Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Il Metodo Induttivo e la verifica di un protocollo crittografico

M. Caiazza, G. Laccetti, G. Schmid

Le tecniche consistenti nel fare riferimento ad una serie di principi teorici e ad analisti umani per la verifica della sicurezza di un protocollo di comunicazione risultano del tutto insoddisfacenti nell'individuare potenziali nuovi attacchi; così sono stati concepiti nuovi approcci, i cosiddetti *metodi formali*, che permettono l'analisi assistita dal computer di protocolli anche complessi con metodi matematici. Il *Metodo Induttivo*, introdotto da L.C. Paulson dell'Università di Cambridge nel 1998, è un metodo formale particolarmente potente, poichè consente di considerare un numero *infinito* di agenti che eseguono il protocollo contemporaneamente. Il presente lavoro intende fornire una panoramica sul Metodo Induttivo e sulle problematiche relative alla verifica dei protocolli tramite il suo impiego, nonchè costituire un piccolo manuale sull'implementazione e l'uso del theorem prover *Isabelle.* L'impiego del metodo e del prover sono illustrati con riferimento alla verifica del protocollo di autenticazione *Woo-Lam*, consentendo di mostrare concretamente la praticabilità e le difficoltà di tale approccio.

RT-ICAR-NA-05-19 **Dicembre 2005**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*

1.Introduzione

Un algoritmo che utilizza tecniche crittografiche per assicurare uno scambio di informazioni si dice *protocollo crittografico*. La *verifica* di un protocollo crittografico, consistente nel dimostrare che il protocollo realizza effettivamente gli obiettivi di sicurezza definiti e perseguiti in fase di progetto, è un'attività non semplice e soggetta a errori di varia natura [FA'-HE-GUT- 98, RY-SCH 00].

Per molto tempo i protocolli sono stati progettati ed analizzati utilizzando quelle che ora vengono chiamate in letteratura *tecniche informali* [MEAD 91], consistenti nel fare riferimento ad una serie di principi teorici e ad analisti umani per la verifica della sicurezza di un protocollo. Le tecniche informali hanno permesso in vari casi di determinare se un dato protocollo fosse suscettibile o meno di violazione per effetto di attacchi noti, ma risultano del tutto insoddisfacenti per la verifica della sicurezza dei protocolli relativamente a nuovi attacchi. Ciò deriva, in primo luogo, dall'impossibilità per un analista umano di prevedere tutti i possibili piani d'attacco, i quali possono includere l'esecuzione simultanea di parecchie sessioni del protocollo.

In alternativa alle suddette tecniche sono stati quindi concepiti altri approcci, i cosiddetti *metodi formali* [KE 91], che permettono l'analisi assistita dal computer di protocolli anche complessi con metodi matematici. Grazie all'impiego di questi nuovi metodi sono stati rilevati alcuni difetti in protocolli già dichiarati ampiamente sicuri.

La ricerca sui metodi formali copre diversi aspetti, tra cui i linguaggi di specifica utilizzati, le tecniche per la verifica formale e, più in generale, tutti gli aspetti legati al trattamento automatico delle specifiche formali. Le diverse tecniche sviluppate sono state classificate [MEAD 91] in funzione dei diversi approcci adottati dagli analisti e del tipo di strumento utilizzato per effettuare l'analisi. Per esempio nelle *tecniche dei sistemi esperti*, l'analisi di diversi scenari di esecuzione di un protocollo è avviata attraverso un tool esplorativo che serve a determinare se ci sono dei percorsi attaccabili. L'utilizzo di strumenti automatici assistiti da computer ha permesso di individuare difetti che erano stati precedentemente trascurati dagli analisti umani. Ad esempio, l' uso del *NRL Protocol Analyzer* ha permesso di rilevare un difetto nel protocollo selettivo di radiodiffusione di Simmons [MEA 92]; mentre con l'uso del tool *Longley-Rigby* si è trovato un difetto in un protocollo di operazioni bancarie [LO-RI 92]. La scoperta e prevenzione di errori nei protocolli è quindi una delle principali motivazioni per usare i metodi formali, e l'utilizzo di un linguaggio di specifica è uno strumento indispensabile di analisi. Inoltre, se la specifica è fornita in un linguaggio di programmazione, è possibile simulare l'esecuzione del protocollo al computer, rendendo più facile la verifica delle sue proprietà. A tal proposito, negli ultimi anni si è assistito a notevoli miglioramenti nelle tecniche di verifica e sono emersi i due principali approcci, noti come *model checking* e *theorem proving* [PAU 89].

Il primo, utilizzato nelle *tecniche dell'invariante*, è servito a Lowe per individuare un attacco al protocollo di autenticazione a chiave pubblica *Needham-Schroeder* [LOWE 96 A]. La dimostrazione dell'attacco da parte di Lowe ha ispirato molti studiosi ad applicare le sue tecniche ad altri protocolli.
L'approccio theorem proving risulta ancora più duttile del model checking, in quanto non pone limiti sul formato del modello [PAU 89]. Questo motivo ha spinto L. C. Paulson a collaborare allo sviluppo di un *theorem prover* generico, *Isabelle/Isar*, per analizzare protocolli con complessità significative, attraverso una tecnica formale che si basa sul *principio d'induzione matematica* e che, per tale ragione, è nota come *metodo induttivo* [PAU 98]. Il metodo induttivo fornisce una chiarezza nell'esposizione dei risultati e nella specifica dei protocolli che deriva dalla notazione e dagli strumenti matematici messi a disposizione; tale approccio è infatti sviluppato sulla base dell' *High Order Logic* (HOL) [PAU 90]. Una caratteristica di questo metodo consiste nell'ipotizzare un numero *infinito* di agenti che eseguono il protocollo contemporaneamente; in tal modo è possibile studiare sistemi di dimensione variabile, appunto potenzialmente infiniti, e tutte le evoluzioni che tali sistemi possono avere.
Sebbene i metodi formali (ed in particolare il metodo induttivo) rappresentino un indubbio progresso nella verifica dei protocolli crittografici, essi non sono utilizzabili in tutti i casi; anzi, l'assenza di largo consenso sulle definizioni delle proprietà di sicurezza dei protocolli, rende difficile l'uso di questi approcci anche nel caso di alcuni protocolli strutturalmente molto semplici.
Il presente lavoro intende fornire una panoramica sul metodo induttivo e sulle problematiche relative alla verifica dei protocolli tramite il suo impiego, nonchè costituire un piccolo manuale sull'implementazione e l'uso del theorem prover *Isabelle.* L'impiego del metodo e del prover verranno illustrati con riferimento alla verifica di un protocollo di autenticazione strutturalmente molto semplice, il *protocollo di Woo-Lam*, consentendo di mostrare concretamente la praticabilità e le difficoltà di tale approccio.

2.Protocolli crittografici

La sicurezza, nel campo delle comunicazioni, può essere definita come l'insieme di quelle attività che permettono la protezione dei sistemi e delle reti di elaborazione attraverso procedure specifiche, in modo da controllare la fruibilità di dati e applicazioni sulla base di politiche di accesso predefinite.
La salvaguardia ed il controllo dell'informazione digitale sulle reti di comunicazione sono sovente realizzate mediante l'impiego di *protocolli crittografici.* Un *protocollo* (di comunicazione) è un algoritmo che comporta uno scambio di messaggi tra due o più parti, dette *principal*; esso è definito da una sequenza finita di passi che specificano in modo univoco le azioni richieste dai principal per

realizzare determinati obiettivi [MEN 96]. Un *protocollo crittografico* è un protocollo che utilizza tecniche crittografiche al fine di realizzare degli obiettivi di sicurezza.
Nel contesto dei protocolli di comunicazione, l'obiettivo primario di sicurezza è senza dubbio quello dell' *autenticazione* dei principal; esso è difatti alla base di tutti gli altri eventuali servizi di sicurezza offerti da un protocollo.
Un protocollo crittografico si dice *corretto* se, detti A e B - rispettivamente - gli insiemi degli obiettivi previsti ed effettivamente raggiunti dal protocollo, risulta che $A \subseteq B$.
Tutte le tecniche di verifica della correttezza di un protocollo, siano esse formali o informali, cercano di esprimere gli obiettivi di un protocollo in termini di un insieme di nozioni o proprietà di base, tra le quali – oltre alla già citata proprietà di autenticazione dei principal, se ne aggiungono altre che sono relative ai singoli messaggi trasmessi dal protocollo, quali la *riservatezza* (o *confidenzialità*), l'*autenticità,* l'*unicità* e la *non-ripudiabilità.*

2.1.Autenticazione

L'autenticazione è il processo mediante il quale gli agenti di un sistema distribuito dimostrano la propria identità. Essa, solitamente, viene realizzata facendo in modo che ogni partecipante alla comunicazione condivida un *segreto* con un'entità fidata detta *authentication server* o *trusted third party*, TTP: dimostrando di possedere questo segreto, un'agente è appunto in grado di provare la propria identità. L'autenticazione è tipicamente realizzata mediante *protocolli di autenticazione* o *protocolli per lo scambio di chiavi autenticate*, il cui scopo primario consiste - rispettivamente - nel preservare l'identità degli agenti o lo scambio di nuove chiavi segrete di sessione per future comunicazioni. Lowe [LOWE 96 b] suggerisce che i requisiti appropriati per i protocolli che realizzano autenticazione dipendono dagli scopi precisi di tali protocolli, ed identifica alcune possibili definizioni di autenticazione, incentrando il suo lavoro sul concetto di *agreement*:

- **Aliveness**. *Un protocollo P garantisce all'initiator A l'esistenza di un altro agente B se, dopo che A ha effettuato una esecuzione di P, apparentemente con B, allora B ha effettivamente eseguito una esecuzione di P.*

- **Weak agreement** *Un protocollo P garantisce ad un initiator A weak agreement con un altro agente B se, ogni volta che A completa una esecuzione di P, apparentemente con B, allora B ha effettivamente eseguito una esecuzione di P , apparentemente con A.*

- **Non-injective agreement** *Un protocollo P garantisce ad un initiator A* non-injective agreement

con un responder B su un insieme di dati d, se, ogni volta che A completa una esecuzione di P, apparentemente con B, allora:

- *B ha precedentemente eseguito P, apparentemente con A;*
- *B è un responder in tale esecuzione;*
- *i due agenti sono d'accordo nei valori di tutte le variabili in d.*

Questa definizione non garantisce che ci sia una corrispondenza biunivoca tra le esecuzioni di *A* e quelle di *B*: ad esempio, l'agente *A* potrebbe averne completate due, quando in realtà *B* ha preso parte ad una sola.

- **Injective agreement** *Un protocollo garantisce ad un initiator A* injective agreement *con un responder B su un insieme di dati d, se, ogni volta che A completa una esecuzione di P, apparentemente con B, allora:*
  - *B ha precedentemente eseguito P, apparentemente con A;*
  - *B risulta essere un responder in tale esecuzione;*
  - *i due agenti sono d'accordo sui valori di tutte le variabili in d;*
  - *ogni suddetta esecuzione di A corrisponde univocamente ad una esecuzione di B.*

Ognuno di questi quattro livelli sussume il precedente. 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 dell'esecuzione del protocollo.

La *mutua* autenticazione, in cui ciascuno dei principal in comunicazione verifica l'identità dell'altro, risulta più idonea dell'autenticazione unidirezionale per molte applicazioni distribuite. L'obiettivo primario di un protocollo di mutua autenticazione è stabilire l'identità delle parti che partecipano al protocollo, nel senso che ciascun principal deve svolgere sia il ruolo di soggetto autenticante che di soggetto autenticato. Si ottiene inoltre un obiettivo secondario, vale a dire una nuova chiave segreta per ulteriori comunicazioni fra i principal; questa chiave segreta può essere usata per l'autenticità e la segretezza di comunicazioni future. Seguendo l'approccio di Lowe [cfr. Pag. 5], queste due proprietà costituirebbero gli obiettivi da prendere in considerazione nella validazione della correttezza di un protocollo mutua autenticazione.

Comunque, contrariamente a quanto sembrerebbe suggerire l'intuizione, la verifica di un protocollo di mutua autenticazione non è riducibile a quella di due protocolli di autenticazione unidirezionale; in altri termini, per garantire una valida mutua autenticazione tra due principal A e B non basta validare separatamente l'autenticazione di A nei confronti di B e di B nei confronti di A.

La verifica di un protocollo di autenticazione è inoltre influenzata da svariati fattori, riconducibili, da un lato, ai principi-guida impiegati nella stesura del protocollo (v. §2.3) e, dall'altro, al tipo di ragionamenti operativi formali utilizzati per la verifica (v. §2.4).

2.2. Ulteriori proprietà dei protocolli crittografici

Riservatezza

Dato un messaggio m, un insieme di agenti S ed un protocollo P, si dice che P gode della proprietà di riservatezza rispetto al messaggio m ed all'insieme di agenti S se, nell'ipotesi che il messaggio m prima dell'esecuzione di P sia noto al più a tutti gli agenti appartenenti ad S, tale ipotesi risulta verificata anche dopo l'esecuzione di P.

La riservatezza di messaggio è forse, tra gli obiettivi secondari di sicurezza di un protocollo crittografico, quello cui è stata data maggior rilevanza; tant'è che i protocolli crittografici sono distinti in due classi:

- ***protocolli di prima generazione***: sono quei protocolli che hanno come unici obiettivi di sicurezza l'*autenticazione* dei principal secondo la formalizzazione di Lowe (v. §2.1) ed, eventualmente, la *riservatezza* di messaggio;
- ***protocolli di seconda generazione***: protocolli che si basano sull'impiego di protocolli di prima generazione per conseguire ulteriori obiettivi secondari di sicurezza, come l'*anonimato*, il *non-ripudio* e la *consegna certificata*.

Autenticità

Dato un messaggio m ed un protocollo P, si dice che P offre ad un agente A garanzia di autenticità rispetto al messaggio m se A è in grado di determinare chi ha generato tale messaggio.

Unicità

Dato un messaggio composto $m = \{m_1 , m_2 , m_3 , \ldots , m_n\}$, e detto I un insieme di indici tale che $I \subset \{1....n\}$, si dice che il protocollo P gode della proprietà di unicità rispetto al messaggio m e all'insieme I se, per ogni messaggio $m' = \{m'_1 , m'_2 , m'_3 , \ldots , m'_n\}$ che viene spedito sulla rete durante lo svolgimento di una sessione di P, vale il seguente predicato:

$$(i \in I \rightarrow m_i = m'_i) \quad \rightarrow \quad (i \in \{1....n\} \rightarrow m_i = m'_i)$$

In altri termini, le componenti indicate nell'insieme I caratterizzano univocamente il messaggio m. Tale insieme I è di solito costituito da un unico indice, che rappresenta un messaggio *atomico*, in grado da solo di identificare il messaggio composto. Un requisito di questo genere viene infatti utilizzato per stabilire una corrispondenza tra un messaggio inviato durante una sessione del protocollo e la sessione stessa.

Non ripudiabilità

Un protocollo P fornisce garanzia di non repudiabilità ad un agente A se assicura al suddetto agente i mezzi per dimostrare che una certa azione è avvenuta.

Supponiamo di esaminare un protocollo nel quale A invii un messaggio m a B. Una proprietà di non repudiabilità potrebbe essere: "A riesce a dimostrare la ricezione del messaggio m da parte di B".

Fairness

Si dice che P gode di fairness se per ogni garanzia G, che permetta ad un agente A di dimostrare che B ha eseguito un'azione, viene fornita a B una garanzia complementare.

La proprietà complementare, nell'esempio precedente, risulterebbe: "B riesce a dimostrare la spedizione del messaggio m da parte di A".

2.3.Principi-guida nella progettazione dei protocolli

Per quanto riguarda i principi impiegati nella stesura di un protocollo di autenticazione, vanno anzitutto ricordati il *primo principio di Abadi e Needham* ed il *principio delle informazioni complete* [AB-NE 95].

Primo Principio di Abadi-Needham *Ogni messaggio dovrebbe indicare il suo significato intrinseco, nel senso che l'interpretazione del messaggio dovrebbe dipendere soltanto dalle relative richieste da soddisfare.*

Il principio delle informazioni complete si basa sulla seguente definizione di

Protocollo di informazioni complete *Un protocollo di autenticazione P è detto protocollo di informazioni complete se i relativi initiator e responder in P includono in ogni messaggio cifrato uscente tutte le informazioni che ciascuno ha raccolto nello scambio di autenticazione.*

e costituisce una conseguenza del primo principio:

Principio delle informazioni complete *Ogni protocollo di autenticazione dovrebbe risultare un protocollo di informazioni complete.*

Dato un protocollo di autenticazione P, la versione P* di P soddisfacente il principio delle informazioni complete risulta in genere meno violabile delle varianti di P che non lo soddisfano, in quanto i messaggi in P* trasportano informazioni supplementari che, opportunamente utilizzate dai destinatari, possono ridurre la probabilità di riuscita degli attacchi. Un esempio di quanto sia determinante questo fattore per la robustezza di un protocollo crittografico, è il protocollo *Woo-Lam semplificato* [LOWE 96 B]. Questo protocollo, a differenza della versione proposta in [AB-NE 94], non soddisfa il principio delle informazioni complete, ed è soggetto ad attacchi per i quali il secondo non risulta vulnerabile [AB-NE 94, LOWE 96 B].
La semplificazione dei protocolli completi di informazioni è auspicabile soltanto se rappresenta un notevole risparmio, sia in termini di lunghezza di messaggio che di tempo di elaborazione, e deve essere effettuata con attenzione. Per ottenere miglioramenti significativi nell'efficienza, un progettista di protocollo di autenticazione dovrebbe soprattutto lavorare sulla riduzione del numero dei messaggi impiegati dal protocollo.

2.4.Metodi formali per la verifica dei protocolli

Sono molti i metodi formali oggi disponibili per l'analisi dei protocolli; tuttavia, nessuno di essi è ancora in grado di garantire un'analisi esaustiva della sicurezza, almeno per buona parte dei protocolli di impiego reale [MEA 03]. I motivi di ciò sono molteplici; essi sono riconducibili sia alla natura stessa del problema della validazione dei protolli, che ai limiti dell'approccio formale in generale, che a limiti più specificatamente ascrivibili ai metodi finora introdotti e al loro stadio di maturità.
In primo luogo, va ricordato che quello della sicurezza dei protocolli crittografici è, nella sua generalità, un problema *indecidibile* [EV-GO 83]; ciò comporta che, in ogni caso, nessun metodo avrà successo nella verifica di ogni possibile protocollo, e che sarà sempre necessario l'intervento di un analista umano.
Per quanto riguarda i limiti dell'approccio formale in generale, è poi importante sottolineare che i metodi formali, per loro natura, consentono solo descrizioni ad alto livello dei protocolli, tralasciando quindi di considerare tutti gli aspetti di dettaglio a livello implementativo, che spesso però giocano un ruolo determinante nella sicurezza complessiva di un sistema.
Naturalmente, l'aspetto di maggiore interesse per chi utilizza i metodi formali o ne fa oggetto di studio è quello relativo ai limiti dei metodi finora introdotti e del loro stadio di maturità.
A tal proposito, va osservato anzitutto che esistono ancora divergenze e disomogeneità significative,

sia fra le nozioni formali ed informali di correttezza di un protocollo, che fra le nozioni formali relative a metodi diversi.

