata solo parzialmente adeguata rispetto agli obiettivi del nostro progetto. SPIN, infatti, è progettato per l'analisi della correttezza di sistemi concorrenti, ma presenta limiti significativi nel trattamento di aspetti crittografici simbolici. In particolare:

- **Aritmetica rudimentale:** le operazioni modulari devono essere implementate tramite inline e cicli ripetuti, limitando la scala dei moduli e rendendo meno immediata l'estensione a esponenziazioni reali su grandi numeri.
- **Stato finito:** SPIN richiede che lo spazio degli stati sia finito e relativamente piccolo; la verifica di numerosi sessioni o di parametri grandi conduce rapidamente a un'esplosione combinatoria.
- **Assenza di teoria crittografica:** non è possibile incorporare direttamente equazioni del Diffie–Hellman o ragionare su proprietà come la non invertibilità del logaritmo discreto a livello simbolico.

Queste limitazioni ci hanno spinto a integrare nel progetto lo strumento Tamarin, specificamente progettato per la verifica simbolica di protocolli crittografici. Il suo funzionamento è descritto in seguito.

2.4 Tamarin

Tamarin Prover è uno strumento open-source per la verifica formale di protocolli crittografici, sviluppato con l'obiettivo di analizzare le proprietà di sicurezza di protocolli complessi. È basato su una logica del primo ordine con supporto per il tempo e la riscrittura, e consente la modellazione di un ampio spettro di primitive crittografiche, inclusi Diffie-Hellman, cifratura simmetrica, hash e firme digitali.

2.4.1 Funzionamento

Il funzionamento di Tamarin si basa sulla definizione di un *modello formale* del protocollo, espresso tramite un insieme di *regole di riscrittura* che descrivono le transizioni di stato, come l'invio e la ricezione di messaggi, la generazione di nonce, e il calcolo di chiavi. Il protocollo viene così rappresentato come un sistema simbolico, il cui spazio delle esecuzioni viene esplorato automaticamente per verificarne le proprietà di sicurezza.

Il comportamento dell'avversario (come descritto in precedenza) è modellato secondo il paradigma di Dolev-Yao, che assume un attaccante con pieno controllo del canale di comunicazione: può intercettare, modificare, reinviare e comporre messaggi, pur senza rompere le primitive crittografiche (considerate ideali).

Tamarin consente di specificare le proprietà di sicurezza tramite *lemmi*, espressi in logica temporale, per formalizzare requisiti come autenticazione, segretezza, consenso e integrità. In caso di violazione, lo strumento produce un controesempio sotto forma di trace del sistema.

Tra le principali caratteristiche di Tamarin si annoverano:

- uso del paradigma *multiset rewriting* combinato con tecniche di *constraint solving*, per modellare formalmente primitive crittografiche (ad es. la teoria Diffie–Hellman);
- specifica dichiarativa dei ruoli e delle regole di comunicazione, con supporto nativo per equazioni modulo e per operazioni tipiche della crittografia, come il logaritmo discreto;
- possibilità di verificare proprietà in un numero arbitrario di sessioni, anche simultanee, mediante strategie automatiche e lemmi interattivi in logica temporale.

2.4.2 Motivazioni dell'integrazione

Nella fase iniziale del progetto, la verifica del protocollo era stata approcciata tramite il model checker *SPIN*, utile per analizzare la concorrenza, la sincronizzazione e la correttezza funzionale di sistemi concorrenti. Come già accennato, SPIN si basa

su automi finiti e non include costrutti nativi per modellare operazioni crittografiche, né supporta un modello esplicito dell'attaccante nel senso richiesto dalla crittografia formale.

Per questo motivo, è stato deciso di integrare Tamarin nel progetto come strumento primario per la verifica dei protocolli crittografici. I suoi punti di forza includono:

- supporto nativo per le primitive crittografiche;
- modellazione formale dell'attaccante (Dolev-Yao);
- capacità di dimostrare proprietà complesse (es. segretezza di chiavi, autenticazione mutua, forward secrecy);
- interfaccia per analizzare i trace e i controesempi generati automaticamente.

Per questo Tamarin si è rivelato uno strumento più adatto per la verifica formale di Diffie–Hellman, permettendo un'analisi completa e automatica delle potenziali vulnerabilità e delle proprietà critiche da garantire.

CAPITOLO 3

Verifica formale con Spin

*Lo scopo di questo capitolo è utilizzare il **verificatore formale SPIN** per analizzare la sicurezza dello **scambio di chiavi Diffie-Hellman** in presenza di un attaccante di tipo (MITM), modellato secondo Dolev-Yao.*

3.1 Diffie-Hellmann con MITM

Sulla base delle osservazioni introdotte nei capitoli precedenti, ci proponiamo ora di modellare lo scambio di chiavi Diffie–Hellman tenendo conto delle sue potenziali vulnerabilità nel contesto di un canale insicuro. Il protocollo DH si basa su un fondamento matematico solido: la difficoltà computazionale del problema del logaritmo discreto garantisce, in assenza di attacchi attivi, un elevato livello di sicurezza. Tuttavia, la sua implementazione canonica non fornisce alcun meccanismo di autenticazione tra le parti coinvolte nella comunicazione, lasciando così spazio a un attacco MITM.

