

Verifica di protocolli di sicurezza crittografici

Cristiano Longo

4 Aprile 2003

Indice

1	Introduzione	4
2	Protocolli di sicurezza	6
2.1	Obbiettivi dei protocolli di sicurezza	6
2.1.1	Confidenzialità	6
2.1.2	Autenticità	7
2.1.3	Unicità	9
2.1.4	Autenticazione	12
2.1.5	Non repudiabilità	16
2.2	Assunzioni e limiti teorici	17
2.3	Strumenti a disposizione degli agenti	17
2.3.1	Crittografia	18
2.3.2	Funzioni hash	19
2.3.3	Firma digitale	20
2.3.4	Generazione di elementi <i>freschi</i>	20
2.4	Protocolli di seconda generazione	21
3	Metodi formali per la verifica di protocolli di sicurezza :	
	l'approccio induttivo	22
3.1	Belief logic	23
3.2	CSP	23
3.3	L'approccio induttivo	24
3.4	Gli agenti	25
3.5	Chiavi	25
3.6	Messaggi	26
3.6.1	Messaggi atomici	26
3.6.2	Operatori sui messaggi	26
3.7	Eventi	27
3.8	Tracce	27

3.9	Conoscenza degli agenti : l'operatore <i>knows</i>	27
3.10	Operatori su insiemi di messaggi	29
3.10.1	Parts	29
3.10.2	Analz	30
3.10.3	Synth	30
3.11	Formalizzazione del protocollo	31
4	Certified mail protocol	33
4.1	Prerequisiti	33
4.2	Obbiettivi del protocollo	34
4.2.1	Obbiettivi secondari	35
4.3	Livelli di autenticazione	35
4.4	Descrizione del protocollo	36
5	Formalizzazione dei canali di comunicazione	39
5.1	Canali convenzionali	39
5.1.1	Formalizzazione	40
5.1.2	Come rendere questa formalizzazione davvero realistica	40
5.2	Canali di comunicazione con consegna garantita	40
5.2.1	Formalizzazione	41
5.2.2	Perche' questa formalizzazione non indebolisce il mod- ello	41
5.3	Canale di comunicazione confidenziale con consegna garantita	42
5.3.1	Formalizzazione	43
5.3.2	Come rendere questa formalizzazione davvero realistica	44
6	Formalizzazione del protocollo	45
6.1	Strumenti : teoria <i>Traceability</i>	45
6.2	Elementi base del protocollo	46
6.2.1	Password	46
6.2.2	Chiavi del server	46
6.2.3	Meccanismo di autenticazione richiesta-risposta	46
6.2.4	Modalità di autenticazione	47
6.3	Formalizzazione del protocollo	47
6.3.1	Passo 1: l'agente S inizia il protocollo	47
6.3.2	Passo 2: l'agente R richiede a TTP la chiave	48
6.3.3	Passo 3: TTP rilascia la chiave ed invia il certificato	49

7	Analisi delle proprietà del protocollo	50
7.1	Linea dimostrativa	50
7.2	Autenticità del certificato	51
7.2.1	Esempio di dimostrazione per induzione	52
7.3	TTP invia il certificato e la chiave contemporaneamente . . .	56
7.4	R conosce la chiave	57
7.5	Autenticità della richiesta	58
7.5.1	Autenticazione NOAuth ed SAuth, un possibile attacco	58
7.5.2	Autenticazione TTPAuth	59
7.6	R conosce il messaggio criptato	60
7.6.1	Autenticazione TTPAuth e BothAuth, una esecuzione <i>anomala</i>	61
7.6.2	Il certificato afferma che il destinatario conosce il mes- saggio criptato	61
7.7	Il certificato afferma una avvenuta ricezione	63
7.8	Il mittente non è in grado di sintetizzare il certificato	64
8	Una possibile variante del protocollo	67
8.1	Il sistema postale : raccomandate con ricevuta di ritorno . . .	67
8.2	Prima variante: il mittente firma il messaggio	68
8.3	Seconda variante: il mittente esibisce un documento all'ufficio delle postale	69
8.4	Il protocollo di e-mail certificata nella realtà	70
8.5	Terza variante : certificazione del mittente e del ricevente attraverso la firma	71
8.5.1	Passo 1	71
8.5.2	Passo 2	72
8.5.3	Passo 3	72
8.5.4	Osservazioni	73
9	Conclusioni	75
10	Isabelle/Isar files	78
10.1	Message.thy	78
10.2	Message.ML	82
10.3	EventTraceability.thy	82
10.4	Traceability.thy	84
10.5	CertifiedMail.thy	95

Capitolo 1

Introduzione

L'avvento e l'espansione della rete mondiale Internet ha aperto un ventaglio infinito di possibilità per l'uso dei calcolatori elettronici: dalla consultazione e gestione di archivi remoti alla programmazione distribuita, dalle videoconferenze al commercio elettronico. Tali impieghi possono essere riassunti con un unico scopo della rete : permettere la comunicazione tra due o più calcolatori lontani.

Come nelle reti interne(LAN) le interazioni tra calcolatori vengono regolate da **protocolli**. Ogni protocollo è specifico per un tipo di interazione. Abbiamo ad esempio protocolli per la posta elettronica, protocolli per il trasferimento dati, protocolli per la consultazione di documenti, e così via.

In generale un protocollo di rete consiste in un insieme di passi che due o più calcolatori devono effettuare per portare a termine una transazione. I protocolli sono costituiti da un insieme di **messaggi** che gli **agenti** coinvolti si scambiano mediante la rete, e da una serie di **passi** che regolano il comportamento di tali agenti.

Tuttavia, nel momento in cui avvenga una interazione tra insieme di agenti, uno o più degli agenti coinvolti, per fini *fraudolenti*, potrebbero non eseguire correttamente il protocollo che regola tale tipo di interazione. Un evento di questo tipo viene definito **attacco**. Nel migliore dei casi gli agenti *onesti* si rendono conto di questo e l'interazione termina. Nel caso peggiore, l'esecuzione del protocollo viene completata e gli agenti *maliziosi* raggiungono i propri scopi. In questo caso si dice che l'attacco è andato a buon fine. Per difendersi da tali attacchi sono nati i **protocolli di sicurezza**. Nella prima parte di questo documento verranno innanzitutto descritte e analizzate le più comuni problematiche di sicurezza, e alcune soluzioni presentate per rispondere a tali problematiche. Verranno descritti dei metodi *formali*

attraverso i quali determinare se un dato protocollo risponde o meno a certi requisiti di sicurezza. In particolare verrà descritto l'approccio induttivo dovuto a Laurence Paulson.

Nella seconda parte verrà data la definizione di protocollo di *seconda generazione*, e verrà affrontato il problema di come formalizzare tali protocolli nel modello induttivo. Come rappresentante di tale classe di protocolli verrà presentato un protocollo di e-mail certificata. Oltre a rivestire un particolare interesse per la sua natura di protocollo di seconda generazione, tale protocollo ci darà modo di discutere le problematiche nella formalizzazione e nella verifica di protocolli che soddisfano requisiti di *non repudiabilità*. Verrà infatti presentata la formalizzazione di tale protocollo e la verifica delle sue proprietà. Infine verranno presentate alcune varianti di tale protocollo che intendono garantire proprietà non garantite dal protocollo stesso.

Capitolo 2

Protocolli di sicurezza

L'avvento di Internet ha messo in risalto alcune problematiche riguardanti la comunicazione e l'esecuzione dei protocolli di comunicazione. La topologia della rete permette infatti di intercettare i messaggi che intercorrono tra due agenti. In oltre a priori è impossibile stabilire con certezza il mittente di un messaggio. Tali debolezze possono venire sfruttate da utenti maliziosi per gli scopi più diversi.

2.1 Obbiettivi dei protocolli di sicurezza

In generale un protocollo è detto protocollo di sicurezza se raggiunge almeno uno dei seguenti obiettivi :

2.1.1 Confidenzialità

Dato un messaggio m , un insieme di agenti \mathcal{S} ed un protocollo \mathcal{P} , si dice che \mathcal{P} gode della proprietà di confidenzialità rispetto al messaggio m ed all'insieme di agenti \mathcal{S} se, nell'ipotesi che il messaggio m prima dell'esecuzione di \mathcal{P} sia noto al più a tutti gli agenti compresi in \mathcal{S} e a nessun altro, tale ipotesi sia verificata anche dopo l'esecuzione di \mathcal{P} .

Esaminiamo nel dettaglio le varie parti dell'enunciato, definendo innanzitutto la natura dell'insieme \mathcal{S} . L'esistenza stessa del messaggio m implica l'esistenza di un agente A che lo ha generato. Avendolo generato, sicuramente A è a conoscenza del messaggio m . Per tale motivo l'insieme \mathcal{S} è di certo non vuoto e comprende A . Supponiamo che A , utilizzando il protocollo \mathcal{P} , intenda spedire un messaggio m agli agenti B, C, D in maniera

confidenziale. Al termine dell'esecuzione del protocollo non deve esistere nessun altro agente, all'infuori di B , C , D ed A stesso, che sia a conoscenza di m . Se tale proprietà è soddisfatta, possiamo affermare che \mathcal{P} soddisfa il requisito di confidenzialità rispetto al messaggio m ed all'insieme di agenti $\{ A, B, C, D \}$. Non è necessario che tutti gli agenti compresi in questo insieme vengano a conoscenza del messaggio. È necessario che, se un agente E venga a conoscenza di m , esso sia compreso nell'insieme \mathcal{S} . Per questo motivo, un protocollo che goda della proprietà di confidenzialità rispetto ad un insieme di agenti \mathcal{S} , gode di tale proprietà anche rispetto a tutti gli insiemi \mathcal{S}' tali che \mathcal{S} sia compreso in \mathcal{S}' .

Esempio di protocollo che gode della proprietà di confidenzialità

Esistono vari *strumenti* studiati per garantire la confidenzialità di un messaggio. Prendiamo in considerazione un protocollo di esempio che utilizzi la **crittografia** per garantire questo requisito. Un algoritmo crittografico trasforma un messaggio m in un'altro messaggio m' , tale che si possa ricavare da m' il messaggio originale solo conoscendo un parametro aggiuntivo, denominato chiave. Supponiamo allora che l'agente A intenda spedire in maniera confidenziale un messaggio m all'agente B . A deve occuparsi innanzitutto di comunicare a B la chiave da utilizzare. Questo può avvenire utilizzando un ulteriore protocollo per la distribuzione di chiavi, o attraverso un'incontro personale di A e B . La seconda opportunità non è remota come si possa pensare, considerando che tale chiave potrebbe essere utilizzata non per una singola comunicazione, ma per tutti i messaggi che A intenderà spedire a B in via confidenziale nel futuro. Sia allora k tale chiave. A non deve fare altro che *codificare* il messaggio che intende spedire in modo tale che possa essere decodificato utilizzando k , e spedirlo a B . Nell'ipotesi che A e B siano gli unici agenti a conoscenza di tale chiave, nessun altro è in grado di risalire al messaggio originale da quello codificato e trasmesso. Questo protocollo offre garanzia di confidenzialità rispetto al messaggio m ed all'insieme di agenti $\{ A, B \}$.

2.1.2 Autenticità

Dato un messaggio m , un protocollo \mathcal{P} , si dice che \mathcal{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.

La necessità di protocolli che soddisfino un requisito di questo genere nasce direttamente dalle imperfezioni delle reti. In una rete LAN e simil-

mente su internet, il mittente del messaggio viene inserito nell'intestazione del messaggio da chi lo invia, dandogli modo potenzialmente di spacciarsi per chiunque. Su internet il trasporto di un messaggio da mittente e destinatario avviene attraverso una serie di nodi intermedi. Nulla vieta ad uno di questi di modificare l'intestazione di un messaggio prima di trasmetterlo al nodo successivo lungo il percorso. Queste considerazioni rendono evidente il fatto che nessuna garanzia può essere data sul mittente di un messaggio. Tuttavia il requisito di autenticità offre garanzie in merito all'agente che ha *generato* il messaggio, non in merito al mittente del messaggio. Per generazione di un messaggio si intende il momento nel quale tale messaggio è stato immesso sulla rete per la prima volta. Offrire quindi garanzia di autenticità di un messaggio ad un agente rispetto ad un messaggio significa fornirgli gli strumenti per determinare da chi tale messaggio è stato trasmesso originariamente, non se tale messaggio sia stato successivamente riutilizzato da altri. Affinchè tale requisito possa essere soddisfatto è necessario che il messaggio in questione *parli* in qualche modo di chi lo ha generato, in maniera esplicita o implicita.

Esempio di protocollo che soddisfa il requisito di autenticità

Il protocollo mostrato a titolo di esempio per chiarire il concetto di confidenzialità si presta anche come esempio di protocollo che offre il requisito di autenticità. Con una precisazione. Indichiamo con $\{m\}_k$ un messaggio cifrato, tale che sia necessario conoscere k per ricavarne l'originale m . La funzione utilizzata per ottenere m da $\{m\}_k$ è detta funzione di *decifrazione*.

$$d(k, \{m\}_k) = m$$

Essa dipende dall'algoritmo crittografico utilizzato, e si suppone che tutti gli agenti siano in grado di utilizzarla. Essa restituisce un risultato corretto solo se la chiave utilizzata è a sua volta corretta. Nel caso in cui venga utilizzata una chiave diversa da quella utilizzata per la cifratura, viene restituito un messaggio diverso dall'originale.

$$d(k', \{m\}_k) = m'$$

$$k \neq k' \longrightarrow m \neq m'$$

Quando un agente riceve un messaggio cifrato $\{m\}_{k'}$, con opportuni accorgimenti è in genere in grado di determinare se tale messaggio sia stato

cifrato o no con una determinata chiave k . Questa capacità nasce direttamente dalla capacità di determinare se il risultato dell'operazione di decifrazione sia o no corretto. Supponiamo ad esempio che il destinatario sia in attesa di un messaggio contenente un testo in lingua italiana. Alla ricezione di $\{m\}_{k'}$, applica a tale messaggio la funzione di decodifica usando la chiave k .

$$d(k, \{m\}_{k'}) = m'$$

Se m' non contiene un testo in italiano, allora sicuramente $k' \neq k$. In oltre, le probabilità che un utente malizioso riesca, non conoscendo k , a generare un messaggio m tale che $d(k, \{m\}_{k'})$ sia un testo in italiano tendono a zero al crescere della lunghezza di m . Nel caso in cui non si possano fare assunzioni così decisive sul contenuto del messaggio, si potrebbe pretendere che tale messaggio venga accompagnato da una checksum per verificarne la corretta decodifica.