Per esempio, alcuni approcci [DO-YAO 83, KE 91] considerano la segretezza come principale elemento di test per la verifica di un protocollo, senza specificare alcune proprietà supplementari che derivano da questa; altri [BU-AB-NE 90] specificano la correttezza di un protocollo in termini di "state of belief", e considerano implicite le proprietà di segretezza.

In secondo luogo, anche nei metodi formali più attuali, la nozione ivi utilizzata di correttezza di un protocollo può dar luogo ad ambiguità (con conseguente difficoltà di analisi) o, ancora peggio, a falsi negativi (cioè far ritenere corretto un protocollo che invece non risulta esserlo).

Un esempio di ambiguità è fornito dalla formula "P <--K --> Q" della costruzione della logica di BAN*, in quanto - come rilevato in [NES 90] – essa i riferisce sia alla proprietà di riservatezza che alla distribuzione di chiavi.

Come vedremo, il metodo induttivo [PAU 98], seppur ritenuto uno dei migliori metodi formali proprio per la sua genericità, attesta la correttezza di un protocollo solamente rispetto alle due proprietà di riservatezza ed autenticità, tralasciandone altre. Ciò comporta lo "stallo" del theorem prover Isabelle nel caso dell'analisi di un protocollo strutturalmente semplicissimo quale quello già citato di Woo-Lam. Ritorneremo su questo punto in seguito, allorchè nel §5 illustreremo in dettaglio il processo di verifica di tale protollo mediante il metodo induttivo.

Ciò che emerge dalle precedenti considerazioni è la necessità di disporre, se non di un'unica, omnicomprensiva nozione di correttezza formale di un protocollo, almeno di nozioni complementari, in modo tale che più metodi formali possano essere utilizzati sinergicamente per l'analisi di quei protocolli che richiedono la validazione di proprietà di sicurezza che, nel loro insieme, non sono verificabili con un singolo metodo.

3.Il metodo Induttivo e *Isabelle*.

L'efficacia di un modello formale è principalmente misurata dai due seguenti fattori:

- Il grado di vicinanza che tale modello offre rispetto alla realtà;
- La semplicità e la flessibilità della tecnica d'analisi messa da esso a disposizione.

Un modello formale ideale deve quindi essere in grado di analizzare congetture sufficientemente complesse, attraverso tecniche di verifica che siano maneggevoli, perché convertire manualmente un protocollo concreto in una specifica convenzionale risulta essere un processo critico che in sé potrebbe essere incline ad errori difficilmente risolvibili.

* Teoria logica introdotta nel 1989 da Burrows, Abadi e Needham [BU-AB-NE89] e largamente utilizzata per l'analisi dei protocolli di autenticazione, il cui nome deriva dalle iniziali degli autori.

Il metodo induttivo di Paulson [PAU 98], pur fondandosi sulla logica dei predicati (in particolare, sull'*higher-order logic*), adotta un approccio completamente diverso da quello dei metodi formali basati su logiche modali quali la BAN [BU-AB-NE89]: mentre queste ultime utilizzano dimostrazioni brevi, col preciso intento di minimizzare l'errore umano, la verifica induttiva di un protocollo crittografico è lunga e dettagliata, e richiede l'utilizzo di strumenti automatici assistiti da computer per poter essere condotta.

"*Ogni proprietà di sicurezza è dimostrata per induzione, ogni caso considera una condizione del sistema raggiungibile in un preciso passo del protocollo. La semplificazione delle proprietà di sicurezza potrebbe, in alcuni casi, rivelare una combinazione delle circostanze che conducono alla relativa violazione. Soltanto se tutti i casi fossero esplicitati, la dimostrazione si potrebbe dire completa*" [PAU98].

Paulson ha sintetizzato in tre punti principali i problemi da risolvere per un'adeguata formalizzazione dei protocolli di seconda generazione (v. §2.2):

- **Obiettivi fondamentali.** Le principali proprietà che devono essere garantite da un protocollo crittografico di prima generazione restano tali anche per i protocolli di seconda generazione perché, come abbiamo già detto, questi ultimi contano sull'impiego di protocolli di sicurezza di livello inferiore per conseguire obiettivi più specifici, come l'anonimato, il non-ripudio e la consegna certificata.

- **Modello di minaccia.** Il model threat campione per i protocolli di autenticazione è quello del protocollo Dolev-Yao [DO-YAO 83]: un singolo attaccante (`Spy`) controlla l'intera rete, avendo la capacità di intercettare qualsiasi messaggio e di riutilizzare i messaggi intercettati, ma non può violare i messaggi cifrati. Per i protocolli di seconda generazione, il modello di minaccia formalizzato da Paulson è ancora una spia di Dolev-Yao: essa intercetta il traffico di rete attraverso l'operatore *Know*, lo analizza attraverso l'operatore *Analz* e lo sintetizza attraverso l'operatore *Sinth*. In questo modo la spia può trasmettere un messaggio compiuto *M* ad un qualsiasi agente *B,* che è una variabile indipendente, e tutto questo è formalizzato da una particolare regola, la regola `Fake`, che introduce l'evento `Says Spy BM` (si legge *Spy Says m to B*) nella traccia corrente. É importante sottolineare che tale modello di minaccia non interferisce con gli obiettivi già previsti e verificati per i protocolli di livello inferiore; in altri termini, la spia del model threat proposto da Paulson non interferisce nelle comunicazioni che sono già assicurate dai protocolli di autenticazione di primo livello.

- **Nuovi obiettivi.** La verifica degli obiettivi di sicurezza specifici dei protocolli di seconda generazione, secondo Paulson, richiede anzitutto una loro formalizzazione. Ad esempio, la distribuzione delle chiavi ad una coppia di agenti può essere formalizzata da una regola induttiva che dà ad entrambi gli agenti, attraverso un evento denominato `Notes`, una chiave appositamente generata (generazione di elementi nuovi). Un ulteriore esempio è fornito dalla formalizzazione dell'obbiettivo di *consegna certificata*: si introduce un evento `Gets` per la ricezione di un messaggio m, nello stesso tempo in cui si introduce l'evento `Says` di invio del messaggio. Quindi non è necessaria la regola `reception` che scinde i due eventi `Says` e `Gets`. L'obbiettivo è espresso su una generica traccia `evs` di eventi che derivano dalla relativa esecuzione di un numero *infinito* di agenti. Il modello non obbliga gli agenti a rispondere ad alcun messaggio: essi possono rispondere in ritardo, o rispondere più di una volta, o non rispondere affatto. Sono previsti anche funzionamenti "intrecciati", in modo che gli agenti possano rispondere ai vecchi messaggi.

Alla luce di queste osservazioni, Paulson propone come tool esplorativo il *theorem prover Isabelle* ([PAU 90]) anziché un *model checker*, in quanto il primo non pone limiti sul formato del modello. In questo modo Paulson può formalizzare la caratteristica più innovativa del suo modello: il fatto che esso contempli la possibilità che un numero non limitato, potenzialmente infinito di agenti esegua il protocollo contemporaneamente.

Isabelle è un generico theorem prover progettato per il ragionamento interattivo in una vasta gamma di teorie formali. Esso fornisce le procedure utili a svariate teorie dimostrative: la *teoria degli insiemi* di Zermelo-Frankel [SUP 72, NÖ 93], la *logica dei predicati di primo ordine*, *l'higher-order logic* (HOL)[GO 88], fino alla più recente *Constructive Type Theory*, introdotta da Martin Löf nel 1984 [Löf 84]. La genericità di Isabelle può apparire insensata: se un theorem prover a logica fissa è già considerato difficile, perché con Isabelle si complicano le cose lasciando variare la logica? La risposta è che la maggior parte delle difficoltà sollevate dall'utilizzo delle logiche formali riguardano proprio il *tipo di logica da impiegare* per risolvere un dato problema. L'idea alla base di Isabelle è quella di utilizzare una logica generica per tutto il corso della verifica, e di operare separatamente su parti specifiche con delle logiche individuali. Il sistema risultante può essere molto più potente di un theorem prover specifico, ed è anche più flessibile.

Il metodo induttivo prevede, altre alla nozione di *agente*, poche altre nozioni di base: *chiave*, *messaggio*, *evento* e *traccia*. A ciascuna di queste nozioni corrisponde, nel linguaggio del theorem prover Isabelle, un tipo di dato con eventuali sottotipi.

3.1.Agenti

L'insieme degli agenti è costituito da :

- Una terza parte fidata `TTP`, che si presuppone onesta e che nello svolgimento del protocollo agisce, solitamente, sia da tramite che da garante;
- Un insieme infinito `Friend` di agenti in grado di eseguire il protocollo, ciascuno dei quali è supposto onesto ed è individuato da un numero naturale `nat`;
- Una Spia `Spy` che, a differenza degli agenti onesti, ha il potere di intercettare il traffico che non le è diretto e di sintetizzare ed inviare messaggi non previsti dal protocollo;
- Un insieme di agenti "compromessi" `Bad`, che cioè inavvertitamente (e in buona fede) hanno permesso alla spia di impadronirsi di una chiave segreta in loro possesso. Il modello non distingue un agente compromesso da un altro, cosicchè `Bad` va considerato un'unico elemento, al pari di `Spy` e di `TTP` .

Nella sintassi di Isabelle l'istruzione dichiarativa per il tipo agente è data da:

```
Datatype
agent = TTP | Friend nat | Spy |Bad
```

dove | denota l'operatore logico OR.

3.2.Chiavi

Si suppone che gli agenti siano in grado di utilizzare un qualsiasi algoritmo crittografico, ciascuno dei quali è considerato sicuro. Viene quindi introdotto un insieme infinito e numerabile di coppie di chiavi, `priEkA` (*Pri*vate *E*ncryption *K*ey of *A*) e `priSkA` (*Pri*vate *S*ignature *K*ey of *A*), che rappresentano le chiavi private - rispettivamente di cifratura e di firma - dell'agente A . Analogamente, `pubEkA` (*Pub*lic *E*ncryption *K*ey of *A*) e `pubSkA` (*Pub*lic *S*ignature *K*ey of *A*) rappresentano le rispettive chiavi pubbliche dell'agente A, mentre `shrKA` (*Shar*ed *K*ey of *A*) rappresenta la chiave simmetrica che A condivide con il server TTP. In genere, applicare una cifratura con chiave k, richiede la conoscenza dell'inversa di tale chiave per decifrare l'informazione criptata: la funzione `invKey` lega ogni chiave alla sua inversa. Tale funzione gode della proprietà riflessiva che, espressa nel linguaggio del theorem prover, si scrive:

```
Consts
invKey :: "key => key"
axioms
invKey [simp] : "invKey (invKey K) = K"
```

dove `Consts` rappresenta la dichiarazione delle costanti da utilizzare ed `axioms` quella degli assiomi da inserire durante la verifica.
La parola `simp` tra parentesi quadre sta ad indicare la *regola simp*, una regola dimostrativa induttiva utilizzata dal theorem prover per provare che $(k^{-1})^{-1}=k$.
Un sottinsieme dell'insieme delle chiavi è quello delle chiavi simmetriche `symKeys`: una chiave k è detta simmetrica se è uguale alla sua inversa.

```
Constdefs
symKeys :: "key set"
"symKeys == {K. invKey K = K}"
```

Ovviamente, il complemento di `symKeys` rispetto all'insieme di tutte le chiavi rappresenta l'insieme delle chiavi asimmetriche.

3.3.Messaggi

I messaggi sono composti a partire da *elementi* o *messaggi atomici*, combinati mediante alcuni operatori. Si distinguono i seguenti tipi di messaggi atomici:

- `Number i` , dove `i` denota un numero naturale. Generalmente adoperiamo un numero per rappresentare un'informazione che non si vuole tenere nascosta, o che un attacker è in grado di individuare facilmente;
- `Nonce i`, con `i` numero naturale. A differenza dei numeri, le Nonces sono dei messaggi che non possono essere facilmente individuati, tipicamente numeri (pseudo)*casuali*;
- `Agent A`, dove `A` appartiene all'insieme degli agenti. Tale messaggio rappresenta l'identificativo dell'agente descritto;
- `Key k`, con `k` appartenente all'insieme delle chiavi;

Particolari messaggi atomici sono poi i messaggi *hash* ed i messaggi *cifrati*:

- `Hash X` denota il risultato (messaggio hash) dell'applicazione di una funzione hash crittografica al messaggio X;
- `Crypt kX` indica invece il messaggio ottenuto applicando la funzione di cifratura `Crypt` con chiave `k` al messaggio X.

In entrambi i casi `X` può rappresentare indifferentemente un messaggio atomico o composto.

I messaggi *composti* sono messaggi formati da due o più messaggi atomici X_i (i =1 ,..., n), che si dicono suoi *componenti*. Un messaggio composto viene costruito iterativamente a partire da coppie dei suoi componenti per il tramite dell'operatore `Mpair`: `MpairX`$_1$`.....(MpairX`$_{n-1}$`, X`$_n$`)`. Alla notazione `MpairX`$_1$`, X`$_2$ è spesso preferita l'alternativa `{|X`$_1$`, X`$_2$`|}`.

L'istruzione dichiarativa seguente stabilisce che un messaggio può essere costituito da uno qualunque dei suddetti messaggi atomici, oppure composto ricorsivamente a partire da questi.

```
Datatype
   msg = Agent  agent
         |  Number nat
         |  Nonce  nat
         |  Key    key
         |  Hash   msg
         |  MPair  msg msg
         |  Crypt  key msg
```

I sette campi definiti nella precedente dichiarazione rappresentano i *costruttori principali* di un messaggio; i primi quattro si dicono *costruttori primari*.

3.4.Eventi

I singoli eventi che possono verificarsi sulla rete sono di tre tipi: l'evento `Says` indica l'invio di un messaggio, l'evento `Gets` indica la ricezione di un messaggio da parte del destinatario e l'evento `Notes` indica la conservazione di un messaggio al fine di un suo utilizzo successivo. Essi vengono formalizzati nel seguente modo:

```
datatype

event = Says agent agent msg
      | Gets agent       msg
      | Notes agent      msg
```

- `SaysAB X` indica l'invio di un messaggio X da parte dell'agente A all'agente B;
- `GetsB X` indica la ricezione del messaggio X da parte dell'agente B;

- `NotesA Y` indica il fatto che il messaggio Y viene conservato dall'agente A, in modo che possa essere riutilizzato in seguito.

Scindere invio e ricezione permette di studiare situazioni nelle quali i messaggi possono perdersi, come d'altronde avviene sulla reti reali.

3.5.Tracce

Una sequenza di eventi viene definita *traccia*: essa registra tutti quegli eventi accaduti durante una sessione del protocollo. Le tracce sono il parametro di riferimento per costruire un modello di protocollo e per verificarne le relative proprietà: ogni traccia rappresenta una possibile esecuzione del protocollo, e la validità di una proprietà per il protocollo consiste nel fatto che tale proprietà è verificata in ogni possibile traccia.

Se T è una traccia, T si compone di un insieme finito di messaggi, ciascuno dei quali è a sua volta un insieme finito di componenenti elementari o messaggi atomici.

3.6.Conoscenza degli agenti

La proprietà di riservatezza implica che solamente gli agenti coinvolti in una sessione del protocollo sono a conoscenza del traffico di messaggi: occorre formalizzare questa conoscenza degli agenti. Per indagare sulla conoscenza degli agenti si utilizza un particolare operatore, l'operatore ***knows***. Tale operatore adotta come argomenti un agente ed una traccia, e restituisce un insieme (non ordinato) di messaggi, che vengono ricavati sia da quelli che compaiono sulla traccia, sia da quelli contenuti nello stato iniziale relativo all'agente:

`knows : agent → event list → msg set`

Per esprimere invece la conoscenza della spia, che si esplica nella conoscenza degli eventi contenuti in una traccia, si introduce un ulteriore operatore simile a `knows`, l'operatore `spies:`

`spies : event list → msg set`

```
consts
 knows :: "agent =>  event list => msg set"

 spies :: "event list => msg set"
```

Occorre aggiungere che la spia, che deve poter monitorare tutto il traffico, immagazzina tutti i messaggi inviati sulla rete tramite un evento `Says` ‡:

‡ Il simbolo "#" indica l'operazione di concatenazione.

```
spies ((SaysAB X) # evs) = {X} ∪ spies evs
```

Si noti che `spies evs` contiene l'intero traffico che si presenta sulla rete durante la registrazione di una storia attraverso un evento `evs`. Di conseguenza, con un abuso di terminologia, `spies evs` è riferito spesso come `traffic on evs`.

I messaggi che un agente apprende dalla traccia sono invece quelli che ha ricevuto, attraverso `Gets`, o quelli che egli stesso ha elaborato, attraverso `Notes`.

```
Primrec

knows_Nil: "knows A [] = initState A"
knows_Cons:
  "knows A (ev # evs) =
    (if A = Spy then
      (case ev of
        Says A' B X => insert X (knows Spy evs)
      | Gets A' X => knows Spy evs
      | Notes A' X =>
         if A' = Spy then insert X(knows Spy evs)
         else knows Spy evs)
     else
      (case ev of
        Says A' B X =>
         if A' = A then insert X (knows A evs)
         else knows A evs
      | Gets A' X =>
         if A'=A then insert X (knows A evs)
         else knows A evs
      | Notes A' X =>
         if A'=A then insert X (knows A evs)
         else knows A evs))"
```

Per tener contodella differenza tra la conoscenza iniziale degli agenti, della TTP e della spia, Paulson introduce la funzione :

```
initState: agent → msg set
```

che, preso in input un agente un agente, restituisce l'insieme di messaggi noti all'agente stesso prima dell'esecuzione del protocollo.

La conoscenza iniziale degli agenti onesti consiste delle loro rispettive chiavi condivise; cioè:

```
initState_Agent = Key (shrK (Agent A))
```

La conoscenza iniziale della terza parte fidata consiste di tutte le chiavi condivise con gli agenti (cioè, tutti segreti di lunga durata), ed è espressa come:

```
initState_TTP = Key (shrK Agent A)
```

Infine, la conoscenza iniziale della spia consiste invece delle chiavi di tutti gli agenti compromessi :

```
initState_Spy = (Key (shrK Agent A) | A ∈ bad)
```

```
Primrec
initState_TTP: "initState TTP = insert (Key (shrK TTP))"
initState_Agent:"initState (Agent A)=insert(Key (shrK Agent A ))"
initState_Spy: "initState Spy =insert (Key (shrK Spy))"
```

3.7.Operatori su insiemi di messaggi

L'operatore *Parts*

Sia M un insieme di messaggi. Si dice *insieme delle parti di M* e si denota con `PartsM` l'insieme dei messaggi m, tale che m appartiene ad M, oppure m è una componente di un messaggio in M, o ancora m è la decodifica di un crittogramma in `PartsM`.

Quindi l'insieme `PartsM` è ottenuto da M aggiungendo ripetutamente i componenti dei messaggi composti ed i testi dei messaggi cifrati (in genere la chiave k che cifra un messaggio non è considerata come componente del messaggio `Crypt kM` a meno che essa non faccia parte di M).