Nel dettaglio, in uno scenario non autenticato, un attaccante $\mathcal{I}$ può facilmente intercettare il messaggio che Alice invia a Bob, sostituirlo con uno proprio, e ripetere la stessa operazione nella direzione opposta. In questo modo, sia Alice che Bob negozieranno una chiave segreta, ma ciascuno di loro lo farà con l'attaccante, anziché con l'interlocutore legittimo. $\mathcal{I}$ potrà così decifrare, modificare o semplicemente osservare qualsiasi comunicazione futura cifrata con tali chiavi.

Dal punto di vista formale, questo comportamento è perfettamente catturato dal modello Dolev–Yao, che consente di esplorare in modo esaustivo tutte le possibili interazioni tra i processi coinvolti. Attraverso l'uso del linguaggio Promela e del model checker SPIN, possiamo:

- Simulare l'intero protocollo, includendo Alice, Bob e l'attaccante MITM.

- Verificare proprietà di sicurezza come l'integrità e la segretezza della chiave condivisa.
- Dimostrare che $\mathcal{I}$ può creare una duplice chiave per A e per B senza che essi ne siano a conoscenza.

L'idea centrale è dimostrare che, nonostante la forza crittografica del protocollo DH in sé, esso può essere logicamente compromesso se utilizzato senza meccanismi di autenticazione o controllo del canale. Ciò rafforza l'importanza di integrare DH in schemi di autenticazione robusti (come TLS o IPsec) o di utilizzare versioni autenticabili del protocollo.

3.2 Codice

Il modello Promela è organizzato in tre `proctype` , ciascuno corrispondente a un partecipante:

- **Alice**: genera $g^a \bmod p$ e calcola la chiave con l'intruso.
- **Bob**: riceve $g^c \bmod p$, genera $g^b \bmod p$ e calcola la chiave con l'intruso.
- **Intruder**: intercetta g^a e g^b, calcola le chiavi $(g^a)^c$ e $(g^b)^c$ e ritrasmette g^c.

3.2.1 Parametri e `inline`

Definiamo un piccolo primo $p = 23$, un generatore $g = 5$ e un inline `modexp` per il calcolo di $\text{base}^{\text{exp}} \bmod p$:

```
#define P 23
#define G 5

inline modexp(base, exp) {
    int _i; int _res;
    _res = 1; _i = 0;
    do
    :: (_i < exp) ->
         _res = (_res * base) % P;
         _i++;
    :: else -> break
    od;
    return _res;
}
```

3.2.2 Canali e variabili globali

Utilizziamo canali di tipo `int` e variabili globali per le chiavi e i flag di completamento:

```
chan A_to_I = [1] of { int };  // Alice → Intruder
chan I_to_B = [1] of { int };  // Intruder → Bob
chan B_to_I = [1] of { int };  // Bob → Intruder
chan I_to_A = [1] of { int };  // Intruder → Alice

int keyA_A, keyA_I;   // chiavi AliceIntruso
int keyB_B, keyB_I;   // chiavi BobIntruso
bool doneA=false, doneB=false, doneI=false;
```

3.2.3 Processo `Alice`

```
proctype Alice() {
    int a = 6, GA, recv;

    /* Fase 1: invia g^a mod p */
    GA = modexp(G, a);
    A_to_I ! GA;

    /* Fase 4: riceve g^c mod p e calcola chiave */
    I_to_A ? recv;
    keyA_A = modexp(recv, a);

    doneA = true;
}
```

3.2.4 Processo `Bob`

```
proctype Bob() {
    int b = 15, GB, recv;

    /* Fase 2: riceve g^c mod p */
    I_to_B ? recv;

    /* Fase 3: invia g^b mod p */
    GB = modexp(G, b);
    B_to_I ! GB;

    /* Fase 4: calcola chiave */
    keyB_B = modexp(recv, b);
```

```
    doneB = true;
}
```

3.2.5 Processo `Intruder`

```
proctype Intruder() {
    int c = 13, x;

    /* Step 1: intercetta g^a, calcola (g^a)^c mod(p) e invia g^c mod(p) a Bo
    A_to_I ? x;
    keyA_I = modexp(x, c);
    x = modexp(G, c);
    I_to_B ! x;

    /* Step 2: intercetta g^b, calcola (g^b)^c mod(p)e invia g^c mod(p) ad Al
    B_to_I ? x;
    keyB_I = modexp(x, c);
    x = modexp(G, c);
    I_to_A ! x;

    doneI = true;
}
```

3.2.6 Processo `init` e verifica

Alla fine si lancia tutto e si controllano gli `assert` :

```
init {
    run Alice(); run Bob(); run Intruder();
    do
    :: (doneA && doneB && doneI) ->
        /* 1) Intruso vs Alice */
        assert(keyA_I == keyA_A);
        /* 2) Intruso vs Bob */
        assert(keyB_I == keyB_B);
        /* 3) Alice vs Bob (deve fallire) */
        assert(keyA_A == keyB_B);
        break
    od
}
```

L'ultimo `assert` fallirà, generando la contro-esecuzione che mostra l'attacco MITM: Alice e Bob finiscono con chiavi diverse nonostante credano di condividere la stessa secret key.