Supponiamo che un agente B riceva un messaggio $\{m\}_{k'}$, e che esista una chiave k nota solo ad un altro agente A ed a B stesso. Sia l'agente B in grado di determinare se il messaggio ricevuto è o meno un messaggio cifrato con la chiave k . Nel caso che tale controllo dia esito positivo, essendo tale chiave nota solo a B e A , tale messaggio è stato di certo generato da B oppure da A . B è ovviamente in grado di determinare se tale messaggio sia o meno stato generato da egli stesso. In caso contrario di certo tale messaggio è stato generato da A .

2.1.3 Unicità

Dato un messaggio composto

$$m = \{m_1, m_2, m_3, \dots, m_n\}$$

Sia \mathcal{I} un insieme di indici compresi fra 1 ed n .

Dato un protocollo \mathcal{P} , si dice che \mathcal{P} gode della proprietà di unicità rispetto al messaggio composto m e all'insieme di indici \mathcal{I} se, per ogni messaggio $m' = \{m'_1, m'_2, \dots, m'_n\}$ che viene spedito sulla rete durante lo svolgimento di una sessione del protocollo, vale il seguente predicato

$$\forall i \in \mathcal{I} \ m_i = m'_i \longrightarrow \forall i \in \{1 \dots n\} \ m_i = m'_i$$

Le componenti indicate nell'insieme \mathcal{I} caratterizzano univocamente il messaggio m . Tale insieme è di solito costituito da un'unico indice che rappresenta un messaggio atomico in grado da solo di identificare la sessione.

Un requisito di questo genere di solito viene infatti utilizzato per stabilire una corrispondenza tra un messaggio inviato durante una sessione del protocollo e la sessione stessa. Supponiamo che un agente A spedisca un messaggio a B per ottenere una informazione legata in qualche modo allo stato di B nel momento nel quale A spedisce il messaggio. Ad esempio A potrebbe voler conoscere il valore di un contatore interno a B . A in un certo istante A invia a B un messaggio tramite il quale manifesta a B la volontà di conoscere il valore di tale contatore. Supponiamo che tale contatore, nel momento in cui B riceve questa richiesta, abbia un determinato valore v . B allora invia ad A un altro messaggio m_B nel quale indica il valore v . Un utente malizioso C intercetta tale messaggio m_B . Successivamente A intende conoscere il nuovo valore che tale contatore interno a B ha assunto. Indichiamo tale valore con v' . Ovviamente v' non necessariamente coincide con v . A invia nuovamente la richiesta all'agente B . Tuttavia tale richiesta viene intercettata da C , che invia ad A il messaggio m_B ottenuto dalla sessione precedente.

- Il valore del contatore interno a B assume il valore v .
- A invia a B il messaggio $m_A =$ comunicami il valore del tuo contatore interno.
- B invia ad A il messaggio $m_B =$ il valore del mio contatore interno è v .
- Il messaggio m_B viene intercettato da C e da questi ritrasmesso ad A .
- A riceve il messaggio m_B deducendo che il valore del contatore interno a B sia v .
- ...
- il valore del contatore interno a B assume un nuovo valore v' .
- ...
- A invia a B il messaggio $m'_A =$ comunicami il valore del tuo contatore interno.
- Tale messaggio viene intercettato da C .
- C invia ad A il messaggio m_B , nel quale era indicato il valore v .
- A , quando riceve tale messaggio, si convince che il contatore interno a B abbia ancora il valore v , mentre tale contatore già prima dell'inizio della sessione aveva assunto il valore v' .

Nell'esecuzione presentata il requisito di autenticità del messaggio m_B non viene violato, in quanto tale messaggio è effettivamente stato generato da B . Tuttavia esso è stato generato da B per essere utilizzato in una sessione precedente. Il fatto che A non sia in grado di stabilire se il messaggio che riceve si riferisca alla sessione attuale o ad una svolta precedentemente causa il fallimento del protocollo.

Per rendere impossibile una esecuzione quale quella indicata, è necessario inserire nel messaggio m_B un elemento che indichi a quale sessione si riferisca. Soprattutto è necessario che anche l'agente A sia in grado di discernere se tale messaggio si riferisce alla sessione corrente o ad una svolta precedentemente. Supponiamo allora che l'agente A sia in grado di generare un elemento *nuovo* n . Finchè tale n non viene inviato sulla rete, A ha la certezza di essere l'unico agente a conoscenza dello stesso. In oltre, poichè tale elemento non era mai stato utilizzato, ogni messaggio presente sulla rete che contenga n di certo è stato generato successivamente ad n stesso. Presentiamo una versione del protocollo per la quale, nell'ipotesi che A sia in grado di autenticare m_B , non soffra dell'inconveniente visto prima :

- A genera un elemento nuovo n (ad esempio un numero casuale)
- A invia a B il messaggio $m_A =$ comunicami il valore del tuo contatore interno, la sessione è identificata da n .
- B invia ad A il messaggio $m_B =$ il valore del mio contatore interno, per la sessione identificata da n , è v .

L'agente A , ricevendo m_B , poichè tale m_B contiene l'identificativo di sessione n , è sicuro che tale messaggio è stato generato in un istante successivo alla generazione di tale identificativo. Quindi tale messaggio non riguarda una sessione precedente, ma la sua generazione è di certo successiva all'inizio della sessione in corso. Nell'ipotesi che nessun altro agente sia in grado di generare il messaggio m_B oltre B stesso, prevedendo quindi un sistema per autenticare tale messaggio, e nell'ipotesi che B agisca onestamente, comunicando il valore che il contatore assume al momento nel quale la richiesta viene ricevuta, il valore di tale contatore e l'identificativo di sessione risultano strettamente legati. In particolare non esistono due messaggi nei quali viene indicato lo stesso numero di sessione e due valori diversi per il contatore. Il protocollo in questione gode allora del requisito di unicità del messaggio m_B rispetto all'elemento n , compreso in m_B .

2.1.4 Autenticazione

Supponiamo che un agente A porti a termine il protocollo \mathcal{P} con un agente B . L'obiettivo di autenticazione, secondo la formalizzazione di Lowe, può essere raggiunto dal protocollo \mathcal{P} in 4 livelli:

Aliveness l'agente A è sicuro che B ha eseguito il protocollo.

Weak agreement l'agente A è sicuro di aver eseguito il protocollo con B . Questo è il significato che si asserisce comunemente al termine autenticazione

Non-injective agreement l'agente A è sicuro di aver eseguito il protocollo con B ed inoltre A e B concordano su un insieme di messaggi \mathcal{H} .

Injective agreement l'agente A è sicuro di aver eseguito il protocollo con B , A e B concordano su un insieme di messaggi \mathcal{H} e B non risponde più di una volta nella sessione del protocollo.

Ognuno di questi quattro livelli sussume il precedente. Ad esempio, se l'agente A è sicuro di aver eseguito il protocollo con B , allora chiaramente B ha eseguito il protocollo. Le situazioni nelle quali garantire tale requisito si rende necessario sono fondamentalmente due :

1. un agente A vuole mettere a conoscenza B di una certa informazione, ed accertarsi che B abbia effettivamente ricevuto tale informazione;
2. un agente B richiede l'accesso ad alcuni servizi forniti da un server S , ed il server S vuole accertarsi dell'effettiva identità di chi ha effettuato tale richiesta.

Esaminiamo nel dettaglio le quattro forme del requisito di autenticazione, e come spesso queste si riconducano ad altri requisiti di sicurezza.

Aliveness

Come si deduce dal nome stesso, un requisito di tal genere offre garanzie ad un agente A che un secondo agente B sia *attivo*. Per offrire una garanzia di questo genere è necessario innanzitutto offrire garanzia di autenticità rispetto qualche messaggio che si presume inviato da B . La presenza o meno di B può essere dedotta solo dall'esistenza o meno di messaggi sulla rete sicuramente generati da B . Nel caso in cui l'agente A voglia accertarsi che

le tracce di attività lasciate da B siano recenti, è necessario che dai messaggi che manifestano la presenza di B possano essere dedotte informazioni di carattere temporale. Per offrire queste garanzie bisogna ricondursi alle considerazioni riguardanti il requisito di unicità. Supponiamo infatti che esista un server S , nel quale A ripone completa fiducia, che ogni ora generi un *nuovo* numero casuale, e lo distribuisca a tutti gli agenti sulla rete. Sia n il numero distribuito all'ora h . Se un messaggio comprende al suo interno tale n , di certo la sua generazione risale all'ora h o è al più successiva. In tal modo, se l'agente A entra in possesso di un messaggio sicuramente riconducibile a B che contenga il numero n , può avere la certezza che l'agente B all'ora h era ancora attivo. Vari aspetti della generazione di elementi del tutto nuovi verranno presi in considerazione nei paragrafi successivi.

Weak agreement

Questo livello di autenticazione prevede non solo che un agente A sia in grado di determinare se B ha eseguito il protocollo recentemente, ma anche se B in questa esecuzione A stesso sia stato coinvolto oppure no. Per chiarire questo concetto, esaminiamo un protocollo che non è in grado di garantire questo grado di autenticazione: il protocollo Needham-Schroeder[13]. Tale protocollo sfrutta degli elementi verranno che approfonditi successivamente in questo documento : la crittografia a doppia chiave e la capacità da parte degli agenti di generare numeri casuali mai utilizzati prima sulla rete. La crittografia a doppia chiave si basa su un insieme di coppie di chiavi: una chiave *pubblica* ed una *privata*. Ogni coppia viene assegnata ad un unico agente e non esistono due agenti ai quali venga assegnata la stessa coppia di chiavi. Ogni agente conosce tutte le chiavi pubbliche, ma solo il proprietario conosce la propria chiave privata. Una scrittura del tipo $\{x\}_{K_a}$ indica un messaggio cifrato con la chiave pubblica dell'agente A . Essendo tale chiave pubblica, qualsiasi altro agente a partire da x è in grado di generare un messaggio di questo tipo. Per decifrare questo messaggio è invece necessario conoscere la chiave privata dell'agente A . Ma solo A stesso è a conoscenza di tale chiave. Per questo motivo la crittografia a doppia chiave rappresenta un utile strumento per la trasmissione di dati confidenziali. Ogni agente è in grado di spedire un messaggio ad un'altro agente B , cifrandolo con la chiave pubblica di B . In questo modo solo B sarà in grado di decifrare questo messaggio. A differenza del protocollo mostrato come esempio per il requisito di autenticità, non è necessario presupporre che, prima della comunicazione, i due agenti si siano scambiati in maniera confidenziale alcunchè. Siamo ora in grado di discutere il protocollo Needham-Schroeder.

1. L'agente A genera un elemento nuovo N_a , e spedisce a B il messaggio $\{N_a, A\}_{Kb}$.
2. L'agente B , alla ricezione di questo messaggio, genera a sua volta un'altro elemento nuovo N_b e spedisce ad A il messaggio $\{N_a, N_b\}_{Ka}$.
3. L'agente A spedisce infine a B il messaggio $\{N_b\}_{Kb}$

Essendo l'elemento N_a un elemento nuovo, alla ricezione del messaggio indicato al passo 2 l'agente A ha la certezza che tale messaggio riguarda la sessione corrente. In oltre A spedisce N_a al passo 1 cifrato con la chiave pubblica di B . In tal modo si assicura che solo B possa venire a conoscenza di N_a . Questo basta ad affermare l'autenticità del messaggio spedito al passo 2. Grazie a queste due considerazioni l'agente A quando riceve il messaggio spedito al passo 2 ha la certezza che B stia eseguendo il protocollo. In oltre il messaggio invitato al passo 2, sicuramente generato da B , viene inviato cifrato con la chiave pubblica di A . Questo manifesta la convinzione di B che all'altro capo della comunicazione ci sia A . Considerazioni analoghe sui passi 2 e 3 ci portano ad affermare che anche B , al termine dell'esecuzione del protocollo, abbia la certezza di aver eseguito il protocollo con A .

Nonostante questa dimostrazione informale ci abbia convinto che il protocollo offra garanzie di autenticazione, nella forma *week agreement*, porteremo un controesempio per dimostrare che così non è. Supponiamo che un agente A decida di iniziare una sessione del protocollo con un altro agente B . A tal fine A invia a C il messaggio $\{N_a, A\}_{Kc}$. C è in grado di leggere il contenuto di questo messaggio. Ricava allora da questo N_a e invia il messaggio $\{N_a, A\}_{Kb}$ ad un terzo agente B . Ricevendo tale messaggio, l'agente B crede che A abbia intenzione di eseguire con lui una sessione del protocollo. Per questo motivo invia ad A il messaggio $\{N_a, N_b\}_{Ka}$. Vedendosi recapitare tale messaggio A è convinto che sia la risposta che attendeva da C . Per questo motivo termina la sessione del protocollo inviando a C il messaggio $\{N_b\}_{Kc}$. A questo punto l'agente A ha la certezza di aver eseguito il protocollo con C . Anche se questo non è del tutto vero, tuttavia C era effettivamente attivo durante la sessione appena terminata, ed inoltre ha preso parte alla sessione stessa. Avendo ricevuto da A il messaggio $\{N_b\}_{Kc}$ C è in grado di ricavare N_b da tale messaggio. Può quindi trasmettere il messaggio $\{N_b\}_{Kb}$ all'agente B , concludendo la sessione del protocollo che aveva aperto con questo. Alla ricezione di questo messaggio, B è convinto di aver eseguito il protocollo con A , ma questo non è affatto vero, poichè il suo effettivo interlocutore era C . L'attacco appena presentato fa riflettere su

quanto una proprietà di questo genere sia difficile da garantire, e su quante insidie un'analisi informale di un protocollo non sia in grado di svelare.