Esempio:

Consideriamo i messaggi atomici `X` e `Cript kX` ed il singolo messaggio composto M =`{|X, Crypt kX|},` e calcoliamo l'insieme `Parts M,` esprimendolo in metalinguaggio. Agiamo con l'operatore `Parts` sul messaggio M, ricordando che, se A e B denotano due proposizioni, con $\frac{A}{B}$ si intende che A implica B:

$$\frac{X \in \{|X, Crypt\ kX|\}}{X \in Parts\{|X, Crypt\ kX|\}} \qquad \frac{Crypt\ kX \in Parts\{|X, Crypt\ kX|\}}{X \in Parts\{|X, Crypt\ kX|\}}$$

$$\frac{\{|X, Crypt\ kX|\} \in Parts\{|X, Crypt\ kX|\}}{X \in Parts\{|X, Crypt\ kX|\}} \qquad \frac{\{|X, Crypt\ kX|\} \in Parts\{|X, Crypt\ kX|\}}{Crypt\ kX \in Parts\{|X, Crypt\ kX|\}}$$

Ciò può essere espresso nel linguaggio del thorem prover Isabelle nel modo seguente:

```
m ∈ M                 →  m  ∈ parts(M)
Crypt km ∈ parts(M)   →  m  ∈ parts(M)
{m1, m2} ∈ parts(M)   →  m1, m2 ∈ parts(M)
```

L'operatore *Parts* è utilizzato per indagare la presenza o meno di opportuni elementi sulla traccia; i cosiddetti lemma di regolarità trattano questo tipo di proprietà.

L'operatore *Analz*

Sia M un insieme di messaggi. Si dice *insieme delle parti analizzate di M* e si denota con `Analz M` l'insieme costituito da tutti i messaggi in M, dalle parti dei messaggi di M e, da tutti i testi in chiaro relativi a crittogrammi e chiavi di decifratura contenuti in M. Tale operatore viene comunemente applicato all'insieme dei messaggi che un agente ha appreso da una traccia; in tal modo siamo in grado di conoscere non solo i messaggi che sono stati inviati all'agente, o che questo è stato in grado di intercettare, qualora si tratti della spia, ma cosa egli è in grado di dedurre da tali messaggi. Resta inteso quindi che la definizione di `Analz` è più restrittiva di quella di `Parts`; invero, si ha che `Analz M` $\subseteq$ `Parts M`. In particolare, `Analz M` = `Parts M` se la chiave k risulta essere una chiave simmetrica, in quanto, in tal caso $k = k^{-1}$ e l'ipotesi aggiuntiva su `Analz` non comporta ulteriori restrizioni.

Esempio:

Consideriamo lo stesso messaggio composto M = {|X, Crypt kX|} dell'esempio precedente e calcoliamo l'insieme *Analz*:

$$\frac{X \in \{|X, Crypt\ kX|\}}{X \in Analz\{|X, Crypt\ kX|\}} \qquad \frac{Crypt\ kX \in Analz\{|X, Crypt\ kX|\} \quad k^{-1} \in Analz\{|X, Crypt\ kX|\}}{X \in Analz\{|X, Crypt\ kX|\}}$$

$$\frac{\{|X, Crypt\ kX|\} \in Analz\{|X, Crypt\ kX|\}}{X \in Analz\{|X, Crypt\ kX|\}} \qquad \frac{\{|X, Crypt\ kX|\} \in Analz\{|X, Crypt\ kX|\}}{Crypt\ kX \in Analz\{|X, Crypt\ kX|\}}$$

Riscriviamo quanto sopra nel linguaggio di Isabelle:

```
m ∈ M                                  →  m ∈ analz(M)
Crypt km ∈ analz(M),Key k ∈ analz(M)   →  m ∈ analz(M)
{m1, m2}  ∈ analz(M)                   →  m1,m2 ∈ analz(M)
```

L'operatore *Synth*

Sia M un insieme di messaggi. Si dice *insieme delle parti sintetizzate di M*, e si denota con `Synth M,` l'insieme ottenuto da M considerando tutti i messaggi che vi appartengono, tutte le parti che li compongono e, per ogni chiave di cifratura, tutti i crittogrammi ottenuti da questa a partire da

messaggi in chiaro. L'insieme `Synth M` è dunque il più piccolo insieme contenente M, i nomi degli agenti ed i numeri casuali, ed è chiuso rispetto ai messaggi composti, cifrati ed i messaggi hashed. Tale operatore viene di solito utilizzato per indagare sulle potenzialità di un agente. L'insieme `Synth (analz(knows A evs))`contiene infatti tutti gli elementi che un agente è in grado di sintetizzare, e quindi potenzialmente di spedire, a partire dalle informazioni che riesce a dedurre dalla traccia `evs`. Investigando sulla natura di tale insieme si riesce in genere a capire ad esempio quali messaggi un agente sia in grado di falsificare e quali no. Sia ad esempio `h=Hash{X; Y}`; se vale la proprietà :

`h ∈ analz(knows A evs) → analz(knows A evs)`

allora l'agente A non è in grado di sintetizzare ex novo l'hash in questione, ma può solo apprenderlo dalla traccia.

Esempio:

Consideriamo il messaggio composto `{|X, Crypt kX|}` ed i messaggi atomici `Agent A` e `Number N.` L'operatore `Synth` agisce in questo modo:

$$Agent\ A \in Synth\{|X, Crypt\ kX|\} \qquad Number\ N \in Synth\{|X, Crypt\ kX|\}$$

$$\frac{X \in \{|X, Crypt\ kX|\}}{X \in Synth\{|X, Crypt\ kX|\}}$$

$$\frac{X \in Synth\{|X, Crypt\ kX|\} \quad Crypt\ kX \in Synth\{|X, Crypt\ kX|\}}{\{|X, Crypt\ kX|\} \in Synth\{|X, Crypt\ kX|\}}$$

$$\frac{X \in Synth\{|X, Crypt\ kX|\} \quad k \in \{|X, Crypt\ kX|\}}{Crypt\ kX \in Synth\{|X, Crypt\ kX|\}}$$

che riscritto in Isabelle diventa:

```
m ∈ M                          →          m ∈ sinth(M)
m ∈ sinth(M), Key k ∈ sinth(M) → Crypt k m ∈ sinth(M)
m ∈ sinth(M)                   → Hash(m)   ∈ sinth(M)
m1,m2 ∈ sinth(M)               → {m1, m2}  ∈ sinth(M)
```

Un ulteriore operatore, introdotto per rappresentare la nozione di *attualità di una traccia*, è `used` [BELLA 00], così formalizzato:

`used : event list → msg set`

Un messaggio *m* è considerato nuovo su una traccia se non compare mai come componente dello stato iniziale di qualsiasi agente o di un messaggio registrato attraverso la traccia. Le proprietà

formali di `used` sono le seguenti:

```
used [ ] = ∪B. parts (initState B )
used((Says A B  X) # evs) = parts{X} ∪ used evs
used((Notes A X) # evs) =  parts{X} ∪ used evs
```

3.8.Formalizzazione di un protocollo

Un protocollo viene formalizzato utilizzando una serie di regole induttive che descrivono la struttura delle tracce. Il caso base è rappresentato dalla traccia vuota; ogni passo successivo viene formalizzato da una regola induttiva.Ciascuna regola condiziona come estendere una data traccia di un insieme attraverso tutti gli eventi possibili, in modo che il protocollo possa trasmettere un nuovo messaggio. Per esempio, se un protocollo consta di quattro passi, avremo quattro regole induttive [BELLA 00].
Supponiamo di analizzare un protocollo P che contenga, al passo ennesimo, l'invio di un messaggio Y condizionato dalla ricezione del messaggio X. Esaminando tutte le possibili tracce costruite a partire dalle regole induttive che descrivono un protocollo, otteniamo tutti i possibili scenari (ovviamente infiniti) nei quali una popolazione infinita di agenti esegue una, nessuna o tante sessioni del protocollo. Supponiamo, ad esempio, di esaminare il seguente protocollo :

```
Un agente A invia ad un'altro agente B il messaggio "Ciao, com'è
andato l'esame?".
L'agente B, alla ricezione del messaggio, puo' rispondere "bene"
oppure "male".
```

Sia T l'insieme di tutte le possibili tracce del protocollo in questione, allora T gode delle proprietà seguenti:

```
[] ∈ T

trace ∈ T →Says AB "Ciao sono A, com'è andato l'esame?"
            # trace ∈ T

trace ∈ T →Gets B "Ciao sono A, com'è andato l'esame?"
            #  Says BA "Bene" # trace  ∈ T

trace ∈ T → Gets B "Ciao sono A, com'è andato l'esame?"
            # Says BA "Male" # trace ∈ T
```

Secondo questa definizione, sia la traccia nella quale B risponde "Bene", sia quella nella quale risponde "Male", sono tracce per il protocollo P. Possono essere quindi studiati entrambi i casi in questione, e tutti gli altri casi previsti dalla definizione induttiva (B non risponde, B risponde due volte, A manda due volte il messaggio…).

Oltre alle regole che descrivono i passi del protocollo, devono esere inserite altre regole di natura generale, per descrivere il funzionamento della rete sottostante.

La più importante di queste è la regola `reception`: "Se l'evento r† appartiene al protocollo P e se un agente A invia il messaggio X ad un agente B, allora B riceve il messaggio".

Nel linguaggio di Isabelle la regola suddetta si scrive:

```
[| evsr ∈ P ; Says A B X ∈ set evsr |] → Gets B X  # evsr ∈ P
```

Tale regola formalizza una situazione nella quale, se un messaggio è stato spedito, esso *può o meno* giungere a destinazione. Serve quindi a scindere l'invio di un messaggio dalla sua ricezione, ed a formalizzare il fatto che i messaggi possono anche perdersi. Scindere l'invio dalla ricezione rende di certo più realistico il modello di una rete inaffidabile. Inoltre l'esperienza insegna che molti attacchi sfruttano proprio questo fatto. Altre regole ausiliarie riguardano le potenzialità aggiuntive della spia, come la possibilità di conoscere messaggi inviati su canali convenzionali, anche se non sono indirizzati a questa.

Tale modello basato invece non offre alla spia la possibilità di inviare messaggi arbitrari, ossia non previsti nel protocollo. A tale scopo viene introdotta la regola induttiva `Fake` :

```
[|evs∈P ...; X∈synth(analz(knows Spy evsfssl))|] → SaysAB X  #  evs ∈ P
```

Come già detto, tale regola permette alla spia di violare il protocollo, ove ciò sia possibile. Essa inoltre permette alla spia di interporsi tra due agenti durante una comunicazione. Supponiamo infatti che due agenti, A e B, stiano eseguendo una sessione di un protocollo. Tale protocollo prevede l'invio da parte di A di un certo messaggio m a B. Tuttavia tale messaggio si perde.

La regola `Fake` permette alla spia di inviare un qualsiasi altro messaggio a B, e B in generale non è in grado di decidere se il messaggio che riceve proviene da A o dalla spia.

Infine, per modellare protocolli di distribuzione di chiavi, è introdotta la regola `Oops`, che formalizza l'acquisizione da parte della spia di una chiave segreta relativa ad in dato agente, che pertanto risulta compromesso. Ciò permette di studiare in quale misura delle inadempienze nella gestione delle chiavi crittografiche possano inficiare la sicurezza di un protocollo.

3.9.Principi di induzione

Le proprietà di un insieme definito induttivamente sono stabilite da opportune regole di induzione. Ogni proprietà deve essere verificata rispetto a tutte le regole che definiscono il suddetto insieme; ciò genera un lungo lavoro di analisi, che Isabelle effettua automaticamente ed efficientemente. Quando il

---

†La lettera r sta per "reception", ed indica che stiamo considerando un evento legato alla ricezione di un messaggio.

theorem checker non riesce ad effettuare la verifica di una proprietà, allora ciò significa che il protocollo cui la proprietà si riferisce contiene un difetto, a meno che la sua descrizione formale non sia stata effettuata scorrettamente. Le classi di regole induttive da considerare sono cinque:

- **Proprietà di possibilità**

Indicano l'esistenza di tracce che si estendono per tutto il protocollo. Non siamo in grado di provare che un evento debba accadere, in quanto gli agenti non sono costretti a comportarsi in un certo modo; se però il protocollo non procede ad un certo punto della prova, nel senso che non è possibile giungere all'ultimo passo da dimostrare, allora esso non può considerarsi corretto.

- **Lemmi di spedizione**

Indicano la spedizione di un messaggio (o di una sua componente) che non può essere decifrato. Esprimeno formalmente il fatto che la spia non otterrà nulla dall'elemento in questione.

- **Lemmi di regolarità**

Indicano che un determinato messaggio o sua componente X non appartiene all'insieme dei messaggi disponibili alla spia. Questo significa che la spia non può ottenere mai la hold di X.

- **Teoremi di autenticità**

Indicano che alcune informazioni importanti come le chiavi di sessione ed i timestamps identificano univocamente il messaggio d'origine. Forniscono le garanzie circa i messaggi cifrati dalle chiavi private.

- **Teoremi di segretezza**

Sono le garanzie più complesse, e sono espresse attraverso l'operatore Analz. É importante dichiarare che non deve esistere nessuna chiave di sessione utilizzata per cifrare altre chiavi. Questo è utile ad evitare che la spia si impadronisca di una chiave di sessione per apprenderne altre. Se il server TTP distribuisce una chiave di sessione a due agenti che non hanno perso le loro chiavi private, e se la chiave di sessione non è ottenibile dalla spia, allora nessun altro agente può ottenerle.

4.Il protocollo Woo-Lam

Il protocollo Woo-Lam [W-L 92] è basato sulla crittografia a chiave simmetrica, e punta all'autenticazione di un initiator A nei confronti di un responder B, grazie all'aiuto di una terza parte fidata S. Nella sua versione originale, esso consta di cinque scambi di messaggi:

Protocollo Woo-Lam (versione originale)	
1) $A \rightarrow B$: A	L'initiator A invia la sua identità al responder B
2) $B \rightarrow A$: N_b	B risponde spedendo il nonce N_b
3) $A \rightarrow B$: $\{\|N_b\|\}_{KaS}$	A re-invia N_b a B, cifrato con la chiave K_{aS} che condivide con S
4) $B \rightarrow S$: $\{\|A, \{\|N_b\|\}_{KaS}\|\}_{KbS}$	B invia ad S l'id di A e il msg (3), cifrati con la chiave K_{bS}
5) $S \rightarrow B$: $\{\|N_b\|\}_{KbS}$	S invia a B N_b, cifrato con la chiave K_{bS} che condivide con B

L'autenticazione di A nei confronti di B ha successo solo se la decifratura del messaggio inviato a B da S restituisce il nonce originario; essa si basa sulle due proprietà di *corrispondenza* e di *riservatezza*. In realtà la proprietà di corrispondenza è indirizzata all'autenticazione vera e propria, mentre quella di riservatezza si riferisce particolarmente al problema di distribuzione delle chiavi.

4.1.Attacchi al protocollo

Il protocollo è immune ad un *attacco di impersonazione* di A:

I ATTACCO : Impersonazione di A	
1) $A' \rightarrow B$: A	A è in DoS ed A' si spaccia per lui con B
2) $B \rightarrow A$: N_b	B risponde inviando un nonce che è intercettato da A'
3) $A' \rightarrow B$: $\{\|N_b\|\}_{K'}$	A' può condividere o meno la chiave K' con S
4) $B \rightarrow S$: $\{\|A, \{\|N_b\|\}_{K'}\|\}_{KbS}$	B invia il nonce cifrato e l'identità di A al server S
5) $S \rightarrow B$: $\{\|N_b\|\}_{K' \circ KaS \circ KbS}$	Il server S invia il responso a B

Quando B decifra con la sua chiave il nonce inviato dal server, ottiene $\{\|N_b\|\}_{K' \circ KaS} \neq N_b$, per cui rifiuta A come autentico.

Il protocollo è immune anche ad un attacco di impersonazione di S:

II ATTACCO : Impersonazione di A e del server S	
1) $A' \rightarrow B$: A	Impersonazione di A da parte di A'
2) $B \rightarrow A$: N_b	A è in DoS ed A' riceve il nonce al suo posto
3) $A' \rightarrow B$: $\{\|N_b\|\}_{K'}$	A' può condividere o meno la chiave K' con S
4) $B \rightarrow S$: $\{\|A, \{\|N_b\|\}_{K'}\|\}_{KbS}$	B invia il nonce cifrato e l'identità di A al server S
5) $A' \rightarrow B$: $\{\|N_b\|\}_{H'}$	S è in DoS ed A' invia il responso al posto suo a B

Invero, se $H' \neq K_{bS}$ allora risulta che $\{\|N_b\|\}_{K' \circ KaS} \neq N_b$ e B rifiuta A come autentico.

Tuttavia Abadi e Needham, attraverso una verifica formale basata sulla BAN logic [BU-AB-NE 90], hanno dimostrato che il protocollo non attesta a B la garanzia più forte [AB-NE 94], cioè la non-

injective agreement (§2.), che assicurerebbe che è stato effettivamente A a stabilire una sessione del protocollo con B. Il motivo è che il "collegamento" fra i messaggi non è sufficiente. In particolare, nulla collega la domanda del responder B al server S con la successiva risposta di S, come mostra il seguente attacco a sessione duplicata, in cui C attiva contemporaneamente *due* sessioni del protocollo con B, in una delle quali impersona A, nel frattempo supposto off-line od in DoS:

III ATTACCO : Sessione duplicata

1) $C \rightarrow B$: A	C invia l'identità di A al responder B
1') $C \rightarrow B$: C	C invia la sua identità a B
2) $B \rightarrow A$: N_b	B risponde con un nonce ad A, ma questo è intercettato da C
2') $B \rightarrow C$: N'_b	Il nonce inviato a C differisce da quello supposto per A
3) $C \rightarrow B$: $\{\|N_b\|\}_{KcS}$	C cifra con K_{cS} il nonce destinato ad A e lo re-invia a B
3') $C \rightarrow B$: $\{\|N_b\|\}_{KcS}$	C camuffa il nonce destinato a lui con quello per A
4) $B \rightarrow S$: $\{\|A , \{\|N_b\|\}_{KcS}\|\}_{KbS}$	B invia ad S il nonce che crede cifrato da A
4') $B \rightarrow S$: $\{\|C , \{\|N_b\|\}_{KcS}\|\}_{KbS}$	B invia ad S il nonce che crede relativo a C
5) $S \rightarrow B$: $\{\|N_b\|\}_{KcS \circ KaS \circ KbS}$	S "decifra" il nonce per A e lo invia a B cifrato con K_{bS}
5') $S \rightarrow B$: $\{\|N_b\|\}_{KbS}$	S decifra correttamente il nonce di C e lo invia a B cifrato con K_{bS}

Nel passo (5), siccome $\{\|N_b\|\}_{KcS \circ KbS} \neq N_b$, B rifiuta l'autenticazione; tuttavia il messaggio (5') fa ritenere a B che A si sia autenticato con successo. In definitiva, B potrebbe quindi ritenere che A si sia autenticato con successo e che, viceversa, C non sia autentico; consentendo a C l'impersonazione di A. L'attacco dimostra che i messaggi non sono sufficientemente collegati all'identità dei mittenti. Abadi e Needham hanno provveduto ad una correzione del suddetto protocollo, modificando sia il passo (4) che il passo (5), come illustrato nella figura seguente:

Protocollo Woo – Lam (versione di Abadi e Needham)

1) $A \rightarrow B$: A
2) $B \rightarrow A$: N_b
3) $A \rightarrow B$: $\{N_b\}_{KaS}$
4) $B \rightarrow S$: A, B, $\{N_b\}_{KaS}$
5) $S \rightarrow B$: $\{A , N_b\}_{KbS}$

Osservando a posteriori l'uso della crittografia nel protocollo, essi dedussero che il suo scopo, in particolare nel passo (4), era quello di legare due parti di un messaggio. Osservando invece il nonce N_b , notarono che si limitatava a fornire una prova di aggiornamento, e che quindi non era necessaria un'associazione alla firma di A. Al contrario, nel passo (5) rilevarono che era necessario riportare esplicitamente la firma di A per autenticare l'identità dell'initiator. A differenza della versione