3.3 Risultato

Eseguendo SPIN sul modello, otteniamo subito la conferma dell'attacco MITM tramite la violazione dell'ultimo `assert` :

```
Verifica chiavi...
145:        proc  0 (:init::1) DH_02.pml:96 (state 5) [printf('Verifica chiavi...\\n')]
146:        proc  0 (:init::1) DH_02.pml:98 (state 6) [assert((keyA_I==keyA_A))]
147:        proc  0 (:init::1) DH_02.pml:100 (state 7) [assert((keyB_I==keyB_B))]
spin: DH_02.pml:102, Error: assertion violated
spin: text of failed assertion: assert((keyA_A==keyB_B))
#processes: 1
148:        proc  0 (:init::1) DH_02.pml:102 (state 8)
4 processes created
Exit-Status 0
```

Figura 3.1: Violazione di `assert(keyA_A == keyB_B)` in SPIN

Come si vede in figura 3.1, SPIN interrompe l'esecuzione perché Alice e Bob non calcolano la stessa chiave:

$$\text{keyA\_A} \;=\; (g^c)^a mod(p) \;\not\equiv\; (g^c)^b mod(p) \;=\; \text{keyB\_B}.$$

Questo dimostra che, nonostante Alice e Bob credano di condividere una key segreta, in realtà ognuno l'ha negoziata con l'intruso.

Con iSpin possiamo anche visualizzare lo schema degli scambi nel counterexample:

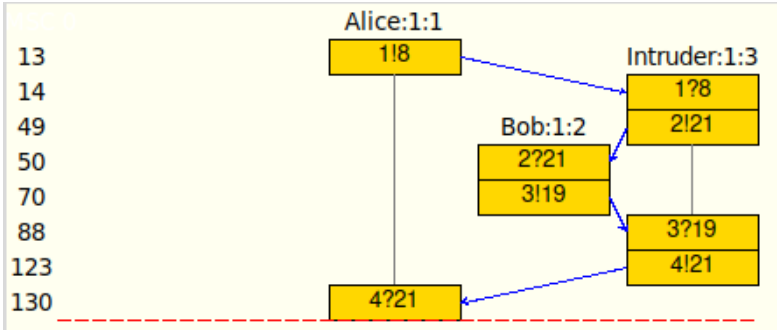

Figura 3.2: Schema degli scambi in iSPIN

In particolare (figura 3.2):

1. **Alice → Intruder:** invio di g^a mod p (`8`).
2. **Intruder → Bob:** invio di g^c mod p al posto di `21` .
3. **Bob → Intruder:** invio di g^b mod p (`19`).
4. **Intruder → Alice:** invio di g^c mod p al posto di `21` .

Al termine, Alice calcola keyA_A $= (g^c)^a$ mod p, Bob calcola keyB_B $= (g^c)^b$ mod p, e poiché $a \neq b$ risulta keyA_A $\neq$ keyB_B, scatenando la violazione del terzo assert e confermando l'efficacia dell'attacco Man-in-the-Middle.

3.4 Conclusione

Il modello conferma formalmente che, in assenza di autenticazione, un attaccante Dolev–Yao capace di intercettare e ricostruire i valori modulari può indurre Alice e Bob a calcolare chiavi diverse. Questo evidenzia la necessità di integrare Diffie–Hellman in protocolli autenticati (es. TLS, IPsec) per fronteggiare efficacemente gli attacchi MITM.

CAPITOLO 4

Verifica formale con Tamarin

In questo capitolo viene affrontato Tamarin per la verifica formale di DH non autenticato. Si descriverà dettagliatamente il codice utilizzato per poi discutere dei lemmi di autenticazione e segretezza violati.

4.1 Diffie-Hellman con MITM

4.1.1 1. Dichiarazioni iniziali

```
theory DH_MITM
begin

builtins: diffie-hellman, hashing
functions: h/1
```

In questa prima parte vengono definite le impostazioni globali della teoria in Tamarin. La teoria si chiama 'DH_MITM', ad indicare che si tratta di un'analisi formale del protocollo Diffie-Hellman soggetto a un attacco di tipo Man-In-The-Middle (MITM).

La dichiarazione 'builtins: *diffie-hellman, hashing* abilita l'uso di due costrutti fondamentali: il primo serve per eseguire il calcolo delle potenze modulari (come g^a), mentre il secondo abilita una funzione di hash crittografico. La funzione 'h/1', definita come un'astrazione unaria, rappresenta proprio una funzione di hash a un solo argomento, che verrà utilizzata per derivare chiavi segrete condivise.