Non injective agreement

Offrire una garanzia di autenticità con accordo non iniettivo significa dare i mezzi ad un agente per verificare se le informazioni che ha inteso comunicare sono giunte correttamente a destinazione. Viene richiesto non solo che un agente A sia in grado di determinare se B ha eseguito il protocollo con lui, ma anche che al termine della sessione B abbia ricevuto le informazioni che A ha inteso comunicargli. Un requisito di tale tipo viene di solito garantito prevedendo che B spedisca, al termine del protocollo, un *riassunto* delle informazioni ottenute da A . Per generare tale riassunto vengono di solito usate delle funzioni dette funzioni di *hashing*. Tali funzioni verranno trattate dettagliatamente nel seguito. Basti sapere che una funzione hash permette di ottenere da una sequenza di messaggi un ulteriore messaggio (tipicamente un numero) che la caratterizzi univocamente. Supponiamo che A e B si scambino i messaggi m_1, m_2, \dots, m_n durante una comunicazione. Per confermare tali messaggi, A e B si inviano vicendevolmente il riassunto dei messaggi scambiati. Nel caso in cui sia possibile autenticare i messaggi che contengono tali riassunti, e che A e B siano in grado di determinare se i messaggi ricevuti si riferiscono o meno alla sessione corrente, alla ricezione del riassunto inviato da B , A può confrontarlo con quello da lui generato. B similmente può confrontare quello che ha generato con quello ricevuto da A . Se i due riassunti coincidono, allora A e B hanno eseguito una sessione del protocollo scambiandosi effettivamente i messaggi m_1, m_2, \dots, m_n .

Injective agreement

Garantire questo grado di autenticazione può essere utile ad esempio nei protocolli ad uso *finanziario*. Supponiamo ad esempio che il fine di un protocollo \mathcal{P} sia quello di permettere ad un agente di spostare una somma di denaro dal proprio conto a quello di un'altro agente. Un agente A decide di trasferire una determinata somma c dal proprio conto a quello di un agente B . Se tale protocollo \mathcal{P} non offre garanzia di autenticità con accordo iniettivo, potrebbe allora accadere che la richiesta dell'agente A venga recepita dalla banca due o più volte, causando un accreditamento sul conto di B superiore alla somma che A aveva intenzione di versare. Viene spontaneo però porsi una domanda : come intendere e come garantire tale requisito su una rete che preveda la ritrasmissione dei messaggi? In genere la ritrasmissione

dei messaggi è prevista nell'architettura di rete ad un livello inferiore di quello nel quale si collocano i protocolli in questione. Di solito la ritrasmissione dei messaggi viene gestita nel livello di trasporto, mentre i protocolli di sicurezza si collocano per lo più a livello applicativo. Per questo motivo la gestione delle ritrasmissioni dovrebbe risultare trasparente al software che implementi tali protocolli. Tuttavia l'attenzione nei confronti delle problematiche di sicurezza si è sviluppata in un'epoca successiva alla realizzazione di molte delle infrastrutture che sono oggi alla base delle comunicazioni sulla rete internet, e dei protocolli che regolano tali comunicazione. Per questo motivo, anche se sarebbe lecito, non conviene presupporre nell'infrastruttura di rete che gli strati sottostanti quello al quale il protocollo che intendiamo esaminare si colloca siano *sicuri*. La ritrasmissione dei messaggi si rende necessaria per garantire la consegna dei messaggi stessi. A seconda degli obiettivi che il protocollo si prefige, questa proprietà potrebbe rivelarsi non necessaria. Tornando al protocollo preso in considerazione all'inizio di questo paragrafo, se la richiesta da parte di A di spostare una somma dal suo conto al conto di C dovesse non giungere a destinazione, A dovrebbe semplicemente eseguire una nuova sessione del protocollo, sperando che vada a buon fine. Sarebbe invece molto più grave se A vedesse sottratta dal suo conto una somma superiore a quella che intendeva trasferire.

2.1.5 Non repudiabilità

Un protocollo \mathcal{P} si dice fornire garanzie di non repudiabilità ad un agente A se fornisce ad A i mezzi per dimostrare che una azione è avvenuta.

Tipicamente, se A e B eseguono il protocollo \mathcal{P} , A deve essere in grado di dimostrarlo. Un protocollo che raggiunga un obiettivo di questo tipo può o no godere di un'altra proprietà : fairness. Si dice che \mathcal{P} gode di fairness se per ogni garanzia \mathcal{G} , che permette ad un agente A di dimostrare che B ha eseguito un'azione, viene fornita a B una garanzia *complementare*. Ad esempio supponiamo di esaminare un protocollo nel quale A invia un messaggio m a B . Una proprietà di non repudiabilità potrebbe essere *A riesce a dimostrare che B ha ricevuto il messaggio m* . La proprietà complementare è *B riesce a dimostrare che A ha spedito il messaggio m* . Lo studio delle garanzie di non repudiabilità differisce dallo studio degli altri requisiti di sicurezza. Un protocollo di sicurezza *classico* offre ad un agente un modo per verificare se una determinata azione sia o no avvenuta, basandosi su assunzioni che tale agente è in grado di verificare. Portare a termine protocollo che offra

garanzie di confidenzialità scambiando con un altro agente un messaggio m dà la certezza agli agenti coinvolti nella comunicazione che essi soli sono a conoscenza di tale messaggio. Similmente *autenticare* un messaggio come proveniente da B dà all'agente che riceve tale messaggio la certezza che B lo abbia generato. Un protocollo di non repudiabilità offre ad un agente gli strumenti per dimostrare che una azione è avvenuta. Le condizioni per verificare se un determinato evento sia o meno avvenuto devono quindi essere verificabili da qualunque agente, non solo da quelli coinvolti nella sessione del protocollo nella quale tale evento si è verificato. Risulta chiaro quindi come lo studio di proprietà di questo genere apra la strada a problematiche del tutto nuove.

2.2 Assunzioni e limiti teorici

Durante l'indagine sulla sicurezza di un protocollo ci si scontra spesso con la possibilità del verificarsi di eventi teoricamente possibili, ma nella pratica poco probabili. Esaminiamo ad esempio un terminale remoto, che autentichi un utente mediante un nome-utente ed una password. Esiste teoricamente una possibilità che un attaccante riesca a spacciarsi per un altro utente *indovinando* la password. Possiamo però ridurre questa possibilità a nostro piacimento aumentando le dimensioni della password, senza però mai raggiungere la certezza che questo evento non accada. Per fronteggiare simili difficoltà nella formalizzazione di un protocollo, vengono aggiunte al modello nel quale questa formalizzazione avviene delle assunzioni giudicate *ragionevoli*. Nel seguito verranno illustrate le assunzioni di questo tipo che vengono di solito usate nello studio dei protocolli di sicurezza.

2.3 Strumenti a disposizione degli agenti

Oggetto dei protocolli di sicurezza, come di tutti i protocolli di comunicazione, sono i messaggi. Tra i prerequisiti necessari all'esecuzione di un protocollo di sicurezza sono spesso previste, da parte degli agenti, capacità(abilità) specifiche nella manipolazione di tali messaggi, o di parti di questi. In genere si richiede che gli agenti siano in grado di utilizzare funzioni di cifratura e decifrazione(simmetriche o asimmetriche), funzioni hash e firma digitale.

2.3.1 Crittografia

Esistono diversi modi per garantire la confidenzialità di un messaggio. Uno di questi è l'utilizzo di un **crittosistema**. Un crittosistema è costituito da

- Un insieme di coppie di **chiavi** $\mathcal{K} = \{(k, k^{-1})\}$
- una funzione di **cifatura** $e(k, m) = m'$
- una funzione di **decifrazione** $d(k^{-1}, m') = m$

Si suppone che le funzioni e , d siano note a tutti gli utilizzatori del crittosistema. Un agente A che voglia inviare un messaggio m in maniera confidenziale ad un altro agente B deve inviare il messaggio cifrato con chiave k a tale agente e comunicargli la chiave di decifrazione per leggere il contenuto del messaggio.

Se per ogni coppia (k, k^{-1}) accade che $k \equiv k^{-1}$ si parla di crittografia **simmetrica**. In caso contrario si parla di crittografia **asimmetrica**.

Assunzioni generali sulle funzioni crittografiche

In genere una funzione crittografica viene definita perfetta se l'unico modo di forzarla sia un attacco esaustivo. Nello studio dei protocolli di sicurezza si assume invece che la funzione utilizzata sia **inviolabile**. Nell'ipotesi di crittografia perfetta, questa nuova assunzione è da considerarsi realistica dopo aver fatto alcune considerazioni. Innanzitutto il termine *sicurezza* di un sistema va sempre riferito al genere di utenti ed organizzazioni dalle quali ci si vuole difendere. Questo ci fa capire che il termine inviolabile è da intendersi come *inviolabile da una determinata classe di utenze*. Conoscere le potenzialità dei nostri attaccanti ci consente di aumentare le dimensioni delle chiavi da usare nella nostra funzione crittografica in modo da rendere altamente improbabile che questi attaccanti riescano ad indovinarla.

Una seconda considerazione è la seguente : supposto che un attaccante tenti di decrittare un messaggio m provando un insieme casuale di chiavi. L'utente non conosce il contenuto del messaggio criptato. Come può allora discernere se il messaggio ottenuto decrittando m con una chiave k sia effettivamente il contenuto del messaggio crittato? Se il contenuto del messaggio fosse un numero sul quale l'attaccante non ha alcuna indicazione, egli non sarebbe in grado di capire se la chiave utilizzata sia quella corretta oppure no.

Segreti a lungo e breve termine

Un approccio sviluppato successivamente prevede come ipotesi che un messaggio inviato criptato sulla rete resti confidenziale per un breve periodo di tempo. Questa ipotesi è di certo più realistica dell'ipotesi vista in precedenza. Una informazione la cui segretezza è necessaria solo per un breve periodo di tempo, tipicamente per la durata di una sessione, è chiamata segreto a **breve termine**. Una informazione della quale viene richiesta la segretezza senza limitazioni temporali viene chiamata segreto a **lungo termine**. Un esempio di protocollo che utilizza segreti a breve e lungo termine è il protocollo KerberosIV. L'esperienza sembrerebbe indicare che è impossibile realizzare protocolli di sicurezza che non prevedano segreti a lungo termine. Tuttavia è opportuno limitare al minimo la presenza di tali segreti nelle comunicazioni.

2.3.2 Funzioni hash

Una funzione hash è una generica funzione che va dall'insieme dei messaggi a questo stesso insieme. Nella pratica le funzioni hash vengono utilizzate per realizzare sunti dei messaggi. Si può richiedere o meno che tali sunti abbiano dimensione fissa. La funzione di tali sunti è paragonabile alla funzione svolta dalle check-sum nei protocolli di trasporto. Accompagnando un messaggio col proprio sunto possiamo verificare se il messaggio è stato o meno compromesso lungo la strada.

Assunzioni generali sulle funzioni hash

Si presuppone che le funzioni hash utilizzate siano pseudo-casuali. Una funzione si definisce pseudo casuale se, presa una sequenza di numeri casuali, la sequenza ottenuta applicando la funzione a questi numeri abbia, per un osservatore esterno, l'aspetto di una sequenza di numeri casuali. In particolare che nessuna indicazione sugli elementi a partire dai quali un hash è stato generato deve poter essere dedotta dallo hash stesso.

Contemporaneamente si presuppone che tali funzioni siano **iniettive**. Questa ipotesi è necessaria per presupporre che se un agente è in grado di generare un hash, allora è a conoscenza di tutti gli elementi necessari per generarlo. Utilizzare funzioni iniettive implica che tali funzioni producano un risultato di dimensione uguale all'argomento della funzione. Poiché una funzione di hash ha tra i suoi scopi originari quello di produrre un sunto dell'argomento, nella realtà le tali funzioni non possono essere iniettive.

Partendo dalle stesse considerazioni fatte per le funzioni crittografiche, possiamo affermare che presupporre una funzione hash iniettiva nella pratica si traduce nell'utilizzare una funzione che abbia un codominio abbastanza grande da rendere la probabilità di collisioni trascurabile.

2.3.3 Firma digitale

Dall'unione della crittografia asimmetrica e delle funzioni hash nascono i protocolli di firma digitale. Un protocollo di tal genere permette ad un agente che riceva un messaggio *firmato* da un'altro agente A, di affermare che tale messaggio sia stato generato proprio da A.

Un protocollo di firma digitale presuppone l'esistenza di una *Infrastruttura di chiavi pubbliche*. In questa infrastruttura ogni agente è caratterizzato univocamente da una coppia di chiavi : una chiave di cifratura **privata**, ed una chiave di decifrazione **pubblica**. L'associazione agente - coppia di chiavi viene mantenuta da enti autorizzati denominati **autorità di certificazione**.