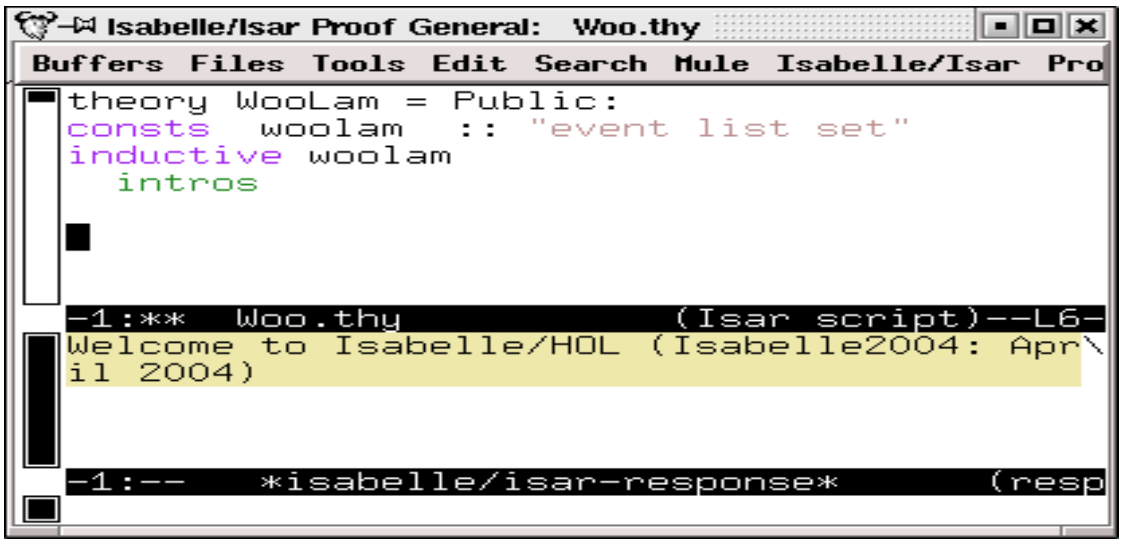
precedente, il suddetto protocollo non è passibile di un attacco a sessione duplicata

4.2.Formalizzazione del protocollo

Ci occuperemo ora della verifica del protocollo Woo-Lam, nella versione di Abadi e Needham, utilizzando il metodo induttivo e con l'ausilio del theorem prover Isabelle. In Isabelle si opera con due tipi differenti di documenti: *theories* e *proof scripts*. Una teoria (*theory*) è una collezione di tipi e di funzioni, così come un modulo in un linguaggio di programmazione o una specifica in un linguaggio di specifica; esse devono risiedere in file con il suffisso .thy. La sintassi generale per un file di teoria T.thy è :

```
T = B1 + . . . + Bn +. . .
        <declarations>
              end
```

dove `B1,..,Bn` rappresentano i nomi delle teorie su cui è basata `T` e `<declarations>` indica l'insieme di costanti e/o di funzioni da utilizzare durante la verifica. `Bi` rappresenta il progenitore diretto della teoria `T`. Tutto quello che risulta definito in esso, è automaticamente visibile. Nel nostro caso, la teoria di base è la `Public.thy`, in cui sono immagazzinati i postulati ed i teoremi utili alla progettazione di protocolli basati sulla crittografia a chiave condivisa.



Interfaccia grafica di Isabelle, intestazione del protocollo

`Bi` è invece rappresentata dalla teoria `Event.thy` degli eventi e dalla teoria `Message.thy` in cui risiedono le regole formali sull'insieme degli agenti e dei messaggi. HOL contiene una teoria conduttrice, `Main.thy`, formata dall'unione di tutte le teorie predefinite di base (aritmetica, insiemistica, ecc ...).

Il protocollo deve essere formalizzato come un insieme di tracce di eventi.

Un evento è di tipo “event list”, per cui la costante che indica il protocollo da analizzare, sarà così formalizzata:

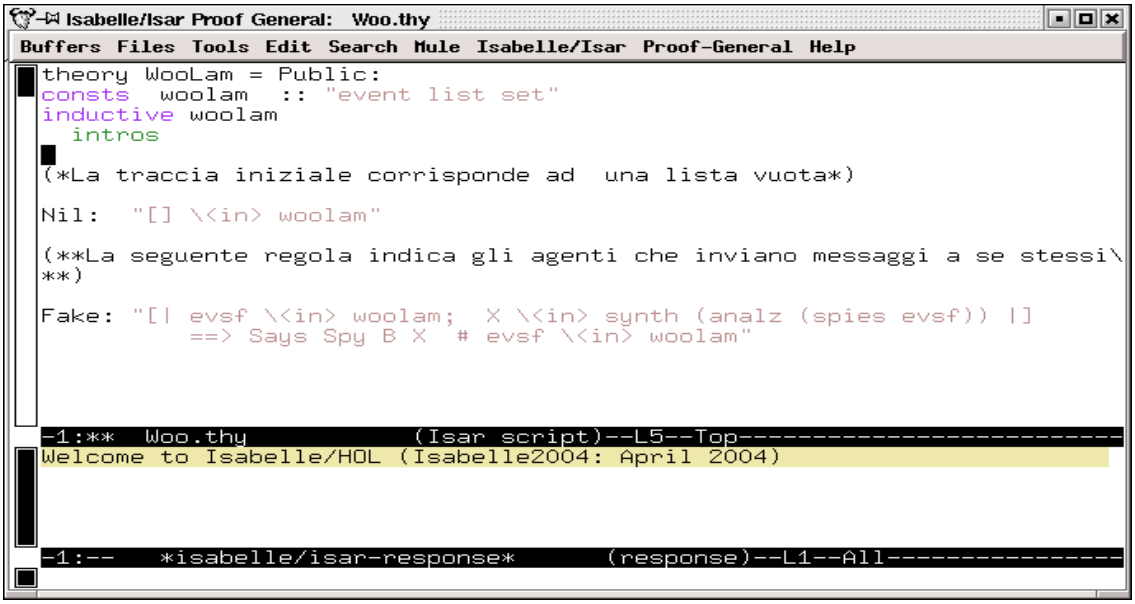
```
consts woolam  :: "event list set".
```

La traccia iniziale è una lista vuota ed è introdotta dalla regola Nil :

```
Nil: "[] \<in> woolam"
```

Innanzitutto è formalizzato il comportamento della spia, attraverso la regola `Fake` coadiuvata da particolari operatori:

```
Fake: "[|evsf \<in> woolam; X \<in> synth (analz(spies evsf))|]
          ==> Says Spy B X  # evsf \<in> woolam"
```



Inizializzazione del protocollo

La spia è parte integrante del protocollo ed è inclusa nell'insieme degli agenti: osserva le comunicazioni ed utilizza tutte le chiavi in suo possesso per decifrare i messaggi; il suo scopo è quindi quello di accumulare chiavi e nonces ed i due operatori `Analz` e `Synth` formalizzano questo comportamento. L'insieme `AnalzH` formalizza che la spia può accedere all'insieme di messaggi H, nel senso che è in grado di leggere i messaggi degli agenti e quello che si può dedurre da questi. L'insieme `SynthH` formalizza che la spia può agire sull'insieme H dei messaggi. Due qualsiasi elementi appartenenti all'insieme SynthH sono combinabili, mentre un elemento dell'insieme può essere cifrato utilizzando una chiave presente in H. Sia `Synth` che `Analz` sono monotoni ed idempotenti.

L'insieme dei messaggi contraffatti che la spia può generare a partire da H è costituito da `Synth (AnalzH)`. Nel nostro caso `H=(spies evsf)`. L'evento `evsf` indica che si sta analizzando un evento legato alla regola `Fake`. L'evento `Says` esprime l'invio ad un agente B di un messaggio `X` da

parte della spia : `Says Spy B X`

Passo 1: L' agente A inizia il protocollo

```
WL1:  "evs1 \<in> woolam ==> Says A B (Agent A) # evs1 \<in> woolam"
```

Questo passo del protocollo, formalizzato nel linguaggio di Isabelle, corrisponde all'evento `Says A B (agent A).` Si noti la traslazione `evs1 ==> Says`, che indica che il primo evento da analizzare è l'evento `Says`.

Passo 2: L'agente B risponde ad A

```
WL2:  "[| evs2 \<in> woolam;
          Says A' B (Agent A) \<in> set evs2 |]
        ==> Says B A (Nonce NB) # evs2 \<in> woolam"
```

`WL2` formalizza il secondo passo del protocollo, in cui il responder B invia un messaggio nonce all'initiator A. Nella prima parte dell'evento `evs2` viene ripetuto il messaggio dell'evento `evs1`, ma con A' al posto di A: questa differenza sta ad indicare che B non conosce A.

Passo 3: A risponde allo scambio di B

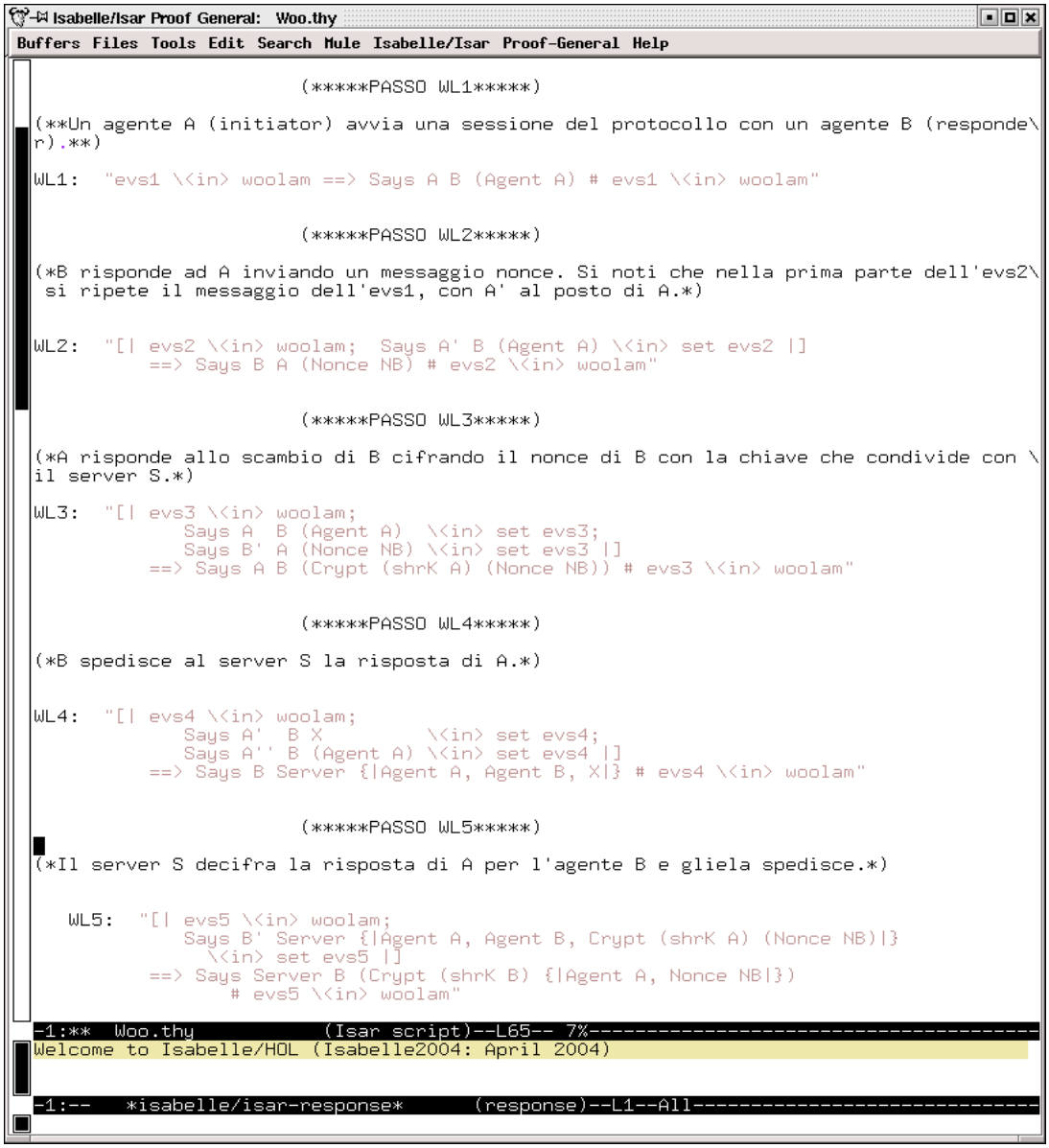
```
WL3:  "[| evs3 \<in> woolam;
          Says A  B (Agent A)  \<in> set evs3;
          Says B' A (Nonce NB) \<in> set evs3 |]
          ==> Says A B (Crypt (shrK A) (Nonce NB))
            # evs3 \<in>  woolam"
```

`WL3` formalizza l'invio del nonce di B da parte di A. A cifra il nonce con la chiave che condivide con S prima di effettuare l'invio, con lo scopo di proteggere il nonce; il nonce cifrato è espresso da: `(Crypt (shrK A) (Nonce NB)).`

Passo 4: B spedisce al server S la risposta di A

```
WL4:  "[| evs4 \<in> woolam;
             Says A'  B X \<in> set evs4;
             Says A'' B (Agent A) \<in> set evs4 |]
          ==> Says B Server {| Agent A, Agent B, X |}#
               evs4 \<in> woolam"
```

Il passo 4 corrisponde all'invio della firma di A e di B, insieme al nonce NB, al server S, da parte del responder B.



I cinque passi del protocollo Woo-Lam

Passo 5: Il Server S decifra la risposta di A

```
WL5: "[| evs5 \<in> woolam;
      Says B' Server {|Agent A, Agent B, Crypt (shrK A)
            (Nonce NB)|} \<in> set evs5 |]
       ==> Says Server B (Crypt (shrK B){|Agent A,Nonce NB|})
        # evs5 \<in> woolam"
```

L'ultimo passo si conclude con l'invio della firma di A e del messaggio decifrato dal server S al responder B; questo passo contiene la modifica di Abadi e Needham alla versione originale del protocollo, che consente di ovviare all'attacco di sessione duplicata:

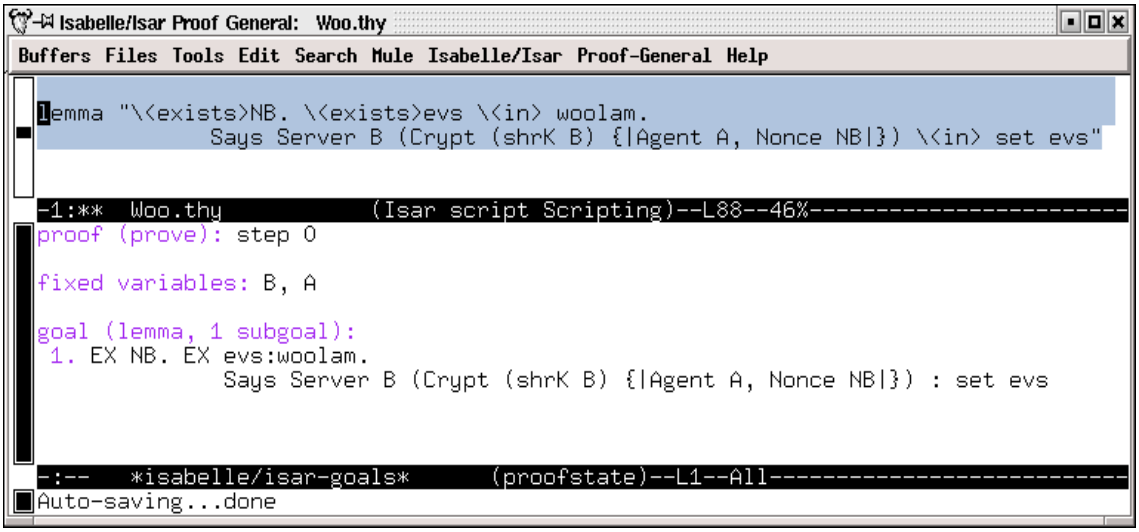
```
Says Server B (Crypt(shrK B){|Agent A,Nonce NB|})
```

4.2.Verifica del protocollo

Proprietà di possibilità

La prima proprietà da dimostrare è quella di *possibilità*, che attesta la conseguenzialità dei passi del protocollo. Poichè gli agenti non sono costretti a comportarsi in un certo modo, non siamo in grado di provare che un evento debba accadere; se però il protocollo non procede ad un certo punto della prova, nel senso che non è possibile giungere all'ultimo passo da dimostrare, allora esso deve essere considerato non corretto. Questa proprietà è formalizzata da un lemma che esprime che l'ultima traccia deve raggiungere la fine della dimostrazione, cioè deve essere trasmessa:

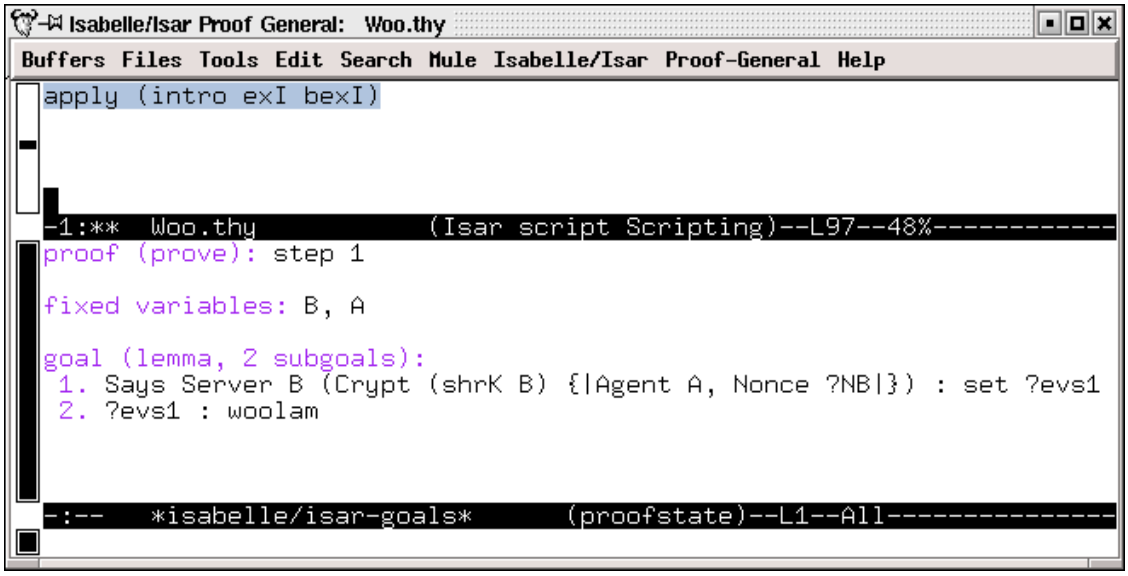
```
lemma "\<exists> NB. \<exists> evs  \<in> woolam.
         Says Server B (Crypt (shrK B) {| Agent A, Nonce NB|})
             \<in>  set evs"
```



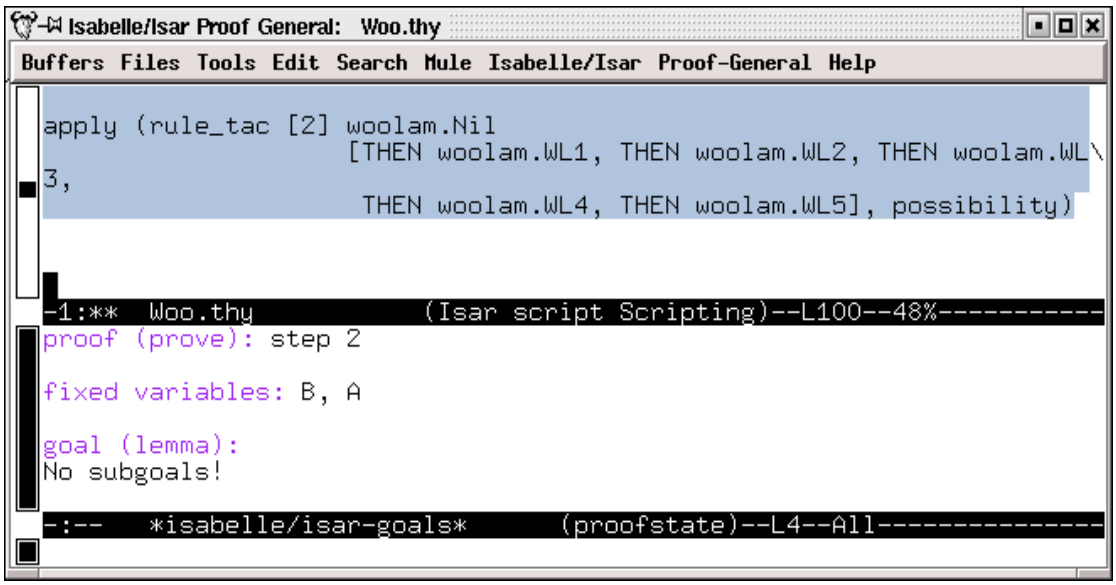
Per risolvere il subgoal applichiamo le seguenti regole induttive :

```
apply (intro exI bexI)
apply (rule_tac [2] woolam.Nil
       [THEN woolam.WL1, THEN woolam.WL2, THEN woolam.WL3,
         THEN woolam.WL4, THEN woolam.WL5], possibility)
done
```

Il comando `apply` è l'istruzione generica per applicare una tattica; spesso è sostituito dal comando `by`. Il termine `intro` sta ad indicare che si sta utilizzando una regola introduttiva, cioè una regola data all'inizio della dimostrazione e che va applicata ripetutamente. Il termine `exI` indica invece che si sta provando l'esistenza del nonce NB.