4.1.2 2. Avvio del protocollo da parte di Alice

```
rule Initiator_Start:
  [ Fr(~a) ]
  -->
  [ Out('g'^~a), State_Initiator(~a) ]
```

In questo primo passo del protocollo, Alice (il mittente) inizia la negoziazione della chiave. Genera un numero segreto casuale a (espresso da $Fr(\sim a)$), che rappresenta il suo esponente privato. Dopodiché, calcola la potenza g^a e la invia pubblicamente nel canale di rete ($Out('g'\ \hat{}\sim a)$). Infine, conserva internamente lo stato ($State\_Initiator(\sim a)$ che le permette di ricordare il valore a per i calcoli successivi.

4.1.3 3. Intercettazione dell'attaccante (prima fase)

```
rule MITM_Intercept_A:
  [ In(ga), Fr(~x) ]
  -->
  [ Out('g'^~x), MITM_A(ga, ~x) ]
```

Questa regola descrive il comportamento dell'attaccante Eve. Quando riceve il messaggio g^a da Alice (attraverso $In(ga)$), non lo inoltra a Bob, ma al suo posto genera un proprio esponente segreto x ($Fr(\sim x)$) e invia g^x al destinatario ($Out('g'\ \hat{}\sim x)$). Questo fa sì che Bob creda di comunicare con Alice, mentre in realtà sta parlando con Eve. L'attaccante salva lo stato $MITM\_A(ga, \sim x)$ per ricordare le informazioni acquisite: ciò sarà utile per continuare l'attacco più avanti.

4.1.4 4. Risposta di Bob

```
rule Responder_Reply:
  [ In(gx), Fr(~b) ]
  -->
  [ Out('g'^~b), State_Responder(~b), Record_B(gx, ~b) ]
```

Bob, che riceve un messaggio (che in realtà è g^x generato dall'attaccante), genera il proprio esponente segreto b ($Fr(\sim b)$). Calcola quindi g^b e lo invia pubblicamente ($Out('g'\ \hat{}\sim b)$). Allo stesso tempo, memorizza sia il proprio esponente con $State\_Responder(\sim b$ sia il valore g^x che ha ricevuto ($Record\_B(gx, \sim b)$. Bob è ignaro del fatto che non sta comunicando direttamente con Alice, ma con l'attaccante.

4.1.5 5. Intercettazione dell'attaccante (seconda fase)

```
rule MITM_Intercept_B:
  [ In(gb), Fr(~y), MITM_A(ga, x) ]
  -->
  [ Out('g'^~y), MITM_Finish(ga, x, gb, ~y) ]
```

Eve intercetta anche il messaggio di Bob contenente g^b ($In(gb)$) e, invece di trasmetterlo ad Alice, genera un altro esponente segreto y ($Fr(\sim y)$) e invia g^y a lei. Questo fa sì che Alice calcolerà la chiave su un valore che non proviene da Bob. L'attaccante aggiorna il proprio stato a $MITM\_Finish(...)$, che contiene tutto ciò che ha appreso: g^a, x, g^b e y. Questi dati le permetteranno di derivare entrambe le chiavi usate da Alice e Bob.

4.1.6 6. Conclusione di Alice (con chiave sbagliata)

```
rule Initiator_Conclude:
  [ In(gy), State_Initiator(a) ]
  -->
  [ Key('A', 'B', h(gy^a)), Commit('A', 'B') ]
```

Alice riceve g^y da quella che crede essere la risposta di Bob ($In(gy)$) e calcola la chiave condivisa come $k_A = h((g^y)^a)$. In realtà, g^y è stato generato dall'attaccante, quindi la chiave non è realmente condivisa con Bob. Infine, Alice considera la sessione come completata e si impegna nella comunicazione ($Commit('A','B')$).

4.1.7 7. Conclusione di Bob (anch'egli con chiave sbagliata)

```
rule Responder_Conclude:
  [ State_Responder(b), Record_B(gx, b) ]
  -->
  [ Key('B', 'A', h(gx^b)), Commit('B', 'A') ]
```

Analogamente, Bob completa la fase finale del protocollo. Usa il valore ricevuto all'inizio (g^x, inviato da Eve) e calcola la chiave $k_B = h((g^x)^b)$. Anche in questo caso, la chiave non corrisponde a quella calcolata da Alice. Tuttavia, Bob si fida e conclude la sessione con ($Commit('B','A')$).

4.1.8 8. Attaccante calcola le chiavi condivise

```
rule MITM_Derive_Secrets:
  [ MITM_Finish(ga, x, gb, y) ]
  -->
  [ K(h(ga^y)), K(h(gb^x)) ]
```

Qui l'attaccante sfrutta tutte le informazioni raccolte per calcolare entrambe le chiavi private:

- $k_1 = h((g^a)^y)$: chiave che Alice crede di condividere con Bob.
- $k_2 = h((g^b)^x)$: chiave che Bob crede di condividere con Alice.

Poiché Eve conosce sia a, b indirettamente (via g^a, g^b) che i propri x, y, può computare entrambe le chiavi. La notazione 'K(...)' significa che queste chiavi ora sono note all'attaccante.