Un agente A che intenda inviare un messaggio m firmato invia il messaggio m accompagnato dal suo sunto. Tutto questo cifrato con la sua chiave privata. In questo modo chiunque può decifrare il messaggio usando la chiave pubblica di A, verificare l'integrità del messaggio ed il mittente confrontando il messaggio ottenuto con il sunto.

2.3.4 Generazione di elementi *freschi*

Molto spesso viene richiesto ad un agente di generare elementi che non siano mai comparsi prima sulla rete. Tale ipotesi, in questa forma, non rappresenta di certo una ipotesi realistica. Essa, tuttavia, può essere intesa in due modi. Innanzitutto richiedere che si utilizzi un elemento fresco potrebbe significare supporre che questo elemento non sia stato utilizzato in un determinato lasso di tempo precedente all'istante corrente, ad esempio il tempo di vita massimo stimato per i messaggi della rete.

Tuttavia l'ipotesi di freschezza molto spesso viene intesa nel senso di casualità. Ossia quando viene richiesto un elemento fresco, in pratica viene richiesto un elemento tale da non poter essere *indovinato* da chi lo conosce. Per raggiungere questo scopo, come più volte ribadito, è sufficiente aumentare quanto basta le dimensioni dell'elemento preso in esame. È inoltre necessario che la generazione di tale elemento avvenga in maniera **casuale**, in modo da non dare ad un attaccante nessuna informazione aggiuntiva sull'elemento stesso.

2.4 Protocolli di seconda generazione

L'architettura di una rete è in genere strutturata a livelli. Un livello fisico e vari livelli software regolati da altrettanti protocolli. La specifica di un protocollo deve contenere indicazioni sul livello al quale tale protocollo si colloca e sui requisiti che devono soddisfare i protocolli del livello sottostante. Chiaramente tali requisiti devono essere realistici. Ancora meglio se nella specifica vengono indicati protocolli esistenti che li soddisfano. Tipicamente un protocollo di sicurezza viene concepito per essere implementato sulla rete Internet, e si colloca subito sopra il livello di trasporto. Come tale, presuppone un protocollo di livello di trasporto che garantisce un controllo sulla integrità dei messaggi, ma non dà garanzie sul mittente dei messaggi. Il progresso nello studio e nella formalizzazione di protocolli di sicurezza induce la nascita di quelli che definiamo protocolli di *seconda generazione*[21]. Questa classe include tutti i protocolli che prevedano al loro interno l'esecuzione di altri protocolli di sicurezza, specificando i requisiti che questi ultimi devono soddisfare, ma non necessariamente i dettagli del loro funzionamento.

Capitolo 3

Metodi formali per la verifica di protocolli di sicurezza : l'approccio induttivo

L'attacco mostrato durante l'analisi del protocollo Needham-Schroeder mostra come una dimostrazione informale possa facilmente essere ingannevole. Per questo si è resa necessaria la creazione di modelli formali che permettessero uno studio *rigoroso* dei protocolli di sicurezza e delle loro proprietà. La bontà di un modello formale di questo tipo è commisurata al grado di vicinanza che tale modello offre rispetto alla realtà. Il modello formale ideale non permette di verificare ipotesi che nella realtà possano essere contraddette. Tuttavia una formalizzazione non può essere troppo astrusa. Questo renderebbe complicata e quindi passibile di errori la formalizzazione dei protocolli in tale modello. Renderebbe difficilmente comprensibili i risultati ottenuti dall'analisi di un protocollo e troppo lunghi i tempi richiesti per effettuare tali analisi.

Come è facile immaginare, realizzare un modello che soddissi contemporaneamente questi due requisiti non è certo facile. Soprattutto alla luce del fatto che solo l'esperienza è in grado di dirci se un modello sia sufficientemente aderente alla realtà. Nel seguito verranno presentati alcuni modelli per la formalizzazione e lo studio dei protocolli di sicurezza.

3.1 Belief logic

Questo modello rappresenta il primo tentativo di formalizzazione per i protocolli di sicurezza. Esso concentra la sua attenzione sulle affermazioni alle quali gli agenti sono portati a credere, e su quelle cui sono *autorizzati a credere*. I vari passi di un protocollo vengono formalizzati con delle formule logiche, e vengono forniti una serie di postulati. Tutte le formule che possono essere derivate da quelle che rappresentano i passi del protocollo attraverso tali postulati rappresentano le proprietà del protocollo.

Il grande pregio di questo approccio è la chiarezza. Tuttavia proprio l'attacco scoperto da Lowe sul protocollo Needham-Schroeder ha mostrato la debolezza di questo modello. In questo modello tale protocollo risultava raggiungere in pieno il requisito di autenticazione, mentre Lowe ha dimostrato così non è.

3.2 CSP

Questo modello di calcolo[18] viene utilizzato per lo studio dei sistemi automatici. Grazie alle nozioni di *processo* e di *canale*, si presta notevolmente alla rappresentazione e all'analisi di protocolli di comunicazione. In particolare l'analisi di un protocollo di sicurezza avviene nel seguente modo :

1. si studia il sistema costituito da una serie di processi(gli agenti) che si scambiano informazioni seguendo il protocollo che si vuole esaminare, e si studia quali sono tutti i possibili stati nei quali il sistema può venirsi a trovare.
2. si ripete questa analisi introducendo un nuovo processo che si comporta in maniera arbitraria. Tale processo rappresenta un ipotetico attaccante.
3. Se gli stati raggiungibili dal sistema con l'aggiunta di questo nuovo processo sono gli stessi(e solo quelli) raggiungibili dal sistema senza tale processo, allora il protocollo non è passibile di attacchi.

Anche per questo modello di calcolo la chiarezza rappresenta un punto di forza. Tuttavia calcolare tutti i possibili stati nei quali un sistema possa venirsi a trovare potrebbe risultare complicato. Per questo modello però sono stati studiati dei software per automatizzare questo tipo di dimostrazioni.

È necessario però fare alcune considerazioni. Innanzitutto il fatto che un protocollo non sia passibile di attacchi non necessariamente implica che esso raggiunga i suoi obiettivi. Se ad esempio un protocollo che abbia come obiettivo di mantenere confidenziale un messaggio tra due agenti, prevede che uno di questi agenti invii tale messaggio ad un'altro agente, allora l'obiettivo di confidenzialità viene violato senza bisogno dell'intervento di nessun agente malizioso. In secondo luogo, per verificare la sicurezza di un protocollo usando questo modello è necessario che gli stati che il sistema può assumere durante l'esecuzione di tale protocollo siano finiti. Se fossero infiniti essi non potrebbero essere confrontati con quelli raggiunti dal sistema in presenza dell'attaccante. Supponiamo ad esempio che un protocollo permetta a due agenti di scambiarsi un numero arbitrario di messaggi, potenzialmente infinito. L'insieme degli stati nei quali il sistema possa venirsi a trovare è di conseguenza infinito. Un protocollo di questo genere non può essere quindi verificato usando questo modello.

3.3 L'approccio induttivo

Il modello formale che verrà successivamente utilizzato in questo documento per la formalizzazione e la verifica dei protocolli di sicurezza è denominato *approccio induttivo*[14] ed è dovuto a Lawrence Paulson. Tale modello fornisce una chiarezza nell'esposizione dei protocolli e dei risultati derivante direttamente dalla notazione e dagli strumenti matematici che vengono messi a disposizione. Tale approccio è infatti sviluppato sulla base della *High Order Logic*, utilizzando la formalizzazione di questa per il dimostratore meccanizzato Isabelle/Isar[16]. La sintassi HOL[15] è di certo vicinissima alla sintassi comunemente usata per la matematica. La novità di questo metodo, rispetto a quelli sperimentati in precedenza, consiste nel fatto che viene formalizzato 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.

3.4 Gli agenti

Questo modello suppone che l'insieme degli agenti che possono eseguire il protocollo sia infinito. L'insieme degli agenti è costituito da

- Un **Server**, che si presuppone onesto. Agisce di solito da tramite e da garante nello svolgimento del protocollo.
- Un insieme infinito di **agenti** in grado di eseguire il protocollo.
- Una **Spia**, dalla quale dobbiamo difenderci. A differenza degli altri agenti la spia ha il potere di intercettare il traffico che non le è diretto, e di sintetizzare ed inviare messaggi non previsti dal protocollo.

datatype

```
agent = Server | Friend nat | Spy
```

3.5 Chiavi

Si presuppone che gli agenti siano in grado di utilizzare un qualche algoritmo crittografico considerato *sicuro*. Per questo motivo viene introdotto un insieme infinito e numerabile di coppie di chiavi. In genere, per decifrare una informazione cifrata usando una chiave k è necessario conoscere l'*inversa* di tale chiave. La funzione **invKey** lega ogni chiave alla sua inversa. Tale funzione gode della proprietà riflessiva.

consts

```
invKey :: "key=>key"
```

axioms

```
invKey [simp] : "invKey (invKey K) = K"
```

Un sotto-insieme dell'insieme delle chiavi è quello delle **chiavi simmetriche**. Una chiave k è detta simmetrica se è uguale alla sua inversa.

constdefs

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

3.6 Messaggi

I messaggi vengono costruiti a partire da elementi atomici, combinati mediante alcuni operatori

3.6.1 Messaggi atomici

Vengono considerate le seguenti classi di messaggi atomici.

- Sia i un numero naturale, allora $\text{Number } i$ è un messaggio. Un *numero* viene in genere utilizzato per rappresentare un'informazione che non si vuole tenere nascosta, o che un attaccante è in grado di indovinare facilmente.
- Sia i un numero naturale, allora $\text{Nonce } i$ è un messaggio. A differenza dei numeri, le *Nonces* sono dei messaggi che non possono essere facilmente indovinati. Tipicamente numeri casuali molto grandi.
- Sia a appartenente all'insieme degli agenti, allora $\text{Agent } a$ è un messaggio. Tale messaggio rappresenta il *nome* col quale viene indicato l'agente specificato.
- Sia k appartenente all'insieme delle chiavi. Allora $\text{Key } k$ è un messaggio.

In realtà l'insieme degli agenti e l'insieme delle chiavi possono essere messi entrambi in corrispondenza biunivoca con l'insieme dei naturali. Tuttavia sono semanticamente differenti. Creare due insiemi ad hoc per rappresentare agenti e chiavi permette a sua volta di suddividere tali insiemi in altre sottoclassi. Un esempio classico è la suddivisione dell'insieme delle chiavi negli insiemi delle chiavi simmetriche e asimmetriche.

3.6.2 Operatori sui messaggi

- Siano $M1, M2$ due messaggi. Allora $\text{MPair}(M1, M2)$ è ancora un messaggio. Tale operatore rappresenta l'operatore di concatenazione, e viene di solito indicato con la scrittura $\{|M1, M2|\}$.
- Sia M un messaggio e k appartenente all'insieme delle chiavi. Allora $\text{Crypt } k M$ è ancora un messaggio. L'operatore Crypt rappresenta l'applicazione di una funzione di cifratura al messaggio M .

- Sia M un messaggio. Allora $\text{Hash } M$ è ancora un messaggio. Tale operatore, come indicato chiaramente dal nome, rappresenta l'applicazione di una funzione hash al messaggio M .

3.7 Eventi

Gli eventi *atomici* che possono verificarsi sulla rete vengono formalizzati nel seguente modo :

datatype

```

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

```

$\text{Says } A B X$ indica l'invio di un messaggio X da parte dell'agente A all'agente B .

$\text{Gets } B X$ indica la ricezione del messaggio X da parte dell'agente B .

$\text{Notes } A Y$ indica il fatto che il messaggio Y viene conservato dall'agente A , in modo da poter essere riutilizzato in seguito.

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

3.8 Tracce

Una sequenza di eventi viene definita **traccia**. Ogni traccia rappresenta una possibile *storia della rete*. Oggetto dell'indagine saranno proprio tali sequenze. Per affermare la validità di una proprietà, si dimostrerà che tale proprietà è verificata in ogni possibile traccia.