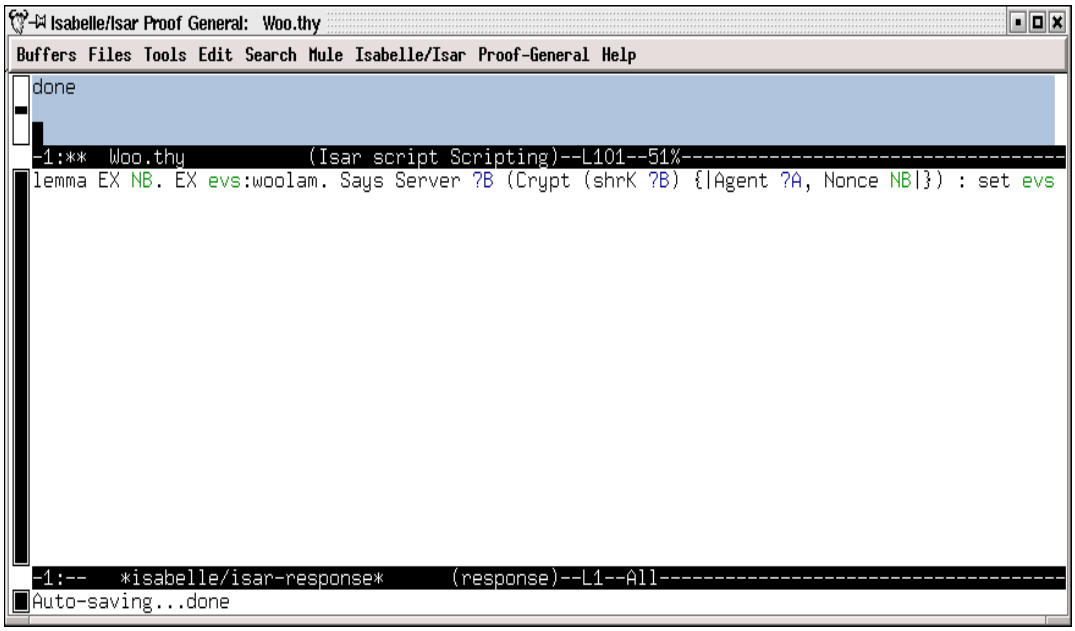


Il comando THEN che compare nella seconda regola applicativa è un metodo risolutivo speciale, in grado di richiamare alcuni passi o alcune funzioni precedenti. Nel nostro caso sono richiamati i 5 passi del protocollo, in quanto è necessario per il corretto svolgimento della prova.



Si noti che adesso non compaiono subgoal; ciò vuol dire che il lemma è stato dimostrato.

Il comando finale done chiude la dimostrazione del lemma; esso serve ad Isabelle per associare il lemma provato al suo nome relativo (oltre a stampare il risultato):

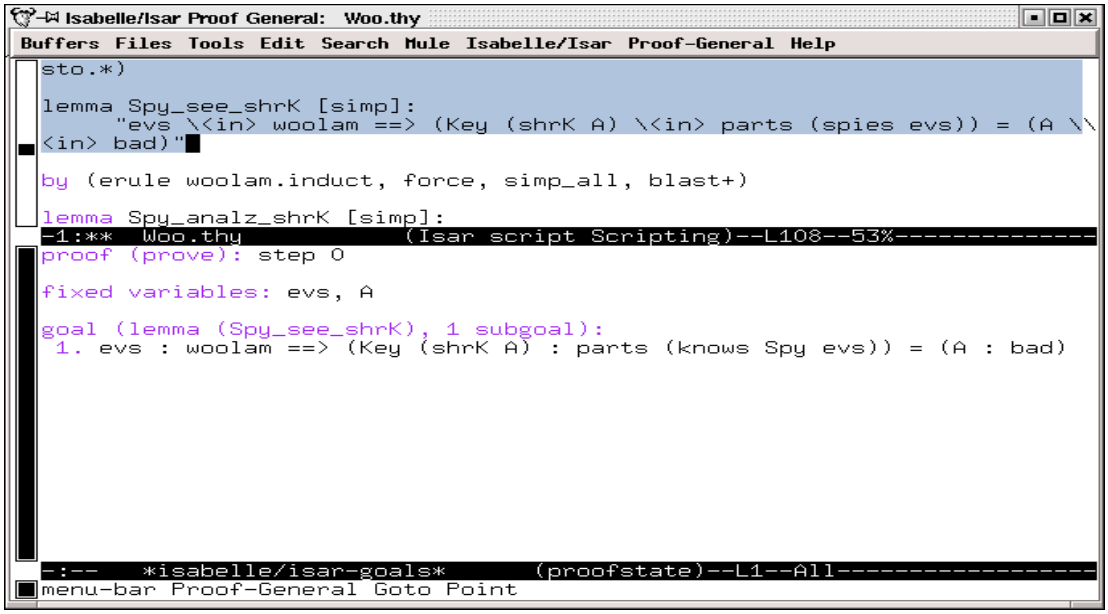


Proprietà di riservatezza

Comportamento della spia: ***lemma di spedizione***

Questo lemma si applica al passo in cui l'agente spedisce il messaggio nonce cifrato; con esso si intende specificare che la spia non può mai ottenere la chiave segreta di un agente non compromesso, e quindi non può apprendere il nonce:

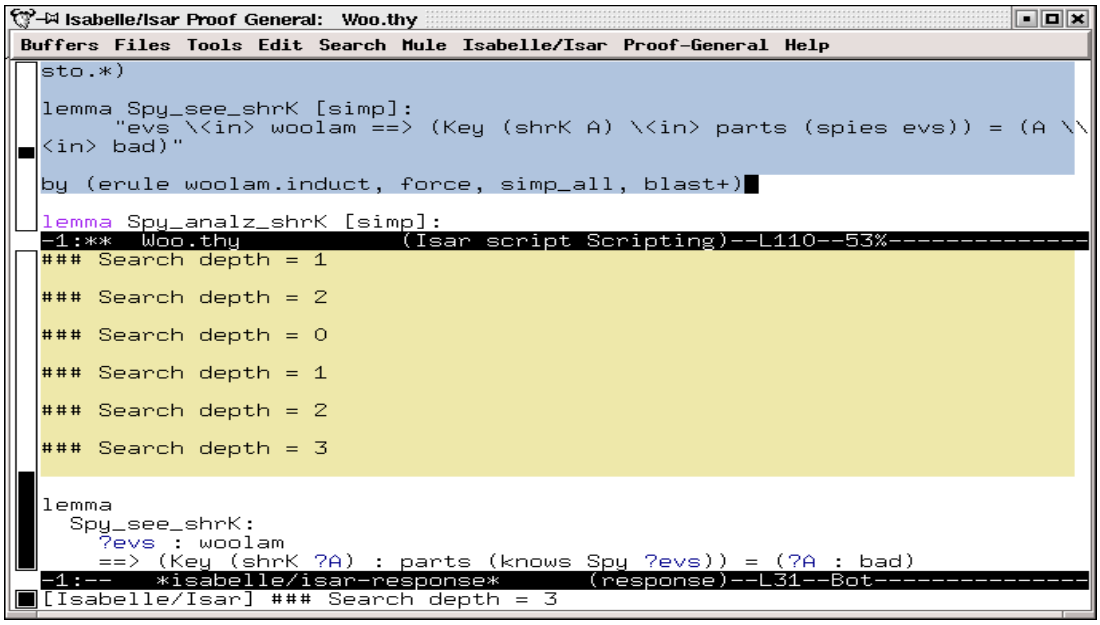
```
lemma Spy_see_shrK [simp]:  "evs \<in> woolam
    ==> (Key (shrK A) \<in> parts (spies evs)) = (A \<in> bad)"
```



Il lemma utilizza l'operatore `parts`, in quanto la presenza stessa della chiave di un agente A in un messaggio, protetto o meno da crittografia, basta a stabilire che A è compromesso. La dimostrazione, come quasi tutte quelle del protocollo, è svolta per induzione sulle tracce:

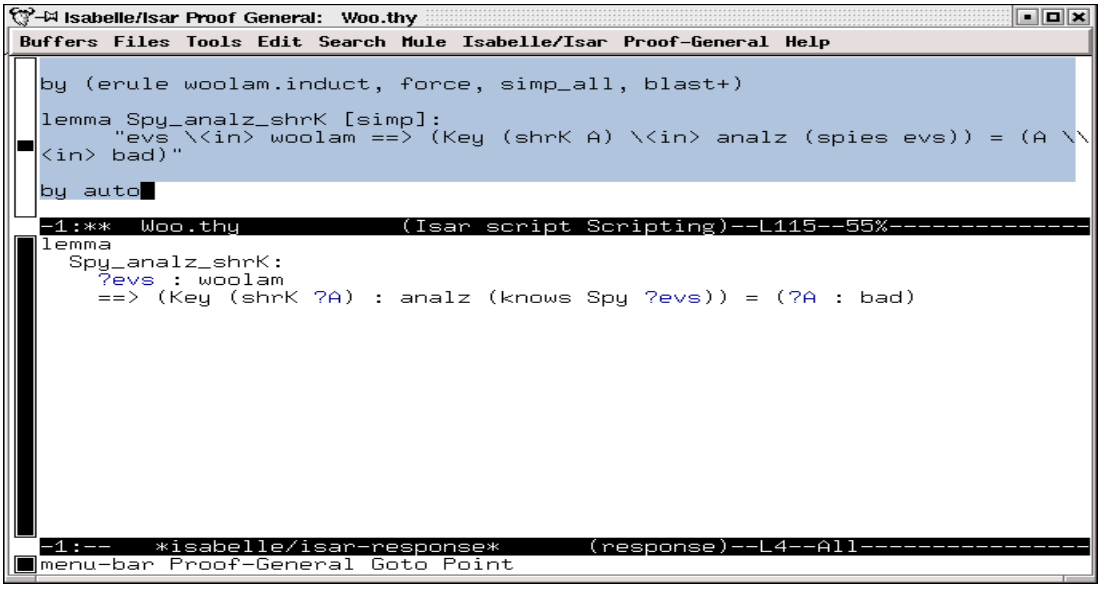
```
by (erule woolam.induct, force, simp_all, blast+).
```

Questa particolare ricerca è avviata tramite la regola `blast`, che accelera la dimostrazione caricando automaticamente tutte le regole necessarie:

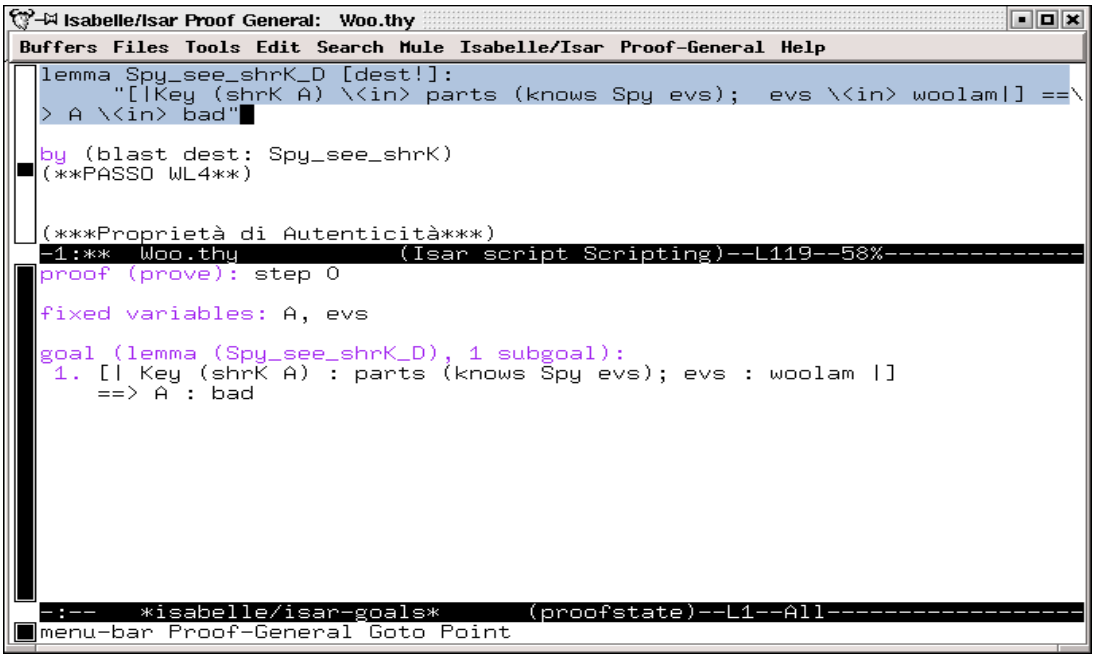


Il primo subgoal del lemma di spedizione è formalizzato con l'operatore **Analz** ed afferma che se la spia fosse in grado di accedere al messaggio nonce, allora A sarebbe un agente compromesso. È una particolarizzazione del lemma precedente, in quanto `Analz H ⊆ Parts H`:

```
lemma Spy_analz_shrK [simp]:
        "evs \<in> woolam ==> (Key (shrK A) \<in>
          analz (spies evs)) = (A \<in> bad)"
```



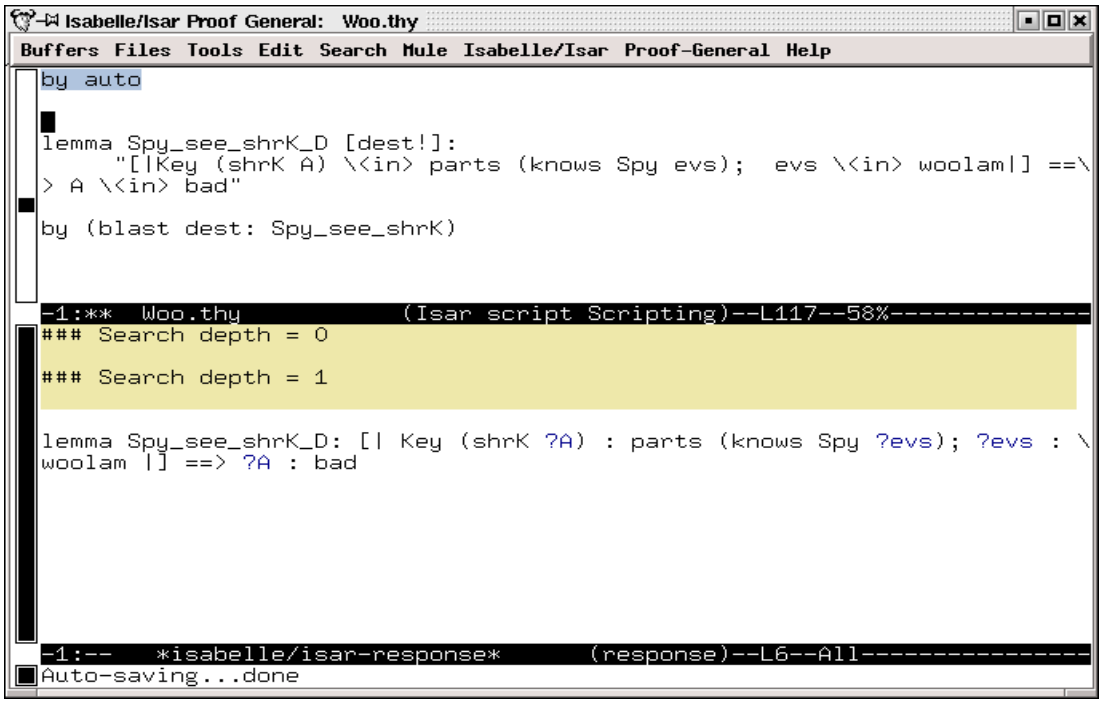
Anche in questo caso la dimostrazione è automatica, grazie al comando `by auto` che ottiene:



L'ultimo subgoal serve a richiamare il risultato del precedente:

```
lemma Spy_see_shrK_D [dest!]:
      "[|Key (shrK A) \<in> parts (knows Spy evs);
          evs \<in> woolam|] ==> A \<in> bad"
```

La dimostrazione è effettuata con `by (blast dest: Spy_see_shrK)`:

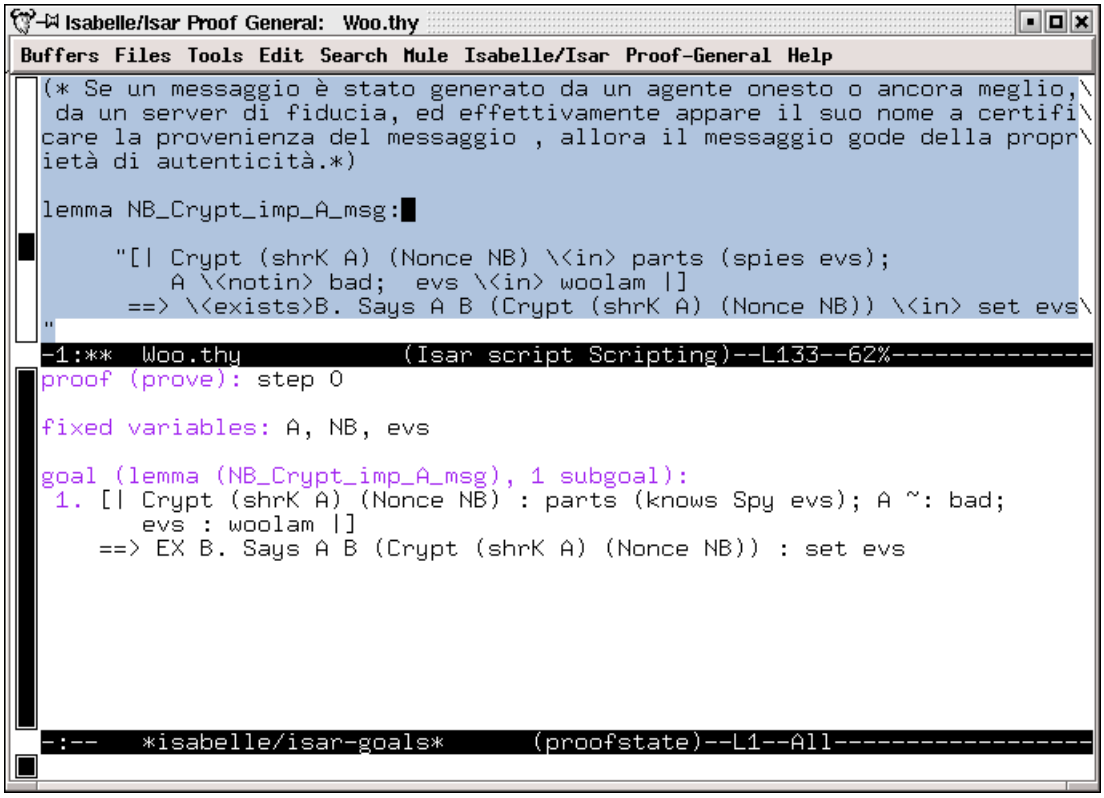


Proprietà di Autenticità

Se un messaggio fosse generato da un agente onesto o da un server di fiducia, ed effettivamente apparisse il nome di questo a certificare la provenienza del messaggio, il messaggio godrebbe della proprietà di autenticità. Molti ricercatori considerano il mittente di un messaggio come parte essenziale del messaggio; di conseguenza, verificare l' integrità del messaggio implica la conferma della sua autenticità. Viceversa, individuare il creatore di un messaggio ricevuto, conferma che il messaggio non è stato cambiato. Le due proprietà possono essere considerate equivalenti. Vogliamo dimostrare che se appare il messaggio nonce cifrato, allora sicuramente è stato generato da A:

```
lemma NB_Crypt_imp_A_msg:
  "[| Crypt (shrK A) (Nonce NB) \<in> parts (spies evs);
        A \<notin> bad;  evs \<in> woolam |]
  ==> \<exists>B. Says A B (Crypt (shrK A) (Nonce NB))
        \<in> set evs"
```

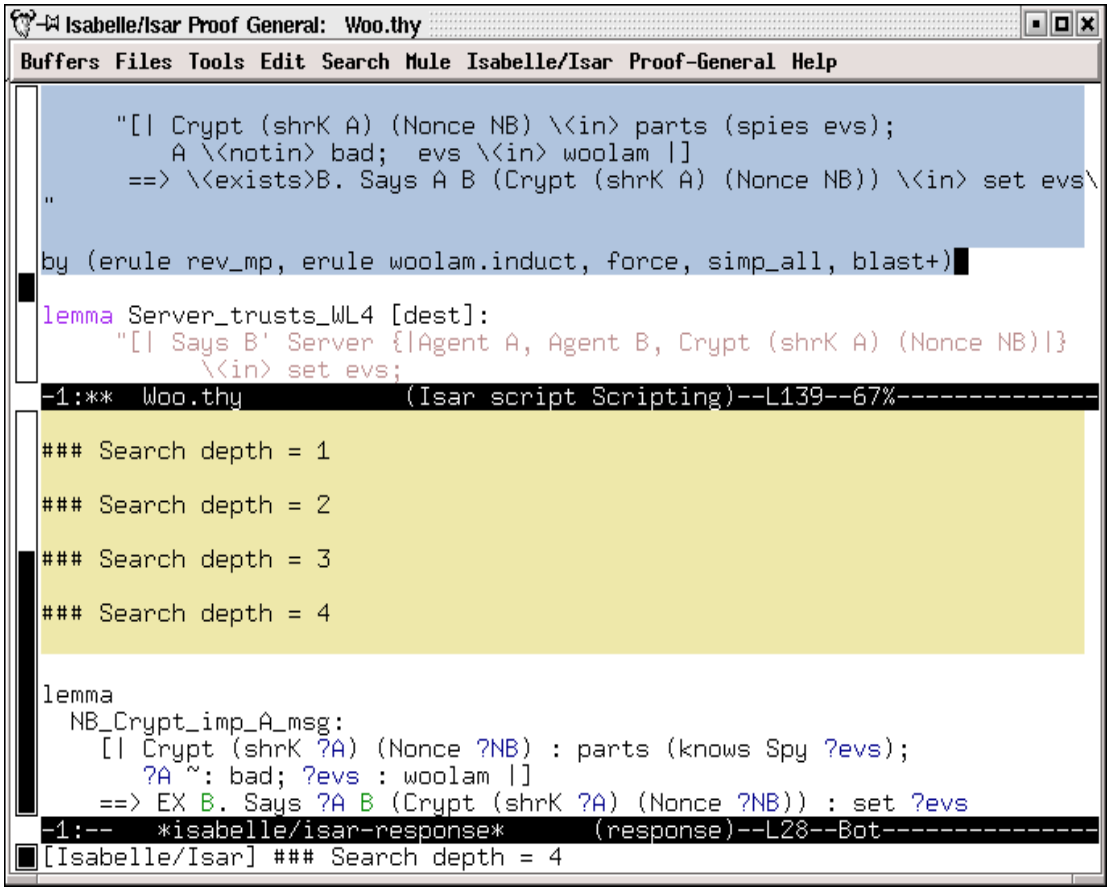
Il nome stesso del lemma ne spiega il senso: Il nonce NB cifrato implica che il messaggio è di A.



Applichiamo il comando dimostrativo by:

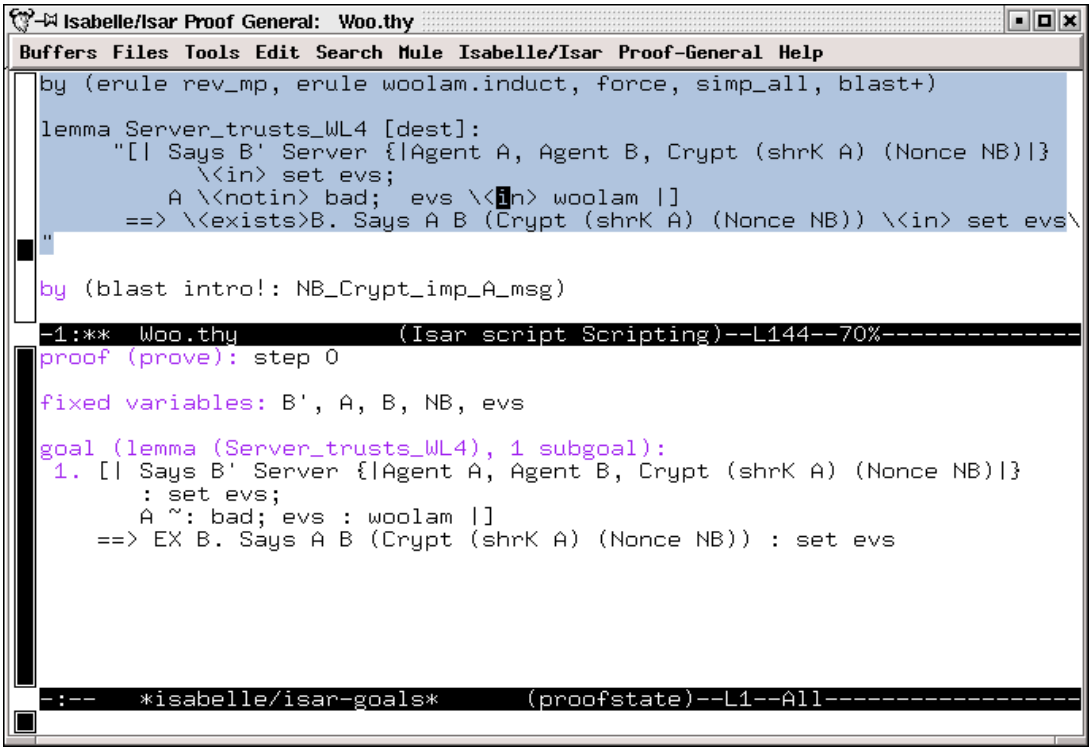
```
by (erule rev_mp, erule woolam.induct, force, simp_all, blast+)
```

Il termine `rev_mp` sta ad indicare l'applicazione di una regola di illazione, la regola *modus ponens*, che determina due subgoal, uno in cui si dimostra una proprietà antecedente ed un altro in cui è possibile ammetterne la diretta conseguenza.



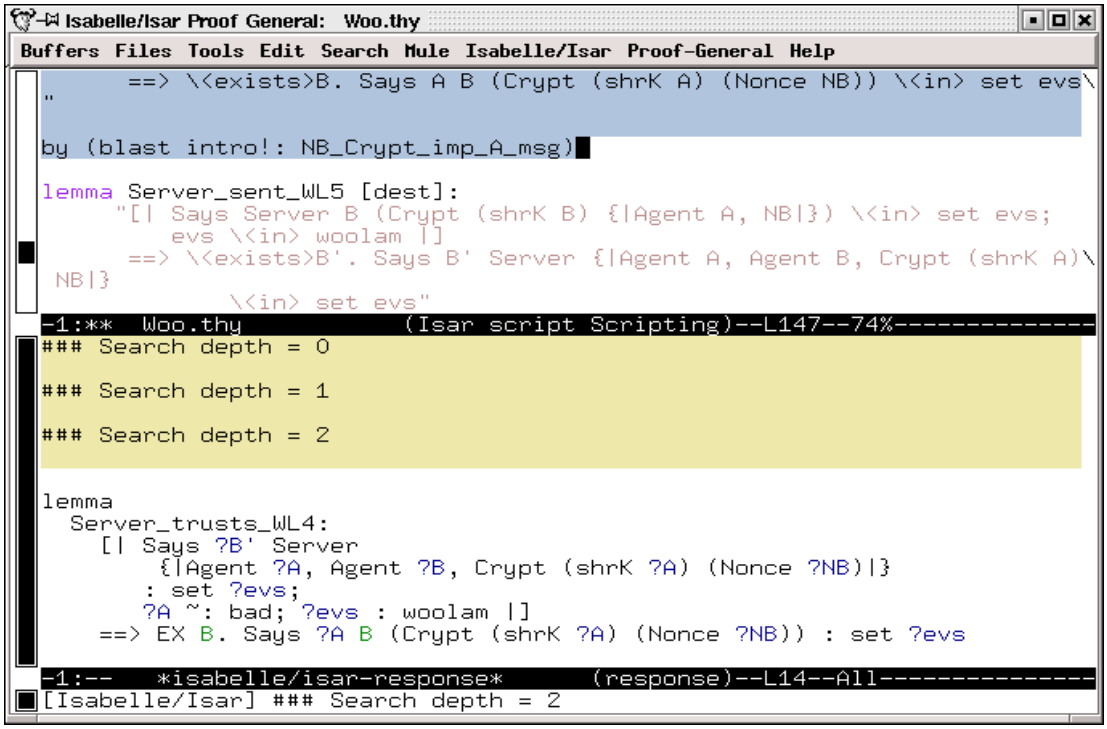
Tra i lemmi di autenticità troviamo quelli di *garanzia* in quanto asseriscono una forma di autenticità. In particolare qui si vuole esprimere una garanzia per il server: " Se il server riceve un messaggio contenente un certificato dell'initiator A, allora sicuramente è stato prodotto da A. Non siamo però certi che il responder B lo abbia appreso; questa eventualità potrebbe spingere la spia a redirigere il messaggio al server.

```
lemma Server_trusts_WL4 [dest]:
"[| Says B' Server {|Agent A, Agent B,
     Crypt (shrK A) (Nonce NB)|} \<in> set evs;
              A \<notin> bad;  evs \<in> woolam |]
==> \<exists>B. Says AB (Crypt(shrKA)(Nonce NB)) \<in> set evs"
```



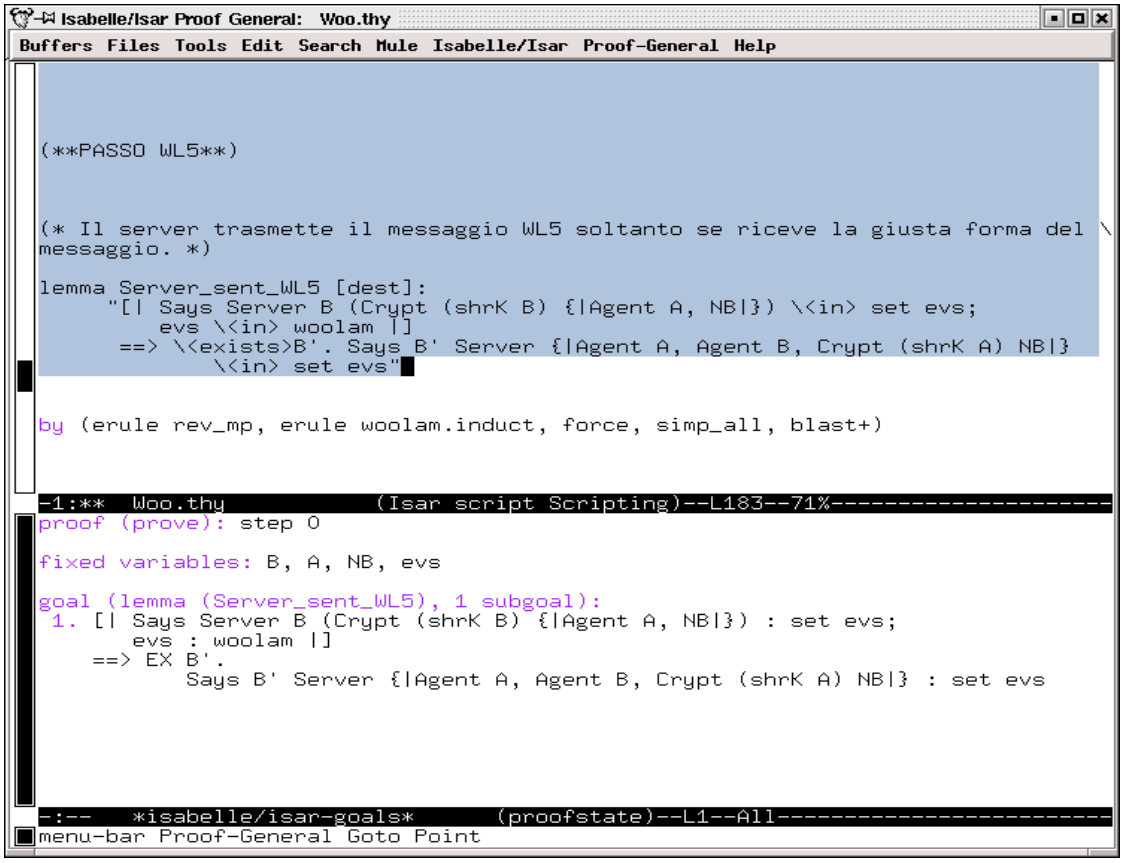
La dimostrazione si effettua con ancora una volta col comando by

`by (blast intro!: NB_Crypt_imp_A_msg)`



Il server trasmette il messaggio WL5 solo nel caso in cui la verifica del protocollo sia proceduta correttamente fino a questo passo; si è dunque ricondotti al lemma di possibilità**.**

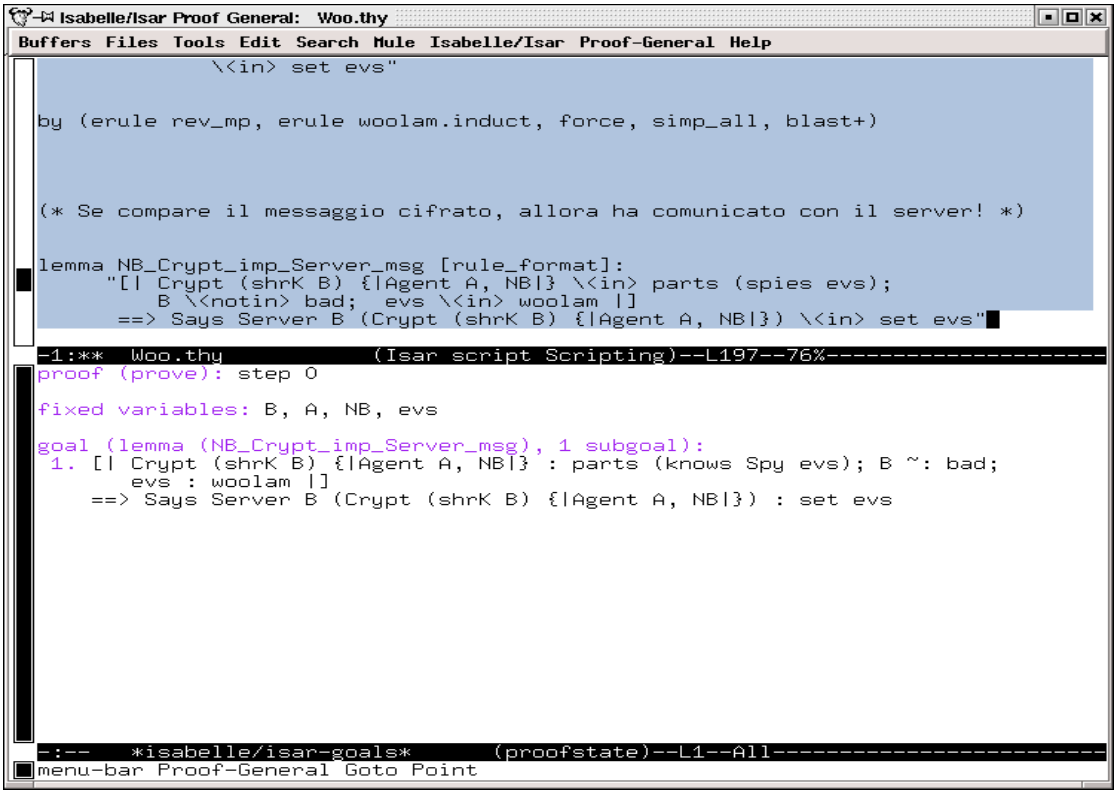
```
lemma Server_sent_WL5 [dest]:
 "[| Says Server B (Crypt (shrk B) {|Agent A, NB|} \<in> set evs;
    evs \<in> woolam |]
   ==> \<exists>B'. Says B' Server  {|Agent A,Agent B, Crypt
   (shrk A) NB|} \<in> set evs"
```



Si introduce nuovamente la regola di illazione *modus ponens*, che da luogo a due subgoal: uno in cui si dimostra la *proprietà di possibilità* antecedente, ed un altro in cui è possibile ammetterne la diretta conseguenza, e cioè che il messaggio WL5 è trasmesso unicamente se il protocollo è corretto.