4.1.9 9. Lemmi di violazione

Violazione dell'autenticazione

```
lemma authentication_violation:
  "Ex #i. Commit('A','B')@i & not(Ex #j. Commit('B','A')@j)"
```

Questo lemma verifica la proprietà di autenticazione. Vuole mostrare se esiste almeno un punto nel trace in cui Alice si è impegnata nella comunicazione con Bob ($Commit('A','B')@i$), ma in nessun momento Bob ha fatto lo stesso: ($not(Ex\#j.Commit('B','A')@j)$). Questo dimostra che l'attaccante è riuscito a ingannare Alice, facendole credere che stesse parlando con Bob quando invece non è vero.

Simbolo / Parte	Significato
`Ex`	"Esiste" – quantificatore esistenziale che afferma l'esistenza di almeno un elemento che soddisfa la proprietà indicata.
`#i`	Istanziatore di tempo: rappresenta un punto temporale o uno step all'interno della trace di esecuzione.
`Commit('A','B')@i`	Evento `Commit` avvenuto all'istante `i`, dove `A` ritiene di aver completato correttamente una sessione con `B`.
`&`	Connettivo logico "E": entrambe le condizioni collegate devono essere vere simultaneamente.
`not(...)`	Negazione logica: ciò che è contenuto tra parentesi non deve essere vero.
`Ex #j. Commit('B','A')@j`	Esiste un istante `j` in cui `B` fa commit con `A` $\rightarrow$ in questo lemma, si afferma che tale evento non deve mai verificarsi.

Tabella 4.1: Spiegazione del lemma Tamarin `Ex #i. Commit('A','B')@i & not(Ex #j. Commit('B','A')@j)`

Violazione della segretezza

```
lemma secrecy_violation:
  "Ex k #i. Key('A','B',k)@i & K(k)@i"
```

Il secondo lemma verifica se l'attaccante è riuscito a violare la segretezza della chiave. Esprime che esiste una chiave k condivisa da Alice e Bob ($Key('A','B',k)$), ma che questa chiave è conosciuta anche da un attaccante ($K(k)$). In un protocollo sicuro,

ciò non dovrebbe mai accadere. Questo lemma formalizza quindi la violazione della confidenzialità.

Simbolo / Parte	Significato
`Ex k`	"Esiste una chiave `k` " – quantificatore esistenziale che afferma l'esistenza di una chiave specifica che soddisfa la proprietà.
`#i`	Istanziatore di tempo: rappresenta un punto temporale o uno step nella trace di esecuzione.
`Key('A','B',k)@i`	Evento che indica che la chiave `k` è stata generata per la comunicazione tra `A` e `B` al tempo `i` .
`K(k)@i`	Il fatto che la chiave `k` è nota all'attaccante (`K` sta per "knowledge") nello stesso istante `i` .
`&`	Connettivo logico "E": entrambi gli eventi devono verificarsi nello stesso istante temporale `i` .

Tabella 4.2: Spiegazione del lemma Tamarin `Ex k #i. Key('A','B',k)@i & K(k)@i`

4.1.10 10. Fine della teoria

```
end
```

Questa semplice riga chiude la definizione della teoria Tamarin. Tutti i blocchi, regole e lemmi sono ora parte della teoria 'DH_MITM' che può essere eseguita e verificata nel tool.

4.2 Risultati della verifica formale

Lanciando la verifica formale si esplorano ben **2304** stati (sono stati logici, rappresentano insiemi di proposizioni logiche). La computazione avviene in circa 4 minuti e 20 secondi, evidenziando il complessità esponenziale dovuta all'esplosione delle combinazioni di possibili stati realizzabili.

4.2.1 Interpretazione e significato simbolico delle variabili nei grafi Tamarin

Nei grafi generati da Tamarin Prover, come quelli mostrati nelle figure successive, compaiono frequentemente variabili con una notazione del tipo `x.20` , `y.10` , oppure `~x.20` , `ga.10` , ecc. È importante comprendere correttamente il significato di questa notazione, altrimenti si rischia di fraintendere la semantica del protocollo.

Variabili simboliche, non valori numerici I nomi come `x.20` o `y.10` non indicano che il valore della variabile sia 20 o 10. Il suffisso numerico (es. `.20` , `.10` , `.15`) serve soltanto a rendere **univoco** il nome della variabile all'interno della traccia di esecuzione.

Questo è necessario in quanto Tamarin può eseguire lo stesso protocollo molte volte in parallelo, e deve essere in grado di distinguere ogni istanza. Questa notazione è fondamentale per la tracciabilità e la verifica formale:

- Serve a distinguere le istanze delle stesse regole applicate più volte.
- Permette di legare ogni valore generato o usato a un preciso punto della traccia temporale.
- Aiuta a ricostruire chi ha generato cosa, e se l'attaccante è riuscito a manipolare o calcolare certi segreti.

Ad esempio:

- `x.15` potrebbe rappresentare il segreto privato di Alice (es. l'esponente a in Diffie-Hellman).
- `ga.10` potrebbe rappresentare il valore $g^{x.15}$, calcolato da Alice e inviato come parte del protocollo.
- `y.10` potrebbe rappresentare l'esponente scelto da Bob, distinto da altri possibili valori di `y` generati in altri momenti o sessioni.
- `~x.20` rappresenta una variabile generata **dall'attaccante** (il simbolo `~` indica che il valore è sotto il controllo dell'avversario).