3.9 Conoscenza degli agenti : l'operatore *knows*

Gran parte delle proprietà che i protocolli di sicurezza intendono garantire riguardano la conoscenza degli agenti. Ad esempio la proprietà di confidenzialità implica che gli agenti coinvolti conoscano il messaggio, e nessun altro. Per indagare sulla conoscenza degli agenti si usa l'operatore *knows* . Tale operatore prende come argomenti un agente ed una traccia, e restituisce un

insieme di messaggi. Tali messaggi vengono ricavati sia da quelli che compaiono sulla traccia, sia da quelli contenuti nello **stato iniziale** dell'agente. Lo stato iniziale degli agenti dipende dai prerequisiti richiesti dal protocollo. Ad esempio, se viene presupposta una PKI, allora ovviamente la chiave privata dell'agente stesso e le chiavi pubbliche di tutti gli agenti saranno comprese in questo insieme.

primrec

```

initState_Server: "initState Server =
  insert (Key (priEK Server))(
  insert (Key (priSK Server))
  ((Key ' range pubEK) ∪ (Key ' range pubSK)
  ∪ (Key ' range shrK)))"

initState_Friend: "initState (Friend i) =
  insert (Key (priEK (Friend i)))(
  insert (Key (priSK (Friend i)))(
  insert (Key (shrK (Friend i)))
  ((Key ' range pubEK) ∪ (Key ' range pubSK))))"

initState_Spy: "initState Spy =
  insert (Key (priEK Spy))(
  insert (Key (priSK Spy))(
  insert (Key (shrK Spy))
  ((Key ' range pubEK) ∪ (Key ' range pubSK))))"

```

I messaggi che un agente apprende dalla traccia sono invece quelli che ha ricevuto, tramite una Gets, o quelli che ha egli stesso elaborato, tramite una Notes . Un caso particolare è la spia. La spia deve poter monitorare tutto il traffico. Per questo motivo entrano a far parte della conoscenza della spia **tutti** i messaggi inviati sulla rete tramite un evento Says .

consts

```

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

```

syntax

```

spies :: "event list => msg set"

```

translations

```

"spies" => "knows Spy"