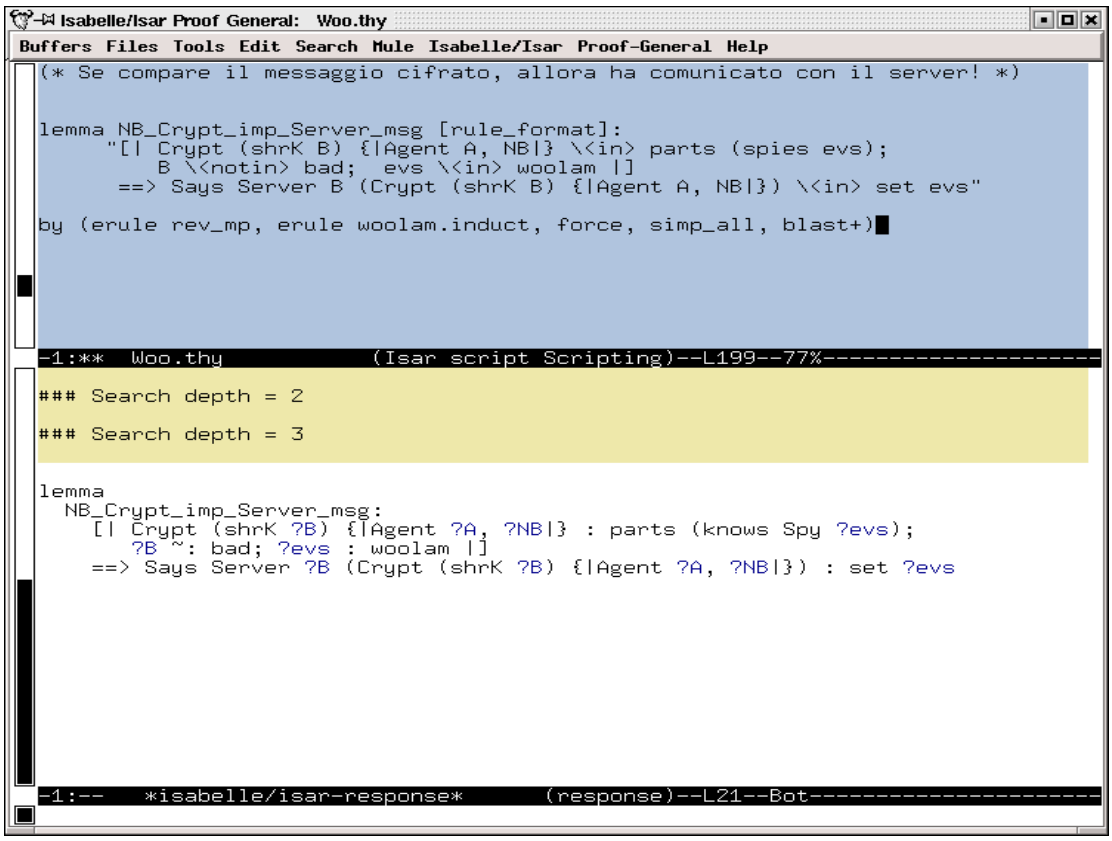
Se compare un messaggio cifrato, allora B ha comunicato con il server. Anche questo lemma attesta l'autenticità del messaggio, in quanto il messaggio cifrato è sinonimo di messaggio invariato al momento della ricezione. Questo implica anche che il creatore del messaggio non è cambiato.

```
lemma NB_Crypt_imp_Server_msg [rule_format]:
 "[| Crypt (shrK B){|Agent A , NB|} \<in> parts (spies evs);
      B \<notin> bad ; evs \<in> woolam |]
    ==> Says Server B (Crypt (shrK B){|Agent A, NB|})
     \<in> set evs"
```



Ancora una volta si fa uso di *modus ponens*:

```
by(erule rev_mp, erule woolam.induct, force, simp_all, blast+)
```

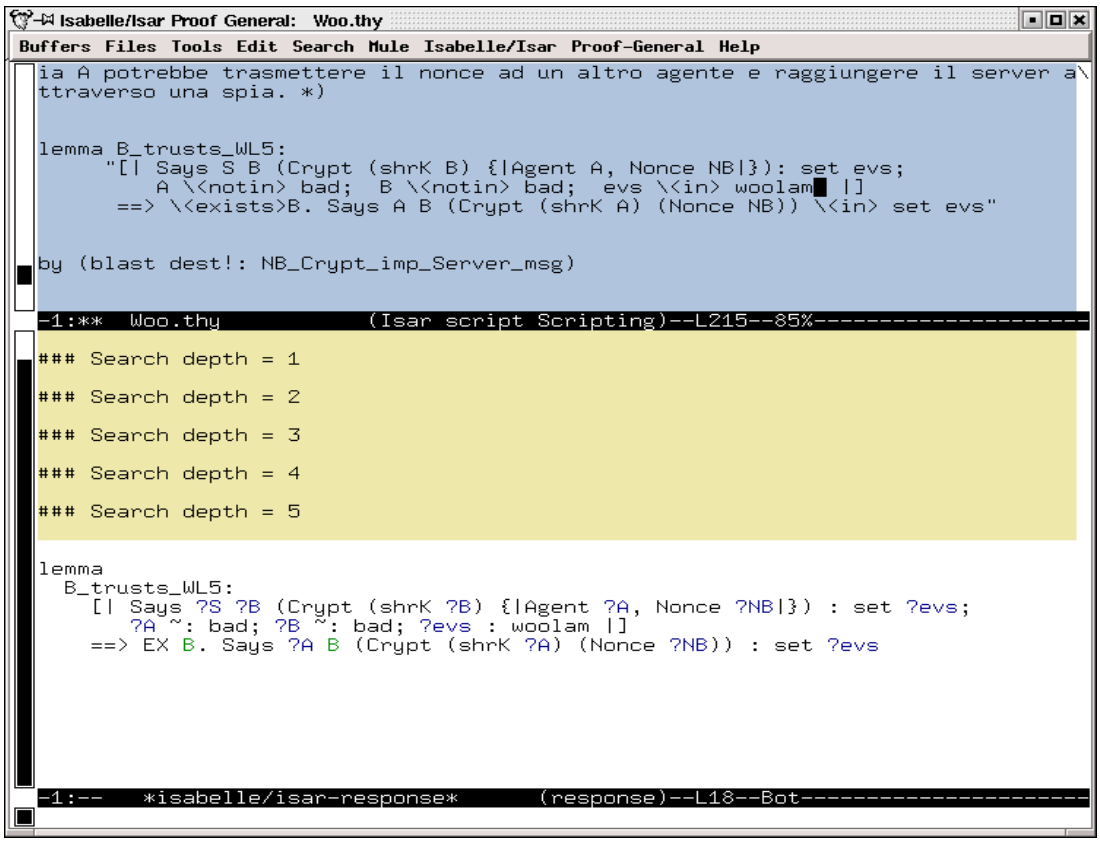


Se il responder B ottiene il certificato del server, allora A ha cifrato il nonce usando la propria chiave. Questo evento non può essere più vecchio del nonce stesso. Tuttavia A potrebbe trasmettere il nonce ad un altro agente che raggiungerebbe il server attraverso una spia.

```
lemma B_trusts_WL5
"[| Says S B (Crypt (shrK B){|Agent A , NB|}) : set evs;
      A\<notin> bad ; B\<notin> bad ; evs \<in> woolam |]
 ==> \<exists>B. Says A B {|Agent A, Agent B,
         (Crypt (shrk A) (Nonce NB)) \<in> set evs"

by (blast dest!: NB_Crypt_imp_Server_msg)
```

La parola `dest` sta per *rule destruction* con cui si vuole intendere una regola distruttiva, ossia una regola capace di invalidare una premessa fatta. La premessa in questione è rappresentata dal lemma precedente: attribuendo la regola `dest` al lemma `NB_Crypt_imp_Server_msg` si impone di usare la regola `blast` come regola distruttiva.



Ciò conclude la verifica induttiva del protocollo Woo_Lam modificato, validandolo.

Si può osservare che il protocollo garantisce solamente una forma di autenticazione unidirezionale. Infatti nel caso di autenticazione di B da parte di A, il protocollo è passibile di un attacco di

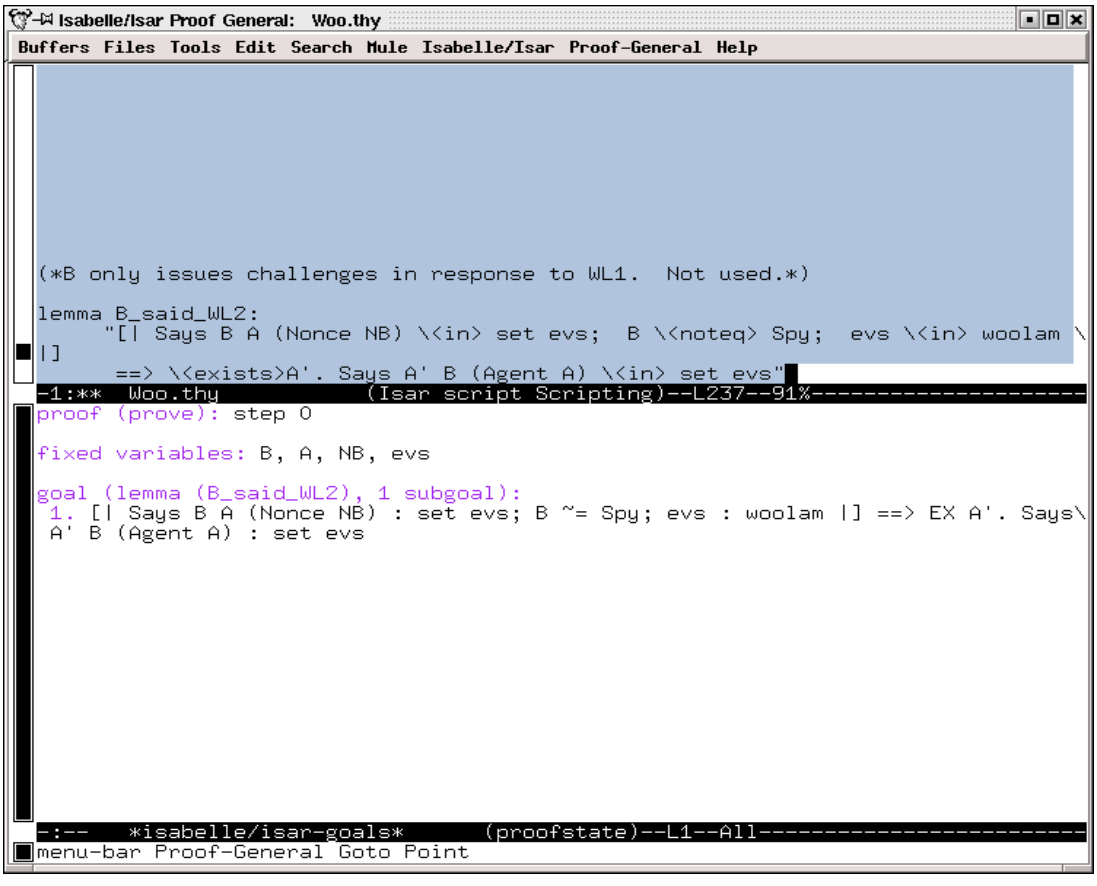
impersonazione del responder. A riprova di ciò vi è uno script realizzato dallo stesso Paulson che riporta due lemmi che dovrebbero attestare la provenienza del nonce NB, ma la prova si blocca alla dimostrazione del secondo ed il protocollo è così invalidato. In effetti, i lemmi implicano una mutua autenticazione, ma il protocollo è stato formalizzato solamente per una forma di autenticazione unidirezionale. Eliminandoli non sorgono problemi ed il protocollo può considerarsi corretto.

ATTACCO : IMPERSONAZIONE DI B

1) A → B : A	A invia la sua identità a B ma il responder è in DoS
2) B ' → A : $N_{b'}$	B' lo impersona spedendo un suo nonce ad A
3) A → B : $\{N_{b'}\}_{KaS}$	A spedisce il nonce cifrato a B, ma B' lo intercetta.
4) B' → S : A, B ', $\{N_{b'}\}_{KaS}$	B ' invia al server l'identità di A, la sua ed il nonce.
5) S → B : $\{A , N_{b'}\}_{Kb'S}$	Il server riconosce la corrispondenza tra identità e nonce e fa la decifratura. B' intercetta anche il messaggio inviato da S a B.

Siccome B' intercetta anche il messaggio finale indirizzato da S a B, la sessione del protocollo si conclude senza che sia possibile rilevare l'errore; A non è quindi garantito circa l'autenticazione del responder B.

Riportiamo l'esecuzione dei due lemmi e lo stallo provocato dal secondo. Lo schema seguente rappresenta il primo lemma:



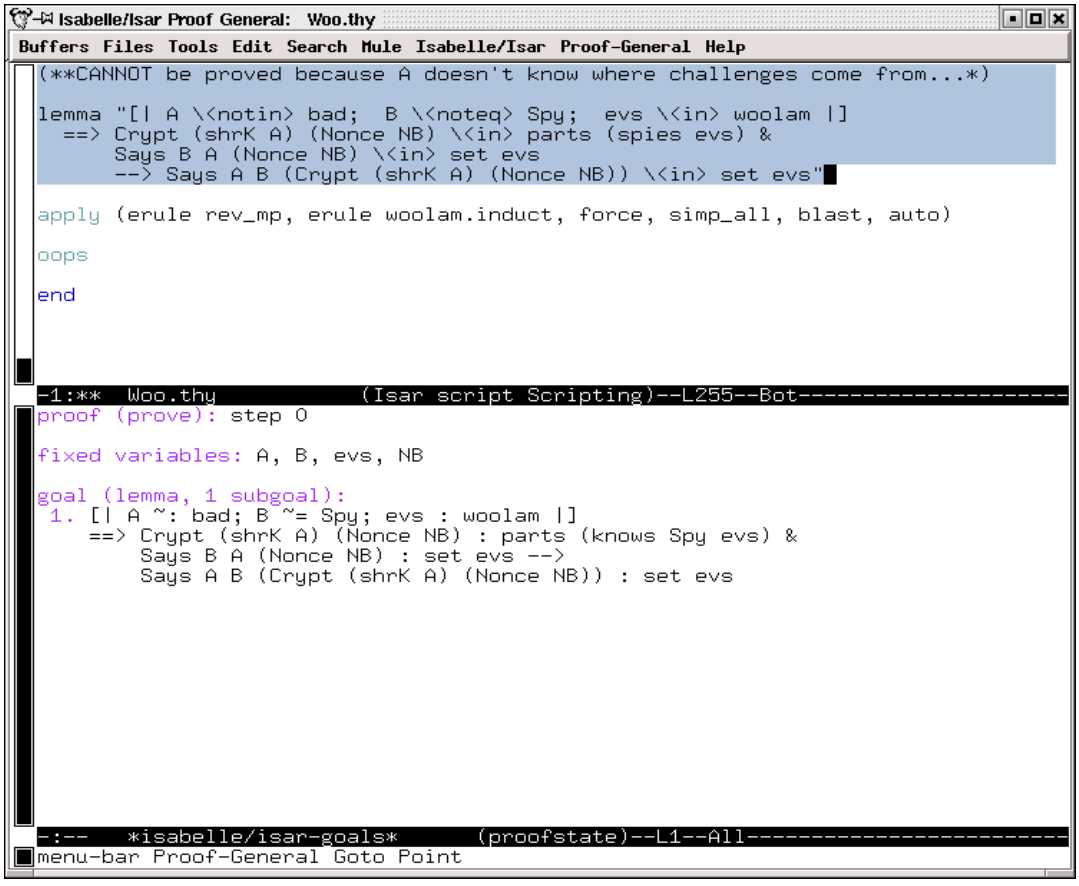
```
lemma B_said_WL2
 "[| Says B A (Nonce NB) \<in> set evs;
      B\<noteq> Spy ; evs \<in> woolam |]
 --> \<exists>A'. Says A' B (Agent A) \<in> set evs"
```

Esso riesce ad essere dimostrato grazie al comando :

```
apply (erule rev_mp,erule woolam.induct,force, simp_all,
       blast,auto)
```

Il lemma che non si riesce a dimostrare è il seguente :

```
lemma "[| A\<notin> bad ; B\<noteq> Spy ; evs \<in> woolam |]
  ==> Crypt (shrk A) (Nonce NB)) \<in> parts (spies evs) &
     Says B A (Nonce NB) \<in> set evs
 --> Says A B (Crypt (shrK A) (Nonce NB)) \<in> set evs”
```

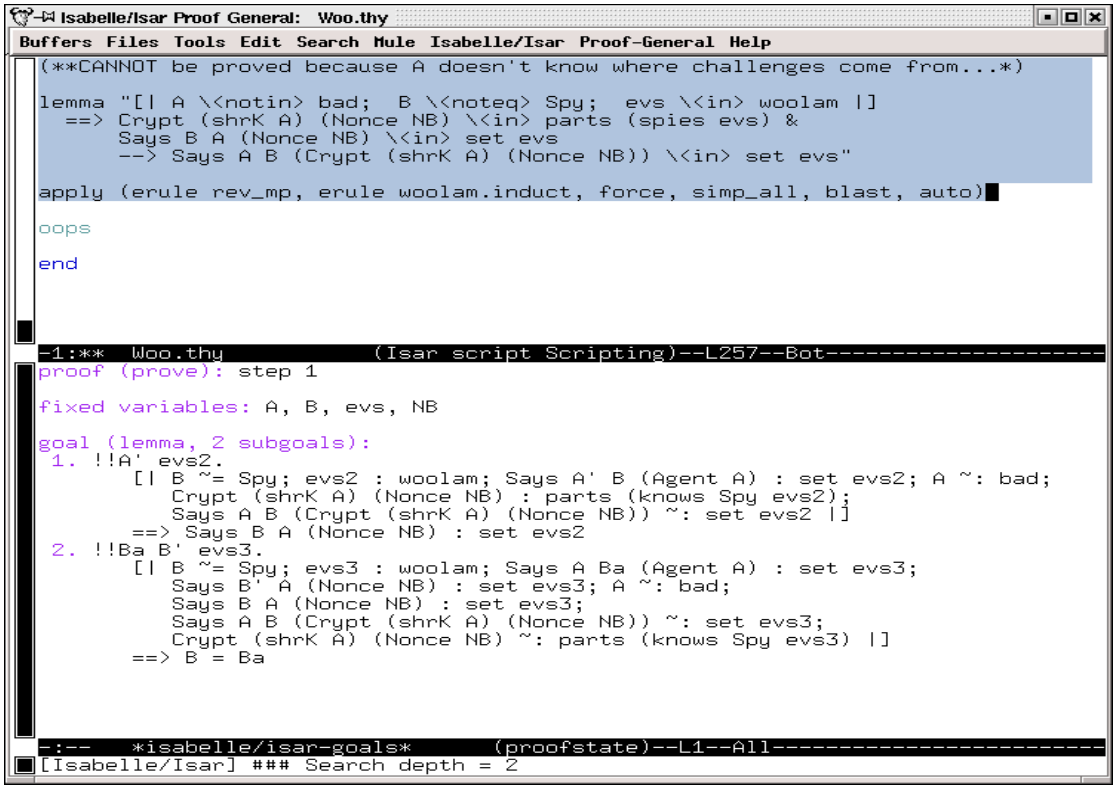


Il comando `apply`

```
apply(erule rev_mp,erule woolam.induct,force,simp_all,blast,auto)
```

genera invero due subgoal che provocano lo stallo di Isabelle, richiedendo il ricorso al comando

oops.