4.2.2 Lemma di violazione dell'autenticazione

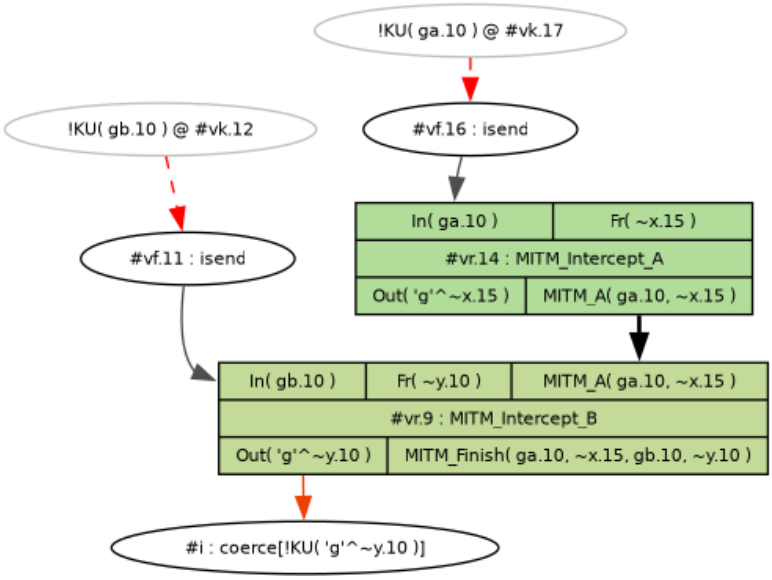

Figura 4.1: Violazione lemma di autenticazione

Nella figura riportata ogni nodo rappresenta un evento, mentre le frecce rappresentano la successione temporale degli eventi secondo una traccia accettata dal sistema.

- **Nodi ovali con bordo grigio** rappresentano fatti statici o assunzioni iniziali, come la pubblicazione delle chiavi pubbliche:
  - `!KU(ga.10)` e `!KU(gb.10)` rappresentano la pubblicazione delle chiavi pubbliche di Alice e Bob.
- **Nodi ovali con bordo nero** (es. `#vf.11 : isend`) rappresentano eventi in cui le chiavi pubbliche vengono effettivamente inviate.
- **Blocchi verdi** rappresentano le regole definite nel modello Tamarin per simulare l'intercettazione da parte dell'attaccante:
  - `MITM_Intercept_A` intercetta il messaggio di Alice contenente `ga.10`, genera un numero casuale `~x.15` e risponde con `gˆ~x.15`.
  - `MITM_Intercept_B` intercetta il messaggio di Bob `gb.10`, genera `~y.10` e risponde con `gˆ~y.10`.
  - `MITM_Finish` unisce i dati generati per completare l'attacco.
- **Freccia rossa tratteggiata** rappresenta un collegamento causale diretto tra l'assunzione di conoscenza della chiave e il suo utilizzo da parte dell'attaccante.
- **Nodo finale (coerce)**: rappresenta l'inferenza dell'attaccante, che riesce a derivare una chiave condivisa `gˆ~y.10`, violando la segretezza.

Come descritto in precedenza, il lemma definito nel modello per verificare la corretta autenticazione tra Alice e Bob è il seguente:

```
lemma authentication_violation:
"Ex #i. Commit('A','B')@i & not(Ex #j. Commit('B','A')@j)"
```

Questo lemma verifica se esiste almeno una sessione in cui:

- Alice ha completato la comunicazione (es. ha ricevuto il messaggio DH da Bob) e pensa di aver comunicato con Bob;
- ma non esiste alcuna sessione in cui Bob ha fatto lo stesso nei confronti di Alice.

Se il lemma è `sat` (soddisfatto), significa che l'autenticazione è violata: Alice ha comunicato con un'entità che crede essere Bob, ma Bob non ha mai partecipato veramente a quella sessione.

Analizzando la figura:

- Alice invia `ga.10`, intercettato dall'attaccante tramite `MITM_Intercept_A`, che risponde con `gˆ~x.15`.

- Bob invia `gb.10` , anch'esso intercettato, e riceve in risposta `g^~y.10` .
- L'attaccante costruisce due chiavi diverse: una con Alice (`g^~x.15`) e una con Bob (`g^~y.10`).
- Alice crede di parlare con Bob e quindi effettua un `Commit('A','B')` , ma Bob in realtà non ha mai interagito con Alice direttamente — dunque `Commit('B','A')` non avviene.

Ciò conferma esattamente la condizione del lemma: esiste una comunicazione unilaterale, e l'autenticazione è compromessa a causa dell'intercettazione attiva da parte dell'attaccante.