```

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))"
```

3.10 Operatori su insiemi di messaggi

Utili strumenti per la formalizzazione delle proprietà di un protocollo sono gli operatori discussi nel seguito.

3.10.1 Parts

Dato un messaggio m ed un insieme di messaggi \mathcal{M} , si dice che m appartiene a $\text{parts } \mathcal{M}$ se m compare in \mathcal{M} come messaggio o come parte di messaggio. Formalmente

- $m \in \mathcal{M} \longrightarrow m \in \text{parts}(\mathcal{M})$
- $\text{Crypt } k \ m \in \text{parts}(\mathcal{M}) \longrightarrow m \in \text{parts}(\mathcal{M})$
- $\{m1, m2\} \in \text{parts}(\mathcal{M}) \longrightarrow m1, m2 \in \text{parts}(\mathcal{M})$

Tale operatore viene utilizzato per indagare la presenza o meno di alcuni elementi sulla traccia. Lemmi che trattino proprietà di questo tipo vengono definiti lemmi di **regolarità**.

3.10.2 Analz

Dato un messaggio m ed un insieme di messaggi \mathcal{M} , si dice che m appartiene a $\text{analz}(\mathcal{M})$ nel caso in cui m compaia in \mathcal{M} come messaggio o come parte di messaggio e, se m compare come crittotesto, allora l'insieme \mathcal{M} fornisce anche la chiave per decifrarlo. Formalmente

- $m \in \mathcal{M} \longrightarrow m \in \text{analz}(\mathcal{M})$
- $\text{Crypt } k \ m \in \text{analz}(\mathcal{M}), \text{Key } k \in \text{analz}(\mathcal{M}) \longrightarrow m \in \text{analz}(\mathcal{M})$
- $\{m1, m2\} \in \text{analz}(\mathcal{M}) \longrightarrow m1, m2 \in \text{analz}(\mathcal{M})$

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 a questo, o che e' stato in grado di intercettare nel caso della spia, ma cosa egli è in grado di dedurre da tali messaggi.

3.10.3 Synth

L'insieme $\text{synth } \mathcal{M}$ è costituito da tutti i messaggi che è possibile costruire a partire da quelli contenuti in \mathcal{M} .

- $m \in \mathcal{M} \longrightarrow m \in \text{synth}(\mathcal{M})$
- $m \in \text{synth}(\mathcal{M}), \text{Key } k \in \text{synth}(\mathcal{M}) \longrightarrow \text{Crypt } k \ m \in \text{synth}(\mathcal{M})$
- $m \in \text{synth}(\mathcal{M}) \longrightarrow \text{Hash}(m) \in \text{synth}(\mathcal{M})$
- $m1, m2 \in \text{synth}(\mathcal{M}) \longrightarrow \{m1, m2\} \in \text{synth}(\mathcal{M})$

Tale operatore viene di solito utilizzato per indagare sulle *potenzialità* di un agente. L'insieme $\text{synth}(\text{analz}(\text{knows } A \ \text{evs}))$ 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 da una 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 $h = \text{Hash}\{X, Y\}$. Se vale la proprietà

$$h \in \text{analz}(\text{knows } A \ \text{evs}) \longrightarrow \text{analz}(\text{knows } A \ \text{evs})$$

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

3.11 Formalizzazione del protocollo

Un protocollo viene formalizzato utilizzando una serie di regole induttive che descrivano la struttura delle tracce. Il caso base è rappresentato dalla traccia vuota. Ogni passo del protocollo viene formalizzato da una regole induttiva. Supponiamo di analizzare un protocollo \mathcal{P} che contenga, al passo ennesimo, l'invio di un messaggio Y condizionato dalla ricezione del messaggio X . Troveremo, tra le regole induttive, la regola

Sia evs una traccia per il protocollo \mathcal{P} . Se $\text{Gets } A X$ appartiene alla traccia evs , allora $\text{Says } A B Y$ appeso ad evs e' ancora una traccia per il protocollo \mathcal{P} .

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, come stai?.
- L'agente B , alla ricezione del messaggio, puo' rispondere Bene oppure male.

Nel modello induttivo, la formalizzazione di tale protocollo sarebbe la seguente :

Sia \mathcal{T} l'insieme di tutte le possibili tracce del protocollo in questione

- $[] \in \mathcal{T}$
- $trace \in \mathcal{T} \longrightarrow \text{Says } A B "Ciao, sono A, comestai?" \# trace \in \mathcal{T}$
- $trace \in \mathcal{T} \wedge \text{Gets } B "Ciao, sono A, comestai?" \in trace \longrightarrow \text{Says } B A "Bene" \# trace \in \mathcal{T}$
- $trace \in \mathcal{T} \wedge \text{Gets } B "Ciao, sono A, comestai?" \in trace \longrightarrow \text{Says } B A "Male" \# trace \in \mathcal{T}$

Secondo questa definizione sia la traccia nella quale B risponde Bene, sia quella nella quale risponde Male, sono tracce per il protocollo \mathcal{P} . Prendendo in considerazione l'insieme \mathcal{T} 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 ...). Le

diciture utilizzate per gli agenti, A e B , sono generiche e possono rappresentare una qualsiasi coppia di agenti, se non un'unico agente. Per questo motivo l'insieme \mathcal{T} contiene anche delle tracce nelle quali vengono aperte piu' sessioni del protocollo, contemporanee o disgiunte.

Oltre alle regole che descrivono i passi del protocollo, devono essere inserite altre regole *standard*, per descrivere il funzionamento della rete sottostante. La più importante di queste è la regola **reception**.

$$[| \text{ evsr } \in \mathcal{P}; \quad \text{ Says } A \ B \ X \in \text{ set evsr } \ |] \implies \text{ Gets } B \ X \ \# \ \text{ evsr } \in \mathcal{P}$$

Tale regola formalizza una situazione nella quale, se un messaggio è stato spedito, esso può o meno giungere a destinazione. Tuttavia esistono delle tracce in cui questo non accade. Tale regola 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. In oltre l'esperienza insegna che molti attacchi sfruttano proprio questo fatto.

Altre regole *ausiliarie* riguardano le potenzialità aggiuntive della spia. Il modello induttivo assicura di per sè alla spia la possibilità di conoscere messaggi inviati su canali convenzionali, anche se non sono indirizzati a questa. Tale modello invece non offre alla spia la possibilità di inviare messaggi *arbitrari* (non previsti nel protocollo). A tale scopo viene introdotta la regola induttiva denominata **Fake** :

$$[| \text{ evs } \in \mathcal{P} \ \dots; \ X \in \text{ synth}(\text{analz}(\text{knows } \text{Spy } \text{evsfssl})) \ |] \\ \implies \text{ Says } A \ B \ X \ \# \ \text{ evs } \in \mathcal{P}$$

Tale regola permette alla spia di *forzare* il protocollo, ove questo sia possibile. Essa inoltre premette alla spia di interpersi 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 X a B . B in generale non è in grado di decidere se il messaggio che riceve proviene da A o dalla spia.

Infine la regola **Oops** viene inserita nei protocolli che intendono fa uso di segreti a breve termine. Essa formalizza il fatto che un messaggio criptato possa essere forzato dopo un periodo di tempo sufficientemente lungo.

Capitolo 4

Certified mail protocol

In questo capitolo verrà descritto il protocollo di posta elettronica certificata[2]. Tale protocollo è di particolare interesse in questa trattazione non solo per gli obiettivi che permette di raggiungere, ma anche perchè può essere considerato un protocollo di seconda generazione, nel senso del capitolo due.

4.1 Prerequisiti

Gli attori coinvolti nello svolgimento del protocollo sono i seguenti

1. Un mittente S che intende spedire un messaggio ottenendo una ricevuta di avvenuta consegna.
2. Il destinatario R del messaggio.
3. Un server *sicuro* TTP (il postino).

Gli autori del protocollo presuppongono che il mittente comunichi con TTP su un canale a consegna garantita, mentre il ricevente comunichi con TTP su un canale a consegna garantita e confidenziale. Per canale a consegna garantita si intende un canale lungo il quale ogni messaggio spedito viene necessariamente recapitato. Nel seguito verranno illustrati alcuni protocolli di livello di trasporto che rendono questa ipotesi *realistica*. In generale la possibilità di garantire la consegna di un messaggio è strettamente correlata all'architettura della rete sulla quale tale canale viene stabilito ed al tempo massimo dopo il quale un messaggio viene considerato perso. Nel caso di protocolli di posta elettronica, tale tempo massimo è di quindici giorni. È

di certo ragionevole pensare di poter garantire che ogni guasto alla rete che impedisca il recapito di un messaggio possa venire riparato entro tale tempo. L'architettura di internet, dove ogni messaggio per così dire trova la strada da sé, permette in genere di garantire la consegna dei messaggi anche nel caso di guasti permanenti alla rete, sempre presupponendo di utilizzare un protocollo specifico. Per questo motivo, su internet si può pretendere che il recapito di un messaggio avvenga in tempi molto inferiori ai quindici giorni. Ragionevolmente qualche ora dovrebbe essere sufficiente per avere la certezza che un messaggio sia stato recapitato.

Ogni agente condivide col server un *segreto a lungo termine*, tipicamente una password, che nel seguito idicheremo con **RPwd A**, ove *A* è l'agente proprietario di tale password.

Inoltre il server dispone di due coppie di chiavi asimmetriche:

1. **TTPDecKey** una chiave di decifrazione nota solo al server.
2. **TTPEncKey** la corrispondente chiave di cifratura, pubblica.
3. **TTPSigKey** una chiave privata utilizzata per firmare i documenti.
4. **TTPVerKey** la corrispondente chiave pubblica che permette a chiunque di verificare la firma impressa con **TTPSigKey**.

4.2 Obiettivi del protocollo

L'obiettivo fondamentale del protocollo è di permettere ad un agente *S* di spedire un messaggio ad un agente *R*, garantendo ad *S* di ottenere una ricevuta di avvenuta consegna se e solo se *R* abbia letto il messaggio.

Si può osservare che, per ottenere la garanzia che un messaggio arrivi a destinazione, sarebbe sufficiente utilizzare un canale che offra tale garanzia. E, visto che l'esistenza di un simile canale viene supposta dagli autori solo per le comunicazioni con *TTP*, non si potrebbe estendere l'uso di questo canale alle comunicazioni tra gli altri agenti? In pratica, se gli agenti comunicano con *TTP* usando un protocollo che garantisce la consegna dei messaggi, nulla vieta che tale protocollo possa essere utilizzato per le comunicazioni tra agenti.

In realtà l'obiettivo che il protocollo intende raggiungere è più forte di una semplice garanzia sulla consegna del messaggio. L'esecuzione del protocollo fornisce all'agente *S*, grazie al certificato di avvenuta consegna, la possibilità di *dimostrare* che *R* ha effettivamente ricevuto il messaggio. Di conseguenza, esattamente come avviene per le raccomandate con ricevuta di

ritorno nell'ambito del sistema postale, tale certificato può essere utilizzato anche in sede legale.

Inoltre il fatto che S possieda un suddetto certificato implica non solo che R abbia ricevuto il messaggio, ma che abbia inoltre posto la sua attenzione su di esso, manifestando la volontà di leggerlo. Il fatto che un messaggio raggiunga la casella di posta elettronica di una persona infatti non implica necessariamente che quella persona si renda conto di aver ricevuto tale messaggio. Egli potrebbe essere momentaneamente impossibilitato a controllare la casella stessa per i più svariati motivi.

4.2.1 Obbiettivi secondari

Un altro elemento che rende questo protocollo particolarmente innovativo è il fatto che, nonostante TTP partecipi alla comunicazione, esso non viene mai a conoscenza del messaggio che i due agenti intendono scambiarsi. Questa proprietà garantisce gli utenti sulla totale riservatezza delle loro comunicazioni.

Ogni messaggio viene accompagnato da una breve descrizione del contenuto. In base a questa un agente decide se leggere o meno il messaggio. Tale descrizione viene riportata nel certificato di avvenuta consegna. In tal modo un agente non può essere *accusato* di aver letto un messaggio, se la descrizione che accompagna tale messaggio non rispecchia il contenuto del messaggio stesso.

Infine il protocollo risulta estremamente leggero, sia per TTP , ma soprattutto per gli agenti, e di semplice implementazione essendo la computazione ridotta al minimo.

4.3 Livelli di autenticazione

Il protocollo fornisce quattro livelli di autenticazione, riservando al mittente la scelta di quale utilizzare. Ogni livello di autenticazione fornisce differenti garanzie agli esecutori del protocollo, ma richiede anche differenti capacità agli stessi.

NoAuth Utilizzando questa modalità non avviene alcuna autenticazione di R . L'unica garanzia offerta al mittente dal protocollo con questa modalità di autenticazione è che, se egli non riceve il certificato, nessuno è venuto a conoscenza del messaggio spedito.

SAuth S autentica R con un meccanismo di domanda e risposta. In pratica S ed R concordano prima della comunicazione una funzione **ack** da

utilizzare, che deve restare confidenziale. Durante la comunicazione S invia una *domanda* q alla quale R risponde con $ack(q)$. In tal modo S è in grado di autenticare R . Tale forma di autenticazione non può assumere alcuna valenza legale, poichè S conosce sia q che $ack(q)$.

TTPAuth L'autenticazione di R viene demandata a TTP , mediante il meccanismo della password. Questo livello di autenticazione è sufficiente affinché il protocollo raggiunga l'obbiettivo principale.

BothAuth Sia TTP che S autenticano R .

L'autenticazione di tipo **SAuth** non ha alcun interesse *giuridico*, come precedentemente rilevato. Tuttavia essa potrebbe essere utilizzata nel caso in cui TTP non sia in grado di autenticare R (ossia non siano disponibili gli altri livelli di autenticazione). Inoltre il meccanismo di autenticazione domanda-risposta risulta poco chiaro e di difficile implementazione.

Per contro, una doppia autenticazione di R (**BothAuth**) appare quanto meno ridondante. L'unico motivo che potrebbe spingere S a richiedere questo livello di autenticazione è una mancanza di fiducia nei confronti di TTP . Questo è assurdo perchè TTP è supposto affidabile per ipotesi. Inoltre, considerando TTP inaffidabile, l'autenticazione di R da parte di TTP diventerebbe inutile, suggerendo quindi la scelta del livello di autenticazione **SAuth**.

Quindi la scelta migliore per il livello di autenticazione sembra essere quella di TTPAuth.

4.4 Descrizione del protocollo

Iniziamo con una descrizione semplificata del protocollo, per approfondirne in seguito i dettagli. Supponiamo che l'agente S decida di inviare il messaggio m all'agente R . S allora genera in maniera casuale una chiave simmetrica k , e costruisce per TTP un *ticket*, $S2TTP$, che conterrà tale chiave e l'identità del destinatario. Tale ticket viene crittografato con la chiave $TTPDecKey$. In tal modo solo TTP è in grado di leggere il contenuto del messaggio. S allora spedisce ad R tale ticket ed il messaggio m criptato con la chiave k . R , per farsi rilasciare la chiave k da TTP , gli invia $S2TTP$. A questo punto TTP invia ad R la chiave, cosicchè possa leggere il messaggio, e ad S il ticket $S2TTP$ firmato con $TTPSigKey$, il certificato. Passiamo ora ad una descrizione più dettagliata del protocollo.

1.
 - S genera in maniera casuale una chiave k .
 - S costruisce $em = m_k$.
 - S genera una domanda q , alla quale R dovrà rispondere con $r = ack(q)$ secondo una funzione ack sulla quale R ed S si erano accordati precedentemente.
 - S genera $cleartext$, una etichetta che indica il contenuto del messaggio.
 - S costruisce un hash $h_s = Hash(cleartext, q, r, em)$.
 - S sceglie $authoption$ come livello di autenticazione desiderato.
 - S costruisce il ticket $S2TTP = \{S, authoption, k, R, h_s\}$.
 - S spedisce ad R il messaggio

$$\{TTP, em, authoption, cleartext, q, S2TTP\}$$

2. R riceve il messaggio 1
 - R genera $r = ack(q)$;
 - R genera $h_r = Hash(cleartext, q, r, em)$.
 - R spedisce a TTP , su un canale sicuro a consegna garantita, la richiesta per farsi rilasciare la chiave

$$\{S2TTP, RPwdR, h_r\}$$

3. TTP riceve tale richiesta da R
 - TTP controlla che h_s , contenuto nel ticket $S2TTP$, ed h_r siano uguali (R ed S concordano su $cleartext$ ed em).
 - TTP controlla che la password contenuta nel messaggio sia proprio quella dell'agente indicato nel ticket come destinatario. In oltre controlla se l'hash contenuto nel ticket corrisponde a quello inviato da R .
 - TTP rilascia la chiave ad R su un canale sicuro a consegna garantita

$$\{k, h_r\}$$
 - TTP invia ad S il certificato di avvenuta ricezione su un canale a consegna garantita

$$\{S2TTP\}_{TTPSigKey}$$

Se la modalità di autenticazione selezionata è **NOAuth** o **TTPAuth**, allora r ed q devono essere uguali ad una costante **null**. Se la modalità di autenticazione è **NOAuth** o **SAuth** semplicemente *TTP* non controlla la password ricevuta al passo 2.

Capitolo 5

Formalizzazione dei canali di comunicazione

Il termine *canale di comunicazione* è da intendersi, in questo documento, come il mezzo sul quale uno o più agenti effettuano una comunicazione che soddisfi alcuni requisiti. Una comunicazione attraverso un canale con certi requisiti si realizza mediante l'esecuzione di un protocollo che di sicurezza che soddisfi tali requisiti. Un *protocollo di seconda generazione* può allora essere inteso come un protocollo che utilizzi siffatti canali.

È importante sottolineare che per implementare nella pratica un dato canale può essere utilizzato un protocollo o una infrastruttura fisica che garantisca, oltre ai requisiti propri del canale, altri requisiti aggiuntivi. A tal proposito parte integrante dello studio di un protocollo diventa l'individuazione dei requisiti minimi che ogni canale previsto nel protocollo deve soddisfare affinché il protocollo stesso raggiunga i propri obiettivi.

Obiettivo di questo capitolo è proporre una serie di schemi inediti per la formalizzazione di alcuni tipi di canale usando il metodo induttivo con la sintassi Isabelle/Isar; schemi da istanziare poi nella formalizzazione di protocolli di seconda generazione.

5.1 Canali convenzionali

Sotto tale dicitura sono da intendersi i canali *nativi* del modello induttivo. Tali canali modellano la normale comunicazione sulla rete internet, e come tali non offrono garanzie di consegna. Essi ovviamente non offrono neppure garanzie assimilabili agli obiettivi classici dei protocolli di sicurezza.

5.1.1 Formalizzazione

Essendo questi canali già previsti nel modello induttivo, la loro formalizzazione ricalca esattamente quella illustrata nel capitolo precedente.

Se un protocollo \mathcal{P} al passo i -esimo prevede che l'agente A spedisca a B il messaggio X su un canale convenzionale, modelliamo tale passo con una regola induttiva nella forma

$$[| \text{ evs } \in \mathcal{P} \dots |] \implies \dots \# \text{ Says } A B X \# \dots \# \text{ evs } \in \mathcal{P}$$

Se un protocollo tra le ipotesi del passo i -esimo prevede che l'agente B riceva il messaggio X su un canale convenzionale, allora la regola induttiva che modella tale passo deve contenere $\text{Gets } B X$ tra le ipotesi.

$$[| \text{ evs } \in \mathcal{P} \dots \text{ Gets } B X \dots |] \implies \dots \# \text{ evs } \in \mathcal{P}$$

5.1.2 Come rendere questa formalizzazione davvero realistica

Rendere *realistica* la formalizzazione di un canale significa identificare i possibili utilizzi *impropri* di tale canale nella realtà, e renderli disponibili alla spia nel modello. L'utilizzo improprio da parte della spia di un canale convenzionale viene formalizzato dalla regola *Fake* illustrata in precedenza.

5.2 Canali di comunicazione con consegna garantita

A norma di definizione, un canale garantisce la consegna dei messaggi se ogni messaggio spedito su tale canale prima o poi raggiungerà la sua destinazione.

L'ipotesi di consegna garantita rientra tra le assunzioni ritenute teoricamente **impossibili**, ma realizzabili nella pratica. Supponiamo infatti di conoscere il tasso di perdita dei messaggi sulla rete che stiamo utilizzando. Sia $0 < T < 1$ tale tasso. Ogni messaggio inviato ha quindi una probabilità $(1 - T)$ di arrivare a destinazione. Se un messaggio viene inviato n volte, la probabilità di arrivare a destinazione sale a $(1 - T)^n$. Quindi, conoscendo il tasso di perdita dei messaggi sulla rete, possiamo prevedere un numero di ritrasmissioni tale da rendere realistica l'ipotesi di consegna garantita.

Supponiamo invece di utilizzare, per recapitare i messaggi, un protocollo del tipo

1. S spedisce ad R un messaggio m .
2. R , se e quando riceve m , spedisce ad S un messaggio di avvenuta ricezione.
3. Se, entro un tempo t fissato, S non riceve la conferma dell'avvenuta ricezione, il protocollo ricomincia.

Fissato un intervallo di tempo $T1$, consideriamo l'obbiettivo di consegna garantita raggiunto se R viene a conoscenza del messaggio m dopo un tempo $T1$ a partire dal primo tentativo di invio da parte di S . L'unico scenario nel quale protocollo fallisce è quello nel quale ad ogni trasmissione o si perde il messaggio, o si perde la conferma. In ogni caso il protocollo viene ripetuto al più $T1/t$ volte, prima di essere considerato fallito. Scegliendo quindi $T1$ infinitamente più grande di t , la probabilità che il protocollo fallisca tende a zero.

5.2.1 Formalizzazione

Formalizziamo tale requisito costringendo invio e ricezione in un unico passo induttivo.

Se un protocollo \mathcal{P} al passo i -esimo prevede che l'agente A spedisca a B il messaggio X su un canale che garantisce la consegna, modelliamo tale passo con una regola induttiva nella forma

$$[| \text{ evs } \in \mathcal{P} \dots |] \implies \dots \# \text{ Says } A B X \# \text{ Gets } B X \dots \# \text{ evs } \in \mathcal{P}$$

Se un protocollo tra le ipotesi del passo i -esimo prevede che l'agente B riceva il messaggio X su un canale che garantisce la consegna il, allora la regola induttiva che modella tale passo deve contenere $\text{Gets } B X$ tra le ipotesi, come per un canale convenzionale.

5.2.2 Perché questa formalizzazione non indebolisce il modello

Comprimere invio e ricezione di un messaggio in un singolo passo non diminuisce le potenzialità di intervento di un agente disonesto. Supponiamo infatti che un agente A decida di spedire un messaggio X ad un'altro

agente B su un canale a consegna garantita. La spia decida di intromettersi nella comunicazione spediendo sempre a B un ulteriore messaggio Y . Qualora B riceva anche questo messaggio Y , la situazione della traccia sarà allora la seguente :

...
...
Says A B X
Gets B X
...
Says Spy B Y
...
Gets B Y
...

In pratica l'agente B , non essendo a priori in grado di risalire al mittente messaggio, non è in grado di capire tra X ed Y quale sia il messaggio compromesso. Il risultato è che la spia è riuscita ad intromettersi in una comunicazione su un canale a consegna garantita, al pari che su un canale convenzionale.

5.3 Canale di comunicazione confidenziale con consegna garantita

Esaminiamo ora un canale che permette a due agenti di scambiarsi informazioni in maniera confidenziale, garantendo inoltre la consegna dei messaggi. Una possibile realizzazione di tale canale si ottiene utilizzando il protocollo SSL, che garantisce confidenzialità, su un protocollo che garantisce la consegna. Il protocollo SSL garantisce anche un requisito aggiuntivo: l'autenticazione dei messaggi. La disponibilità di un grande numero di implementazioni per SSL, nonché la sua provata affidabilità, ne motivano l'utilizzo in una eventuale implementazione.

5.3.1 Formalizzazione

L'evento che utilizziamo per modellare questo canale è **Notes**. Tale evento nasce inizialmente per permettere ad un agente di *annotarsi* qualche informazione, in maniera da poterla utilizzare in seguito. Tuttavia osserviamo che esso si presta, se usato con i dovuti accorgimenti, a modellare un canale confidenziale.

Innanzitutto ricordiamo un'importante proprietà di questo evento: se **Notes** $A X$ compare sulla traccia, da quel momento il messaggio X entra a far parte dello *stato interno* dell'agente A . Per cui, potremmo pensare di formalizzare un agente A che voglia mettere a conoscenza B di un certo messaggio X , appartenente al proprio stato interno, con una regola del tipo

$$[! \text{ evs} \in \mathcal{P} \dots!] \implies \dots \# \text{Notes } B X \# \dots \# \text{ evs} \in \mathcal{P}$$

Tuttavia è opportuno che un agente, nel momento nel quale invia un messaggio, venga a conoscenza di tale messaggio (se non lo era già prima).

Queste due considerazioni ci suggeriscono la regola di costruzione per modellare la trasmissione su un canale con le suddette proprietà.

Se un protocollo \mathcal{P} al passo i -esimo prevede che l'agente A spedisca a B il messaggio X su un canale che confidenziale con consegna garantita, modelliamo tale passo con una regola induttiva nella forma

$$[! \text{ evs} \in \mathcal{P} \dots!] \implies \dots \# \text{Notes } A X \# \text{Notes } B X \dots \# \text{ evs} \in \mathcal{P}$$

Se un protocollo tra le ipotesi del passo i -esimo prevede che l'agente B riceva il messaggio X su un canale confidenziale con consegna garantita, allora la regola induttiva che modella tale passo deve contenere tra le ipotesi **Notes** $B X$.

5.3.2 Come rendere questa formalizzazione davvero realistica

A differenza di quanto accade per i canali a consegna garantita non confidenziali la regola Fake classica non cattura la semantica di un utilizzo improprio da parte della spia di un canale confidenziale così definito. È opportuno per questo motivo introdurre una nuova regola, simile alla regola Fake classica

$$\begin{aligned} & [! \text{ evs} \in \mathcal{P} \dots; X \in \text{synth}(\text{analz}(\text{knows } \text{Spy } \text{evsfssl})) !] \\ \implies & \text{Notes } A X \# \text{Notes } B X \# \text{ evs} \in \mathcal{P} \end{aligned}$$

Capitolo 6

Formalizzazione del protocollo

Questo capitolo tratta la formalizzazione del protocollo usando la sintassi *Isabelle Isar*, secondo gli schemi dell'approccio induttivo. In particolare viene studiato il protocollo nella modalità di autenticazione **BothAuth**. Questo permette di analizzare la formalizzazione sia del meccanismo di autenticazione mediante password, sia del meccanismo di autenticazione mediante domanda-risposta.

6.1 Strumenti : teoria *Traceability*

Tale formalizzazione usa come base la teoria **Traceability**. Tale teoria è stata pensata come base per lo studio di protocolli di non repudiabilità. La novità sostanziale rispetto protocolli allo studio di protocolli di altro tipo è l'assenza dell'insieme **BAD**. In oltre vengono forniti una serie di elementi che possono risultare utili. In particolare vengono forniti un insieme di **chiavi condivise** tra il server e gli agenti, ed una infrastruttura di chiavi pubbliche e private ed un insieme di chiavi per la firma digitale.

6.2 Elementi base del protocollo

Innanzitutto viene inserita una traslazione all'unico scopo di aumentare la leggibilità e di rendere la notazione conforme a quella usata dall'autore del protocollo. L'introduzione di una traslazione non influenza in alcun modo la *computazione* delle dimostrazioni, ma rappresenta solo una *sostituzione* sintattica(macro).