Isabelle/Isar Proof General: Woo.thy
Buffers Files Tools Edit Search Mule Isabelle/Isar Proof-General Help
(**CANNOT be proved because A doesn't know where challenges come from...*)
lemma "[| A \<notin> bad; B \<noteq> Spy; evs \<in> woolam |]
==> Crypt (shrK A) (Nonce NB) \<in> parts (spies evs) &
Says B A (Nonce NB) \<in> set evs
--> Says A B (Crypt (shrK A) (Nonce NB)) \<in> set evs"
apply (erule rev_mp, erule woolam.induct, force, simp_all, blast, auto)
oops
end
-1:** Woo.thy (Isar script Scripting)--L257--Bot
proof (prove): step 1
fixed variables: A, B, evs, NB
goal (lemma, 2 subgoals):
1. !!A' evs2.
[| B ~= Spy; evs2 : woolam; Says A' B (Agent A) : set evs2; A ~: bad;
Crypt (shrK A) (Nonce NB) : parts (knows Spy evs2);
Says A B (Crypt (shrK A) (Nonce NB)) ~: set evs2 |]
==> Says B A (Nonce NB) : set evs2
2. !!Ba B' evs3.
[| B ~= Spy; evs3 : woolam; Says A Ba (Agent A) : set evs3;
Says B' A (Nonce NB) : set evs3; A ~: bad;
Says B A (Nonce NB) : set evs3;
Says A B (Crypt (shrK A) (Nonce NB)) ~: set evs3;
Crypt (shrK A) (Nonce NB) ~: parts (knows Spy evs3) |]
==> B = Ba
-:-- *isabelle/isar-goals* (proofstate)--L1--All
[Isabelle/Isar] ### Search depth = 2

Appendice 1 – Installazione di *Isabelle*

1.Interfacce grafiche per Isabelle

Isamode

Isamode è stato, in ordine cronologico, il primo un front-end per il theorem prover Isabelle. E' integrato strettamente con l'editor XEmacs e fornisce molte altre caratteristiche utili, tra cui la possibilità di esportare in formati opportuni (TeX, , etc.) i files di teoria, ed un' interfaccia grafica a finestre multiple con menù a comparsa. Le finestre servono a visualizzare esclusivamente lo stato delle dimostrazioni, le shell di Isabelle e la teoria corrente. Isamode ed è stato rilasciato nel 1994 e dal 1998 in poi non è stato ulteriormente esteso, anche se sono stati resi disponibili degli aggiornamenti per farlo funzionare con le versioni più recenti di Isabelle e di XEmacs.

Proof General

Proof General è un'interfaccia generica per i proofs assistents, sviluppata al LFCS nell'università di Edinburgo e distribuita dalla GNU Corporation gratuitamente (http://proofgeneral.inf.ed.ac.uk/).

Proof General può funzionare sia con XEmacs che con GNU-Emacs, ed è in via di sviluppo una nuova versione basata su Eclipse: http://proofgeneral.inf.ed.ac.uk/kit/wiki. Proof General su XEmacs offre un adeguato supporto per la gestione dei caratteri speciali attraverso gli X-Symbols,

semplificando grandemente la stesura di script per Isabelle. Molte teorie di Isabelle hanno set di X-Symbols predefiniti, che è facile aggiungere al proprio ambiente di lavoro.

2. Installazione di Isabelle

All'epoca della stesura del presente documento, sono gratuitamente disponibili i package binari di Isabelle2004 per tutte le più comuni piattaforme UNIX (Linux/x86, Solaris/Sparc e Darwin/PPC (MacOS X)). Norbert Völker fornisce una pagina con suggerimenti su come ottenere il core system funzionante su Windows, che però non è una piattaforma ufficialmente sostenuta per Isabelle.

I requisiti minimi di installazione prevedono l'esistenza della `bash` shell, dell'interprete `perl` e di un package per l'esecuzione di ML (per esempio, `Poly/ML`).

L'interfaccia utente più sofisticata è al momento ottenibile grazie all'installazione di `Proof General`, la cui distribuzione (gratuitamente disponibile previa registrazione) include il pacchetto `X-Symbol`, che richiede una versione di `XEmacs` o `Gnu-Emacs` a partire dalla 2.1. Per maggiori informazioni sui requisiti e sulle fasi di installazione si faccia riferimento al file INSTALL, presente nella distribuzione di Isabelle. Un'installazione tipica di Isabelle/HOL su piattaforme Linux/x86 consta nelle fasi seguenti:

Fase1. Download e de-archiviazione dei package necessari

I package necessari al funzionamento sono solitamente disponibili per il download nella forma archivi TAR compressi; l'installazione "core" di Isabelle per il suo impiego con il metodo induttivo prevede i download seguenti: `Isabelle.tar.gz`, `polyml_base.tar.gz`, `polyml_x86-linux.tar.gz` e `HOL_x86-linux.tar.gz`. Le linee di comando seguenti consentono di decomprimere e de-archiviare i download ottenuti nella directory `/usr/local`, che è quella standard prevista:

```
root>tar -C /usr/local -xzf Isabelle.tar.gz
root>tar -C /usr/local -xzf polyml_base.tar.gz
root>tar -C /usr/local -xzf polyml_x86-linux.tar.gz
root>tar -C /usr/local -xzf HOL_x86-linux.tar.gz
```

Fase2. Installazione di Isabelle

L'installazione di Isabelle avviene grazie al comando `isatool install;` l'opzione `-p` del comando permette di definire quale debba essere la directory degli eseguibili. Solo qualora bash e perl non si dovessero trovare nei path standard (risp. /bin/bash e /usr/bin/perl) è necessario premettere

all'esecuzione del comando isatool install quella dello script configure:

```
root>cd /usr/local/Isabelle
[root>./configure]
root>./bin/isatool install -p /usr/local/bin
```

Fase3. Installazione dell'interfaccia grafica

Osserviamo che in genere le distribuzioni Linux dispongono di `Gnu-Emacs` e non di `XEmacs`, ciò richiede una modifica alla distribuzione precompilata di Proof General, rilasciata per `XEmacs`, a meno che non si intenda installare quest'ultimo[#]. Se sul sistema si dispone solo di (`Gnu-`)`Emacs`, occorre cancellare i file con estensione `elc` nella directory di `Proof General`, e ricompilare con (`Gnu-`)`Emacs`. Istruzioni dettagliate su come svolgere tale compilazione sono contenute nel file INSTALL, accluso con la distribuzione di `ProofGeneral`.

A questo punto l'installazione è di fatto conclusa, ed Isabelle può essere mandata in esecuzione lanciando il comando `/usr/local/bin/Isabelle`, che avrà come effetto anche quello di mandare in esecuzione l'interfaccia `Proof General`. Si noti che esiste un'opzione separata nel menu `Opzioni` di `Proof General` per l'uso degli X-Symbol[$].

#Per sapere se di dispone di `XEmacs` , si può provare a digitare il comando `rpm -q xemacs`. Nel caso in cui non si disponga del comando `rpm` (standard su piattaforme Red Hat), una possibile alternativa consiste nell'effettuare una ricerca diretta del package attraverso il comando `find / -name xemacs.`

$Se `Emacs` dovesse andare in stallo quando ha inizio il processo dimostrativo, si leggano le informazioni del file FAQ nella directory di `Proof-General`.

Appendice 2 – Script per la verifica del protocollo Woo-Lam

```
**************************************************************

(* Università degli Studi di Napoli Federico II e ICAR-CNR, Napoli *)
(* Authors: Marianna Caiazza, Giuliano Laccetti, Giovanni Schmid *)
(*  TITOLO :  PROTOCOLLO DI AUTENTICAZIONE Woo-Lam NELLA VERSIONE CORRETTA
DA ABADI E NEEDHAM [AB-NE 94] *)

theory WooLam = Public:
consts  woolam  :: "event list set"


(*La traccia iniziale corrisponde ad  una lista vuota*)

Nil:  "[] \<in> woolam"

(**La seguente regola indica gli agenti che inviano messaggi a se stessi**)

Fake: "[| evsf \<in> woolam;  X \<in> synth (analz (spies evsf)) |]
       ==> Says Spy B X  # evsf \<in> woolam"

(*************************************************************************************)


                (*****PASSO WL1*****)


(**Un agente A (initiator) avvia una sessione del protocollo con un agente B (responder).**)

WL1:  "evs1 \<in> woolam ==> Says A B (Agent A) # evs1 \<in> woolam"



                 (*****PASSO WL2*****)


(*B risponde ad A inviando un messaggio nonce. Si noti che nella prima parte dell'evs2 si ripete
il messaggio dell'evs1, con A' al posto di A.*)


WL2:  "[| evs2 \<in> woolam;  Says A' B (Agent A) \<in> set evs2 |]
       ==> Says B A (Nonce NB) # evs2 \<in> woolam"



                (*****PASSO WL3*****)


(*A risponde allo scambio di B cifrando il nonce di B con la chiave che condivide con il server
S.*)

WL3:  "[| evs3 \<in> woolam;
          Says A  B (Agent A)  \<in> set evs3;
          Says B' A (Nonce NB) \<in> set evs3 |]
       ==> Says A B (Crypt (shrK A) (Nonce NB)) # evs3 \<in> woolam"
```

```
                (*****PASSO WL4*****)


(*B spedisce al server S la risposta di A.*)


WL4:  "[| evs4 \<in> woolam;
          Says A'  B X        \<in> set evs4;
          Says A'' B (Agent A) \<in> set evs4 |]
       ==> Says B Server {|Agent A, Agent B, X|} # evs4 \<in> woolam"



                (*****PASSO WL5*****)


(*Il server S decifra la risposta di A per l'agente B e gliela spedisce.*)


  WL5:  "[| evs5 \<in> woolam;
           Says B' Server {|Agent A, Agent B, Crypt (shrK A) (Nonce NB)|}
            \<in> set evs5 |]
        ==> Says Server B (Crypt (shrK B) {|Agent A, Nonce NB|})
             # evs5 \<in> woolam"

(*****************************************************************************)


declare Says_imp_knows_Spy [THEN analz.Inj, dest]
declare parts.Body  [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un  [dest]


(************************DIMOSTRAZIONE*********************************)


(***Proprietà di possibilità***)


(* Esiste un server S tale che, per ogni agente B esiste un messaggio nonce NB tale che il
messaggio finale "S -> B : {|A, NB|}KBS" sia trasmesso.*)

lemma "\<exists>NB. \<exists>evs \<in> woolam.
          Says Server B (Crypt (shrK B) {|Agent A, Nonce NB|}) \<in> set evs"


apply (intro exI bexI)

apply (rule_tac [2] woolam.Nil
                [THEN woolam.WL1, THEN woolam.WL2, THEN woolam.WL3,
                 THEN woolam.WL4, THEN woolam.WL5], possibility)

done
```

(**Comportamento della spia**)

(*La spia non può mai ottenere la chiave condivisa di un agente onesto.*)

lemma Spy_see_shrK [simp]:
    "evs \<in> woolam ==> (Key (shrK A) \<in> parts (spies evs)) = (A \<in> bad)"

by (erule woolam.induct, force, simp_all, blast+)

lemma Spy_analz_shrK [simp]:
    "evs \<in> woolam ==> (Key (shrK A) \<in> analz (spies evs)) = (A \<in> bad)"

by auto

lemma Spy_see_shrK_D [dest!]:
    "[|Key (shrK A) \<in> parts (knows Spy evs); evs \<in> woolam|] ==> A \<in> bad"

by (blast dest: Spy_see_shrK)

(**PASSO WL4**)

(***Proprietà di Autenticità***)

(* Se un messaggio è stato generato da un agente onesto o ancora meglio, da un server di fiducia, ed effettivamente appare il suo nome a certificare la provenienza del messaggio , allora il messaggio gode della proprietà di autenticità.*)

lemma NB_Crypt_imp_A_msg:

    "[| Crypt (shrK A) (Nonce NB) \<in> parts (spies evs);
       A \<notin> bad; evs \<in> woolam |]
     ==> \<exists>B. Says A B (Crypt (shrK A) (Nonce NB)) \<in> set evs"

by (erule rev_mp, erule woolam.induct, force, simp_all, blast+)

(***Garanzia per il server***)

(* Se il server ottiene un messaggio contenente un certificato da parte di A, allora questo significa che è stato prodotto solamente da A. Non siamo però certi che B lo abbia già visto: questa opportunità potrebbe spingere la spia a ridirigere il messaggio al server. *)

lemma Server_trusts_WL4 [dest]:
    "[| Says B' Server {|Agent A, Agent B, Crypt (shrK A) (Nonce NB)|}
         \<in> set evs;
       A \<notin> bad; evs \<in> woolam |]
     ==> \<exists>B. Says A B (Crypt (shrK A) (Nonce NB)) \<in> set evs"

by (blast intro!: NB_Crypt_imp_A_msg)

(**PASSO WL5**)

(* Il server trasmette il messaggio WL5 soltanto se riceve la giusta forma del messaggio. *)

```
lemma Server_sent_WL5 [dest]:
    "[| Says Server B (Crypt (shrK B) {|Agent A, NB|}) \<in> set evs;
       evs \<in> woolam |]
     ==> \<exists>B'. Says B' Server {|Agent A, Agent B, Crypt (shrK A) NB|}
           \<in> set evs"

by (erule rev_mp, erule woolam.induct, force, simp_all, blast+)
```

(* Se compare il messaggio cifrato, allora ha comunicato con il server! *)

```
lemma NB_Crypt_imp_Server_msg [rule_format]:
    "[| Crypt (shrK B) {|Agent A, NB|} \<in> parts (spies evs);
       B \<notin> bad;  evs \<in> woolam |]
     ==> Says Server B (Crypt (shrK B) {|Agent A, NB|}) \<in> set evs"

by (erule rev_mp, erule woolam.induct, force, simp_all, blast+)
```

(***Garanzia per il responder B***)

(* Se B ottiene il certificato del server allora A ha cifrato il nonce usando la sua chiave. Questo evento non può essere più vecchio del nonce stesso. Tuttavia A potrebbe trasmettere il nonce ad un altro agente e raggiungere il server attraverso una spia. *)

```
lemma B_trusts_WL5:
    "[| Says S B (Crypt (shrK B) {|Agent A, Nonce NB|}): set evs;
       A \<notin> bad;  B \<notin> bad;  evs \<in> woolam  |]
     ==> \<exists>B. Says A B (Crypt (shrK A) (Nonce NB)) \<in> set evs"


by (blast dest!: NB_Crypt_imp_Server_msg)

end
```

****

Bibliografia

[AB-NE 94] : M. Abadi and R. Needham. Prudent engineering practice for cryptography protocols.

Tecnical Report RR 125, Digital Systems Research Center, June 1994.

[BAR 94] : Henk Barendregt, Erik Barendsen. *Introduction to Lambda Calculus*. Revised edition, October 1994

[BELLA 00] G. Bella. Inductive Verification of Cryptographic Protocols. PhD Dissertation. University of Cambridge, 2000

[BO 96] : 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.

[BU-AB-NE 90] : Michael Burrows, Mart'in Abadi, and Roger Needham. A Logic of Authentication. *ACM Transactions in Computer Systems*, 8(1):18--36, February 1990.

[DO-YAO 83] : D. Dolev and A. Yao. On The Security of Public Key Protocols. *Transactions on Information Theory*, 29(2):198-208, 1983.

[EV-GO 83] :S. Even and O. Goldreich – On the Security of multi-party ping-pong protocols. In

Proc. of the 24th IEEE Sym. on the Foundations of Comp. Sci . IEEE Comp. Society Press, 1983.

[FA'-HE-GUT- 98] : J. T. Fàbrega, J. C. Herzog, and J. D. Guttman. Strand Spaces: Why is a Security Protocol Correct? In Proc. of the 17th IEEE Sym. on Sec. and Privacy. IEEE Comp. Society Press, 1998.

[KE 91] : R.A. Kemmerer. Analyzing encryption protocols using formal techniques. IEEE Journal on Selected Areas in Communications, 7(4):448--457, May 1989.

[KE-MEA-MIL 94] : R. Kemmerer, C. Meadows, J. Millen. Three Systems for Cryptographic Protocol Analysis. *Journal of Cryptology*, 7(2), 1994.

[GO 88] : Michael J. C. Gordon. HOL: a proof generating system for higher-order logic. In Birtwistle and Subrahmanyam 1988, pages 73-128.

[Löf 84] : Martin-Löf. *Intuitionistic type theory*. Bibliopolis 1984

[LO-RI 92] : D. Longley and S. Rigby. An automatic search for security flaws in key management schemes. *Computers and Security*, 11(1):75–90, 1992.

[LOWE 96 A] : G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software - Concepts and Tools*, 17:93–102, 1996.

[LOWE 96 B] : G. Lowe. A Hierarchy of Authentication Specifications. Technical Report 1996/33, Department of Mathematics and Computer Science, 1996.

[MEA 91] : 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.

[MEA 92] : C. Meadows. Applying Formal Methods to the Analysis of a Key Management Protocol. *Journal of Computer Security*, 1:5–53, 1992. Press, June 2002.

[MEA 03] :C. Meadows. Formal Methods for Cryptographic Protocols: Emerging Issues and Trends

[MEN 96] : Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

[NES 90] : D. M. Nessett. A Critique of the Burrows, Abadi, and Needham Logic. Operating Systems Review, 24(2):35--38, April 1990.

[NÖ 93] : Philippe Nöel. Experimenting with Isabelle in ZF set theory. *Journal of Automated Reasoning*, 10(1):15-58, 1993.

[PAU 86] : L. C. Paulson. Natural deduction as higher-order resolution. *Journal of Logic Programming*, 3:237-258, 1986.

[PAU 87] : L. C. Paulson. *Logic and Computation: Interactive proof with Cambridge LCF*. Cambridge University Press 1987.

[PAU 89] : Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363-397, 1989.

[PAU 98] : L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. In *Journal of Computer Security*, 6:85-128, 1998.

[RY-SCH 00] : P. Y. A. Ryan and S. A. Schneider. The Modelling and Analysis of Security Protocols: the CSP Approach. Addison Wesley Publ. Co., Reading, Massachussetts, 2000.

[SAN-WAL 01] : D. Sangiorgi and D. Walker. The π-calculus: a Theory of Mobile Processes, Cambridge University Press, 2001.

[SCH 97] : S. Schneider. Verifying Authentication Protocols with CSP. In PCSFW: Proceedings of the 10th Computer Security Foundation Workshop. IEEE Computer Society Press, 1997.

[SIM 00] : P. Simons. Basic of Computer Security: Cryptographic Protocol. Institute of Computer Science, Rheinische Friedrich-Wihelms-Universit¨at Bonn, March 2000.

[SOKO 87] : Stefan Sokolowski (1987). Soundness of Hoare's logic: an automatic proof using LCF. *ACM Transactions on Programming Languages and Systems*, 9:100-120.

[SPI 88] : J. M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*. Cambridge University Press, 1988.

[SYL 90] : P. Syverson. Formal Semantics for Logics of Cryptographic Protocols. In Proceedings of the Computer Security Foundations Workshop III, pages 32--41. IEEE Computer Society Press, June 1990.

[SUP 72] : Patrick Suppes. *Axiomatic Set Theory*. Dover, 1972.

[TOU 93] : M. J. Toussaint. Formal Verification of Probabilistic Properties in Cryptographic Protocols.Lecture Notes in Computer Science, 739, 1993.

[WIK 87] : Å. Wikström (1987). *Functional Programming using ML*. Prentice-Hall International.

[W-L 92]: T.Y.C. Woo and S.S. Lam. Authentication for distributed systems. *Computer*, 25(1):39--52, January 1992.