4.2.3 Lemma di violazione della segretezza

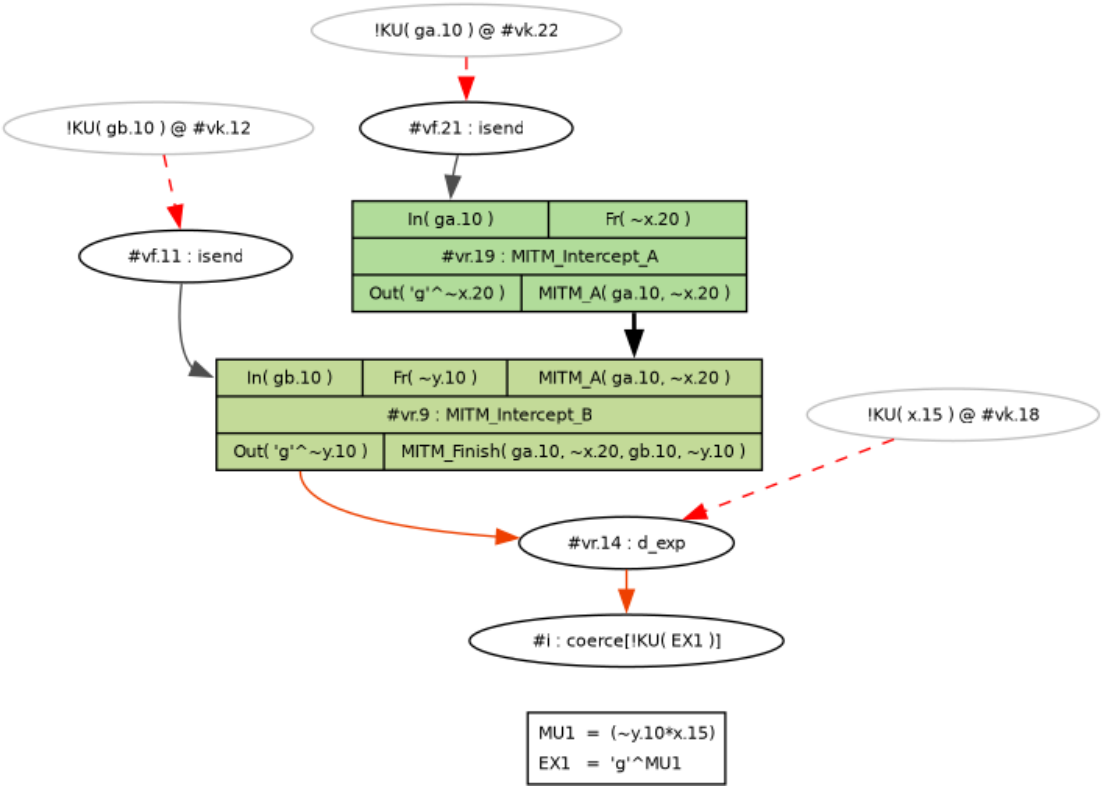

Figura 4.2: Violazione lemma di sicurezza

La seguente figura rappresenta la violazione del lemma di **segretezza della chiave condivisa**.

- **Nodi ovali con bordo grigio** (`!KU(ga.10)` , `!KU(gb.10)` , `!KU(x.15)`) rappresentano la pubblicazione delle chiavi pubbliche di Alice, Bob e di un terzo attore (malevolo).

- **Nodi ovali con bordo nero** (es. `#vf.11 : isend`) indicano eventi di invio (send) che hanno luogo nel protocollo.
- **Blocchi verdi** sono le regole definite per simulare l'attacco MITM:
  - `MITM_Intercept_A` riceve il messaggio `ga.10` da Alice, genera un segreto `~x.20`, e invia in risposta `g^~x.20`.
  - `MITM_Intercept_B` riceve il messaggio `gb.10` da Bob, genera `~y.10`, e risponde con `g^~y.10`.
  - `MITM_Finish` raccoglie i dati di entrambe le intercettazioni e li archivia per uso successivo.
- **Penultimo nodo `d_exp`** rappresenta un'operazione di calcolo della chiave condivisa a partire dagli esponenti:

  `MU1 = (~y.10 * x.15)  EX1 = g^MU1`
- **Nodo finale `coerce`** dimostra che l'attaccante ha ottenuto la conoscenza della chiave derivata `g^(~y.10 * x.15)`, violando così la proprietà di segretezza.

Come descritto in precedenza, il lemma definito nel modello per verificare la corretta segretezza tra Alice e Bob è il seguente:

```
lemma secrecy_violation:
  "Ex k #i. Key('A','B',k)@i & K(k)@i"
\end{lstlisting}
```

Questo lemma indica che l'attaccante è riuscito a calcolare (coercere) il valore della chiave condivisa tra Bob e un altro attore, usando i dati intercettati.

Dalla figura si evince che:

- L'attaccante ha intercettato entrambi i messaggi di scambio chiave (da Alice e Bob).
- Ha manipolato entrambi i canali, sostituendo le basi DH con propri esponenti `~x.20` e `~y.10`.
- Conosce anche il valore di `x.15`, la chiave pubblica di un altro attore (potenzialmente compromesso).
- L'attaccante esegue l'operazione `d_exp` per calcolare la chiave `g^(~y.10 * x.15)`, e riesce ad ottenere il valore finale `EX1`.