```
"TTP" == "Server"
```

6.2.1 Password

Per formalizzare nel nostro contesto le password, esaminiamo la semantica di questi elementi : la password di un certo agente è un segreto condiviso tra TTP e l'agente stesso. L'insieme di chiavi condivise fornito dalla teoria Traceability soddisfa in pieno questa proprietà. Ricorriamo quindi ad una semplice traslazione :

```
"RPwd" == "shrK"
```

6.2.2 Chiavi del server

Il protocollo non necessita di una infrastruttura di chiavi pubbliche e non prevede, da parte degli agenti, alcuna capacità in merito alla firma digitale. Utilizziamo tuttavia questi due oggetti offerti da Traceability per formalizzare le chiavi di *TTP*:

```
"TTPDecKey" == "priEK Server"  
"TTPEncKey" == "pubEK Server"  
"TTPSigKey" == "priSK Server"  
"TTPVerKey" == "pubSK Server"
```

6.2.3 Meccanismo di autenticazione richiesta-risposta

A questo scopo viene introdotta una non meglio definita funzione *ack*, che permette ad un agente di generare la risposta per una qualunque domanda *q*. Questa formalizzazione non cattura di certo la semantica di una funzione condivisa tra una coppia di agenti, come era stata pensata dagli autori del protocollo. Tuttavia raggiunge ugualmente il suo scopo del protocollo. La

capacità di utilizzare questa funzione viene data infatti solo ad un agente che esegua *onestamente* il protocollo, e non viene aggiunta alle capacità di una spia che compia qualche azione illecita. In tal modo la presenza di $ack(q)$ come risposta garantisce sull'onestà del destinatario, quindi lo autentica (un agente *onesto* non si spaccia per un'altro).

```
ack :: "nat => nat"
```

6.2.4 Modalità di autenticazione

Infine, per completezza, vengono definite quattro costanti per le corrispondenti modalità di autenticazione, anche se verrà utilizzata solo quella corrispondente alla, modalità BothAuth.

```
NoAuth      :: nat
TPAuth      :: nat
SAuth       :: nat
BothAuth    :: nat
```

6.3 Formalizzazione del protocollo

In questo paragrafo viene presentata una formalizzazione del protocollo *Certified mail* nel modello induttivo.

6.3.1 Passo 1: l'agente S inizia il protocollo

```
CM1 : "
[| evs1 ∈ certified_mail;

  (* L'agente genera il messaggio da inviare *)
  Nonce m ∉ used evs1;

  (* L'agente genera quindi una chiave fresca *)
  Key k ∉ used evs1;
  k ∈ symKeys;

  (* Genera poi una domanda, tale domanda deve essere fresca, e diversa
  dal messaggio inviato *)
  Nonce q ∉ used evs1;
  m ≠ q;
```

```

r = ack q;

(* S Genera infine lo hash *)
hs = Hash {/ Number cleartext, Nonce q, Nonce r, Crypt k (Nonce m) /};

(* ed il ticket per TTP *)
S2TTP = Crypt TTPEncKey {/ Agent S, Number BothAuth, Key k, Agent R,
hs/}
[] =>
  Says S R {/ Agent TTP, Crypt k (Nonce m), Number BothAuth, Number cleartext,
Nonce q, S2TTP /}#evs1 ∈ certified_mail"

```

La semantica di un predicato del tipo $X \notin used\ evs$ è X non compare precedentemente sulla traccia. Nel caso della generazione della chiave k , della domanda q e del messaggio m tale assunzione è ovviamente troppo forte. Un agente, per generare una chiave in modo che tale chiave non sia mai comparsa nel traffico della rete, dovrebbe conoscere tutto il traffico da quando la rete è stata attivata. In realtà il significato col quale bisogna intendere tale ipotesi è il seguente : *la spia non è in grado di dedurre o indovinare X* . Questo risultato si ottiene nella pratica generando la chiave in maniera casuale ed usando una chiave di *grosse* dimensioni.

6.3.2 Passo 2: l'agente R richiede a TTP la chiave

```

[/ evs2 ∈ certified_mail;

(* Ricezione del messaggio 1 *)
Gets R {/Agent TTP, em', Number BothAuth, Number cleartext', Nonce q',
S2TTP' /} ∈ set evs2;

(* R calcola la risposta alla domanda ottenuta dal messaggio 1 *)
r'=ack q;

(* Genera quindi un hash che dovrebbe essere uguale a quello contenuto
in S2TTP *)
hr = Hash {/ Number cleartext', Nonce q', Nonce r', em' /}
[] =>
  Notes R {/S2TTP', Key(RPwD R), hr/} #
  Notes TTP {/S2TTP', Key(RPwD R), hr/} # evs2 ∈ certified_mail

```

Nella formalizzazione di questa regola, compare per la prima volta un

canale confidenziale a consegna garantita. La formalizzazione di tale canale segue la regola indicata in precedenza.

6.3.3 Passo 3: TTP rilascia la chiave ed invia il certificato

```
[| evs3 ∈ certified_mail;

  (* TTP riceve il messaggio al passo 2 *)
  Notes TTP {|S2TTP'', Key(RPwd R'), hr'|} ∈ set evs3;

  (* Ed e' in grado di leggere il contenuto del ticket *)
  S2TTP''=Crypt TTPEncKey {| Agent S, Number BothAuth, Key k', Agent R',
hs' |};

  (* Controlla se gli hash generati da S e da R sono uguali *)
  hs'=hr';

  (* Ed autentica il messaggio controllando la password *)
  pwd''=RPwd R'
|] ⇒
Notes TTP {| Key k', hr'|} #
Notes R' {| Key k', hr'|} #

Says TTP S (Crypt TTPSigKey S2TTP'') #
Gets S (Crypt TTPSigKey S2TTP'') # evs3 ∈ certified_mail
```

In questo passo il server, dopo aver ricevuto una richiesta da parte di R , rilascia a questi la chiave k su un canale confidenziale e a consegna garantita, ed al mittente dindicato nel ticket un certificato di avvenuta consegna su un canale a consegna garantita.

Capitolo 7

Analisi delle proprietà del protocollo

In questo capitolo verranno descritte le proprietà del protocollo che abbiamo dimostrato. In particolare verrà dimostrato che l'esistenza di un certificato di avvenuta ricezione di un messaggio m da parte di un agente R implica che effettivamente R è a conoscenza di tale messaggio.

7.1 Linea dimostrativa

Per dimostrare che R conosce m dimostreremo che, al termine dell'esecuzione del protocollo, R conosce sia il messaggio criptato em , sia la chiave k con la quale tale messaggio è stato criptato. Si parte dal presupposto che TTP è onesto, quindi se rilascia un certificato di avvenuta ricezione allora

1. ha ricevuto una richiesta di rilascio da parte di R .
2. TTP rilascia anche la chiave k ad R .

Il certificato rilasciato, per essere valido, deve essere firmato da TTP . Se si riesce a dimostrare che solo TTP è in grado di apporre la sua firma, allora l'esistenza stessa di un certificato da lui firmato implica che tale certificato sia stato da questi rilasciato. Essendo TTP onesto, l'esistenza di un certificato implica quindi il fatto che la chiave sia stata inviata al destinatario del messaggio.

La richiesta di rilascio contiene una password, che certifica il mittente di tale richiesta, ed un hash che garantisce sul fatto che tale mittente è già a conoscenza di em , messaggio cifrato. Quindi se TTP ha rilasciato il

certificato, allora R conosce sia em che la chiave k , quindi è in grado di dedurre m decifrando em con la chiave k della quale è in possesso.

7.2 Autenticità del certificato

È innanzitutto necessario dimostrare che, se il certificato compare sul traffico, allora esso è stato inviato da TTP . Per poter garantire una proprietà di questo genere si deve poter garantire che la chiave $TTPSigKey$, che il server utilizza per firmare i messaggi, resti segreta, e quindi non cada nelle mani di qualche malintenzionato che potrebbe a questo punto falsificare la firma di TTP . Il miglior modo per garantire la segretezza di una informazione è ovviamente non inviare mai tale informazione sul traffico. Ma una informazione mai utilizzata è ovviamente inutile. Tuttavia, nel caso di chiavi, ci si può limitare a non inviarle mai in forma *esplicita*. Con questo si intende che una chiave non entra mai a far parte di un messaggio, ma al più viene utilizzata per cifrare parti o interi messaggi. Il seguente lemma afferma che nessuna chiave privata, se non quella della spia, viene mai inviata sul traffico.

lemma 1

$$evs \in certified_mail \implies \\ A \neq Spy \longrightarrow Key (invKey (publicKey b A)) \notin parts(knows Spy evs)$$

I lemmi che partono solo dall'osservazione del traffico, come il precedente, vengono detti lemmi di **regolarità**. Se un elemento non compare in alcun modo sui messaggi che la spia è in grado di intercettare, la spia di certo non è in grado di dedurre tale elemento da questi messaggi.

corollario 1

$$evs \in certified_mail \implies \\ A \neq Spy \longrightarrow Key (invKey (publicKey b A)) \notin analz(knows Spy evs)$$

Siamo a questo punto in grado di dimostrare il seguente teorema. Esso afferma che qualunque messaggio firmato da TTP che compare nel traffico è stato spedito da da questo.

teorema 1

```
[| Crypt TTPSigKey X ∈ parts(knows Spy evs);  
  evs ∈ certified_mail  
|] ⇒  
  ∃ A. (Says TTP A (Crypt TTPSigKey X) ∈ set evs)
```

7.2.1 Esempio di dimostrazione per induzione

Come per la maggior parte dei teoremi riguardanti le proprietà di un protocollo, la dimostrazione di questo avviene per induzione sulla lunghezza delle tracce. La definizione induttiva di tali tracce rende immediato questo approccio.

Effettuare una dimostrazione per induzione sulla lunghezza delle tracce significa dimostrare che la proprietà vale per la traccia vuota (caso base), e poi, presupponendo che tale proprietà valga per una traccia evs , dimostrare che essa rimane valida per una traccia più lunga e $\# evs$ (caso induttivo). Tale e può essere un evento o una sequenza di eventi, e può assumere solo i valori specificati dai passi del protocollo. Di seguito viene illustrato uno schizzo di dimostrazione per chiarire questo concetto.

caso base

Come detto prima, il caso base è rappresentato dalla traccia vuota. La proprietà, applicata alla traccia vuota, diventa

```
[| Crypt TTPSigKey X ∈ parts(knows Spy [] );  
  evs ∈ certified_mail  
|] ⇒  
  ∃ A. (Says TTP A (Crypt TTPSigKey X) ∈ set [] )
```

La traccia vuota rappresenta una situazione nella quale nessun evento è ancora avvenuto sulla rete. Idealmente appena la rete è stata attivata. In questa situazione la conoscenza di ogni agente si riduce al proprio stato iniziale. Tuttavia, in questo contesto, allo stato iniziale di qualsiasi agente appartengono solo delle chiavi e nessun crittostato. Per questo motivo l'ipotesi viene falsificata. La tesi suppone che un evento sia avvenuto, nel caso specifico un evento di tipo `Says`. Ovviamente nella traccia vuota nessun evento è ancora avvenuto. Quindi anche la tesi risulta falsa. Essendo false

sia ipotesi sia tesi l'implicazione risulta valida nel caso della traccia vuota, quindi il teorema è dimostrato in questo caso.

Passo 1: il mittente inizia una sessione

Supponiamo ora che la proprietà valga per una certa traccia *evs*. Dimostriamo che vale anche per una traccia del tipo

Says S R { | Agent TTP, Crypt k (Nonce m), Number BothAuth, Number cleartext, Nonce q, S2TTP | }#evs

Il messaggio firmato con *TTPSigKey* potrebbe comparire già sulla traccia *evs*, oppure essere stato inviato con l'ultimo messaggio. Nel primo caso l'ipotesi induttiva assicura la validità del teorema. Supponiamo ora che l'elemento in questione non compaia mai in *evs*. L'unico crittotelesto inviato al passo 1 è una *Nonce* criptata con una chiave *nuova*. Ovviamente *TTPSigKey* non può essere considerata fresca, dato che appartiene allo stato iniziale di *TTP*. Per questo motivo, se nessun messaggio firmato dal server compare su una traccia *evs*, esso non compare neanche sulla stessa traccia estesa con l'evento indicato dal passo 1 del protocollo.

Passo 2: il destinatario richiede la chiave

Il messaggio inviato dal destinatario al passo 2 non viaggia sul traffico, ma su un canale confidenziale. Per questo motivo tale messaggio accresce la conoscenza della spia solo se è la spia stessa ad inviarlo o a riceverlo. Il passo 2 specifica chiaramente che tale messaggio deve essere inviato a *TTP*, resta quindi il caso in cui sia la spia ad inviare tale messaggio. Esaminiamo le componenti di questo messaggio:

- il ticket *S2TTP*
- la password del destinatario
- uno hash generato dal destinatario.

Ovviamente, nè una password, nè un hash sono dei messaggi cifrati. Del ticket, invece, in questo passo non viene specificato il tipo. Tuttavia il protocollo presuppone che un agente che invii il messaggio 2 lo faccia dopo aver ricevuto il messaggio 1, che conteneva già *S2TTP*. Quindi, se il messaggio firmato dal server oggetto della nostra analisi comparisse in *S2TTP*, esso sarebbe già comparso sulla traccia *evs*, rimandandoci all'ipotesi induttiva.

Passo 3: chiave e certificato vengono rilasciati

Con considerazioni simili a quelle svolte per i passi 1 e 2, possiamo affermare che l'unico messaggio della forma $\text{Crypt } TTP\text{SigKey } X$ che viene inviato al passo 3, viene effettivamente inviato da TTP .