Questo dimostra una **violazione della segretezza**: un agente non autorizzato ha ottenuto la chiave condivisa che dovrebbe essere nota solo ad Alice e Bob.

4.3 Considerazioni e sviluppi futuri

Il modello Tamarin utilizzato come nel nostro progetto è adatto per scopi didattici e dimostrativi: mostra chiaramente le vulnerabilità principali del protocollo Diffie-Hellman privo di autenticazione. Tuttavia, non è sufficiente per una verifica esaustiva. Un'estensione con ruoli generici, canale Dolev-Yao pienamente modellato ed eventi ben definiti, permetterebbe una formalizzazione più robusta, adatta a validazioni accademiche o analisi comparative tra protocolli.

In particolare risultano evidenti i limiti elencati e proposti da migliorere negli sviluppi futuri nell'ultimo capitolo.

CAPITOLO 5

Conclusioni

In questo capitolo conclusivo verranno considerati gli aspetti fondamentali emersi alla fine del progetto, dando spunti per sviluppi e ampliamenti futuri.

5.1 Spin vs. Tamarin

Nel corso del progetto abbiamo confrontato due approcci complementari alla verifica formale del protocollo Diffie–Hellman in presenza di un attaccante di tipo Dolev–Yao:

- **SPIN/Promela**: ideale per l'analisi di sistemi concorrenti con spazio di stato finito. Ha consentito di modellare esplicitamente i processi Alice, Bob e Intruder, evidenziando in pochi passi l'efficacia dell'attacco Man-in-the-Middle tramite la violazione degli `assert` . Tuttavia, le limitazioni nell'aritmetica modulare e l'assenza di primitive crittografiche native ne riducono la scalabilità e la generalità per protocolli reali.
- **Tamarin**: progettato specificamente per protocolli crittografici simbolici. Grazie al supporto nativo per equazioni Diffie–Hellman e al modello Dolev–Yao integrato, ha permesso di formulare lemmi di autenticazione e segretezza e di ottenere controesempi formali in numero arbitrario di sessioni. L'approccio multiset rewriting e i lemmi in logica temporale offrono una maggiore espressività, pur richiedendo una curva di apprendimento più ripida.

5.2 Conclusioni

L'analisi formale ha confermato che:

1. *Diffie–Hellman senza autenticazione* è intrinsecamente vulnerabile a un attacco MITM: un intruso con controllo del canale può negoziare due chiavi distinte ingannando Alice e Bob.
2. *SPIN* fornisce un riscontro rapido e concreto sull'inefficacia del protocollo privo di meccanismi di autenticazione.
3. *Tamarin* offre una visione più generale e modulare, in grado di dimostrare proprietà di sicurezza (segretezza, autenticazione) in modo simbolico e per un numero arbitrario di sessioni.

Questi risultati sottolineano l'importanza cruciale di integrare sempre Diffie–Hellman in contesti autenticati (es. TLS, IPsec) e di sfruttare strumenti formali adeguati al dominio crittografico.

5.3 Sviluppi futuri

Per estendere e rafforzare il lavoro svolto, proponiamo:

- **Parametrizzazione dei ruoli**: attualmente il modello utilizza identificatori fissi (es. `'A'`, `'B'`), limitando la generalizzabilità. Si propone di introdurre ruoli parametrizzati (es. `role_id`) per aumentare la riusabilità e permettere l'analisi di scenari con partecipanti multipli e dinamici.
- **Arricchimento del modello con eventi**: sostituire o affiancare gli stati (`State_Initiator`, `State_Responder`) con eventi espliciti come `Start`, `Commit`, `Reveal`, per una tracciabilità più modulare e per facilitare l'espressione delle proprietà da verificare.
- **Contromisure integrate**: estendere il modello con varianti del protocollo che includano firme digitali, certificati, nonce o chiavi precondivise, al fine di confrontare formalmente le proprietà di sicurezza con e senza difese.
- **Analisi estesa degli attacchi**: attualmente il modello considera solo un singolo attacco. Sarebbe utile simulare un attaccante Dolev-Yao completo, modellando messaggi malformati, replay, reorder e interferenze attive nel canale.
- **Analisi di protocolli avanzati**: valutare estensioni di Diffie–Hellman, come ECDH o versioni con forward secrecy, sfruttando il supporto di Tamarin per teorie equazionali avanzate.
- **Verifica ibrida**: combinare SPIN (per la correttezza concorrente e l'interleaving) e Tamarin (per la sicurezza crittografica simbolica) in un workflow complementare che copra sia l'aspetto comportamentale che quello simbolico.

- **Ottimizzazione di performance e scalabilità**: introdurre strategie di riduzione dello spazio degli stati in SPIN e ottimizzazioni del rewriting in Tamarin per gestire protocolli realistici con molteplici sessioni e parametri crittografici complessi.

Bibliography

DRAGONI, A. F. (2025), «Slides rilasciate a lezione», .

SPIN (2025), «Spin», URL https://spinroot.com/spin/whatispin.html.

TAMARIN (2025), «Tamarin Prover», URL https://tamarin-prover.com/documentation.html.