```
[| evs3 ∈ certified_mail;  
  Notes TTP {|S2TTP'', Key(RPw d R'), hr'|} ∈ set evs3;  
  S2TTP''=Crypt TTPEncKey {| Agent S, Number BothAuth, Key k', Agent R',  
hs' |};  
  hs'=hr';  
  pwd''=RPw d R'  
|] ⇒  
Notes TTP {| Key k', hr'|} #  
Notes R' {| Key k', hr'|} #  
  
Says TTP S (Crypt TTPSigKey S2TTP'') #  
  
Gets S (Crypt TTPSigKey S2TTP'') # evs3 ∈ certified_mail
```

Intervento della spia : la regola Fake

Le regole denominate **Fake** modellano la possibilità che la spia invii su un determinato canale un messaggio arbitrario. Tale messaggio deve appartenere all'insieme $\text{synth}(\text{analz}(\text{knows } Spy \text{ evs}))$. Questo significa che la spia, a partire da tutti i messaggi atomici dei quali viene a conoscenza dall'osservazione della traccia $\text{analz}(\text{knows } Spy \text{ evs})$, *sintetizza* un messaggio arbitrario da inviare sul canale. I canali utilizzati nel protocollo che stiamo esaminando sono : un canale convenzionale per le comunicazioni tra gli agenti, un canale con consegna garantita per l'invio del certificato al mittente, un canale confidenziale con consegna garantita per le comunicazioni tra il server ed il destinatario.

La formalizzazione data nel capitolo 3 prevede tuttavia la possibilità da parte di un qualsiasi agente di stabilire un canale con un'altro agente a sua scelta. Questo è chiaramente necessario per rendere questa formalizzazione adattabile ad ogni protocollo. Poichè il protocollo in questione prevede che canali con consegna garantita e canali confidenziale con consegna garantita vengano stabiliti solo rispettivamente tra il server ed il mittente e tra il destinatario ed il server, è lecito aspettarsi che nessun agente onesto effettui alcuna comunicazione con un altro agente che non sia TTP utilizzando uno dei suddetti canali. Tuttavia la spia, il cui comportamento esula dalle

regole imposte dal protocollo, potrebbe utilizzare uno di questi canali per inviare delle informazioni ad altri agenti. Qualunque sia il canale utilizzato, il risultato è comunque lo stesso :

La spia invia ad un'altro agente una informazione che è stata in grado di sintetizzare.

Lo studio delle regole Fake, introdotte in un generico protocollo a seconda dei canali utilizzati, si riduce quindi allo studio di questo evento. Nel caso in cui si stia tentando di dimostrare una proprietà di autenticità, come nel teorema in questione, il problema si riduce a dimostrare che la spia non è in grado di falsificare un messaggio. Poichè la garanzia di autenticità di un certificato viene data dalla firma del server, per produrre un falso certificato la spia avrebbe dovuto conoscere la chiave utilizzata per la firma dal server. Poichè il corollario 1 dimostra proprio che la spia non conosce tale chiave, si può allora affermare che essa non è in grado di falsificare la firma di *TTP*.

La regola Reception

Le regole di tipo Reception si rendono necessarie nel caso in cui vengano utilizzati canali nei quali la consegna non viene garantita. Scindendo invio e ricezione in due diverse regole induttive entrano a far parte del protocollo alcune tracce nelle quali alcuni messaggi inviati non vengono mai ricevuti. Nel nostro caso l'unico canale utilizzato nel quale la consegna non viene garantita è il canale convenzionale. Per questo motivo viene inserita solo la regola di ricezione per questo tipo di canale.

```
[| evsr ∈ certified_mail;
Says A B X ∈ set evsr
|] ⇒ Gets B X # evsr ∈ certified_mail
```

Tra le ipotesi troviamo un evento che indica che il messaggio che viene ricevuto era in precedenza stato spedito da qualcuno. Supposto quindi che un agente riceva un messaggio del tipo *Crypt TTPSigKey X*, tale messaggio era già presente nel traffico, rientrando così nell'ipotesi induttiva.

7.3 TTP invia il certificato e la chiave contemporaneamente

TTP, essendo un server *sicuro*, segue alla lettera i dettami del protocollo. Per questo motivo, nello stesso momento nel quale invia al mittente il certificato di avvenuta ricezione, invia al destinatario la chiave che gli permetterà di leggere il messaggio del quale si certifica la ricezione.

teorema 2

```
[| evs ∈ certified_mail;
  S2TTP=Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,
hr|};
  Crypt TTPSigKey S2TTP ∈ parts(knows Spy evs) |]
  ⇒ Notes R {|Key k, hr|} ∈ set evs
```

Per il teorema 1, la presenza di un messaggio della forma *Crypt TTPSigKey S2TTP* nel traffico implica necessariamente che tale messaggio è stato spedito dal server. Alla luce di questa osservazione, l'enunciato del teorema 2 diventa

```
[| evs ∈ certified_mail;
  S2TTP=Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,
hr|};
  Says TTP S Crypt TTPSigKey S2TTP ∈ set evs |]
  ⇒ Notes R {|Key k, hr|} ∈ set evs
```

La dimostrazione procede a questo punto per induzione. Essendo specificato tra le ipotesi un evento, l'analisi del passo induttivo del protocollo si riconduce all'analisi di predicati del tipo :

```
Says TTP S Crypt TTPSigKey S2TTP ∈ set e # evs
  ⇒ Notes R {|Key k, hr|} ∈ set e # evs
```

dove *e* assume di volta in volta il valore degli eventi (o delle liste di eventi) indicati nella definizione del protocollo. L'unico passo nel quale *TTP* invia *in chiaro* un messaggio è il passo 3. Tuttavia in tale passo egli invia anche la chiave su un canale confidenziale con consegna garantita al destinatario.

```

[| evs3 ∈ certified_mail;
  Notes TTP {|S2TTP'', Key(RPw d R'), hr'|} ∈ set evs3;
  S2TTP''=Crypt TTPEncKey {| Agent S, Number BothAuth, Key k', Agent R',
hs' |};
  hs'=hr';
  pwd''=RPw d R'
|] ⇒
Notes TTP {| Key k', hr'|} #

Notes R' {| Key k', hr'|} #
Says TTP S (Crypt TTPSigKey S2TTP'') #

Gets S (Crypt TTPSigKey S2TTP'') # evs3 ∈ certified_mail

```

Questo permette di concludere la dimostrazione, affermando che

Se un certificato che attesta l'avvenuta ricezione di un messaggio da parte di un agente R viene spedito sul traffico, allora R riceve dal server la chiave necessaria per decifrare il messaggio in questione.

7.4 R conosce la chiave

Se il certificato compare nel traffico, esso è stato generato da TTP . Ed è stato inviato nello stesso momento nel quale TTP ha inviato la chiave ad R . Poichè TTP invia la chiave su un canale a consegna garantita, essa sicuramente raggiungerà R .

In definitiva la presenza di tale certificato nel traffico implica che R è a conoscenza della chiave.

teorema 3

```

[| evs ∈ certified_mail;
  S2TTP=Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,
hr|};
  Crypt TTPSigKey S2TTP ∈ parts(knows Spy evs) |]
  ⇒ Key k ∈ analz(knows R evs)

```

Siamo nelle ipotesi del teorema 2. Per questo motivo possiamo affermare che R ha ricevuto la chiave su un canale confidenziale a consegna garantita, quindi ne è a conoscenza.

7.5 Autenticità della richiesta

Se TTP ha rilasciato un certificato e la corrispondente chiave, lo ha fatto perchè ha ricevuto da un agente R una richiesta in tal senso. Vedremo nel seguito che sarà proprio il contenuto di tale richiesta a dimostrare che l'agente che la ha spedita è già a conoscenza del messaggio cifrato em .

Innanzitutto bisogna però verificare che nessun altro agente, in particolare la spia, è in grado di spacciarsi per R . Se così fosse la spia, venuta a conoscenza del messaggio criptato em potrebbe inviare una richiesta di rilascio per il messaggio em a nome di R , facendo credere al server che R sia precedentemente venuto a conoscenza di tale em .

7.5.1 Autenticazione NOAuth ed SAAuth, un possibile attacco

Esaminiamo gli elementi che compongono $S2TTP$, per come $S2TTP$ appare al server quando viene ricevuto col messaggio 2. Esso contiene due nomi di agenti, l'indicazione sulla modalità di autenticazione, una chiave ed un hash. Non compare nessun segreto condiviso col server. In oltre tale ticket viene cifrato con la chiave $TTPEncKey$, che è una chiave pubblica. Alla luce di queste osservazioni si deduce che ogni agente può generare un ticket di questo tipo usando una chiave che conosce, un hash qualsiasi e specificando due agenti a suo piacimento.

Se la modalità di autenticazione prevista è **NOAuth** o **SAAuth**, alla ricezione del messaggio 2 TTP controlla solo che l'hash h_s , contenuto in $S2TTP$, corrisponda allo hash h_r , contenuto nel messaggio. Poichè un generico agente è in grado di generare $S2TTP$, specificando un hash a suo piacimento, tale agente può inviare un messaggio della forma del messaggio 2, curandosi di usare $h_r = h_s$. Tale messaggio verrebbe accettato dal server, che provvederebbe a spedire la chiave ed il certificato agli ignari agenti indicati in $S2TTP$.

Ma la spia può fare di più. L'unica parte del certificato che contiene indicazioni sul messaggio è proprio tale hash. A questo punto la spia non deve fare altro che confezionare l'hash nella forma prevista dal protocollo, curandosi di usare un messaggio criptato con la stessa chiave specificata in $S2TTP$, una *domanda* q uguale a quella in $S2TTP$, e procedere come indicato in precedenza. Il server accetterà il messaggio 2, rilasciando sulla rete (quindi in preda alla spia) un certificato che testimonia una conversazione che non è avvenuta, con un messaggio ed un testo in chiaro scelti dalla spia.

Si può ipotizzare uno scenario nel quale la spia sia proprio il mittente del messaggio. In tal modo esso ottiene un certificato che indica che il destinatario ha letto il messaggio, quando a questo non è mai arrivato il messaggio, ma solo la chiave.

7.5.2 Autenticazione TTPAuth

Tale modalità di autenticazione sembra offrire sufficienti garanzie. Esaminiamo come *TTP* effettua il controllo sull'autenticità di una richiesta. La parte della richiesta che permette a *TTP* di autenticare *R* è la password in tale richiesta contenuta. Similmente a quanto visto per l'autenticità del certificato, l'autenticità della richiesta si basa sulla segretezza di questa password. Infatti, se nessun agente, ed in particolare la spia, è a conoscenza di tale password, nessuno è in grado di sintetizzare un messaggio che la contenga. Il seguente lemma afferma che la spia non conosce altre password, se non la propria.

lemma 2

$$evs \in \text{certified_mail} \implies A \neq \text{Spy} \longrightarrow \text{Key}(\text{RPwd } A) \notin \text{parts}(\text{spies } evs)$$

La dimostrazione avviene per induzione e si basa sul fatto che l'unico canale sul quale le password vengono spedite è un canale confidenziale stabilito tra il proprietario della password e *TTP*. Questo basta ad affermare che la password di un agente onesto resta confidenziale tra tale agente ed il server. A partire da questo lemma è immediato dimostrare l'autenticità di una richiesta.

teorema 4

```
[| evs ∈ certified_mail;
  Notes TTP {|
    Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R, hr|},
    Key (RPwd R),
    hr|} ∈ set evs
|] ⇒
Notes R {|
  Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R, hr|},
  Key(RPwd R), hr|} ∈ set evs
```

Questo teorema afferma che per *TTP* è possibile verificare la provenienza

di una richiesta. Per questo motivo vengono specificate le componenti che *TTP* è in grado di esaminare (la password e il ticket). Per quanto riguarda h_r , il server non è in grado di stabilire come è stato generato. Per questo motivo viene lasciato non specificato. La dimostrazione di questo teorema è molto simile a quella del teorema 1.

7.6 R conosce il messaggio criptato

Se *R* invia a *TTP* una richiesta, egli ha *costruito* lo Hash $h_r = Hash(cleartext, q, r, em)$. Osserviamo che la spia potrebbe non costruire *ex novo* questo hash, ma riutilizzarne uno di una sessione precedente. Il seguente lemma afferma che se la spia è in grado di sintetizzare un hash di questa forma ha precedentemente appreso *em*.

lemma 3

```
[| evs ∈ certified_mail ;
  Hash {|Number cleartext, Nonce q, Nonce r, em|} ∈ parts(knows Spy evs)
|] ⇒ em ∈ synth(analz(knows Spy evs))
```

Questa proprietà discende direttamente dal fatto che gli hash viaggiano solo sul canale confidenziale, oppure incapsulati nel ticket. Per questo motivo la spia non è in grado di dedurre hash utilizzati in sessioni che non la vedono direttamente coinvolta. Il prossimo teorema asserisce che un agente che fa richiesta per la chiave k , conosce di certo *em* specificato in h_r .

teorema 5

```
[| Notes R' {|
  Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,
    Hash {| Number cleartext, Nonce q, Nonce r, em |} |},
  Key (RPwd R),
  Hash {| Number cleartext, Nonce q, Nonce r, em |}
|} ∈ set evs;

  evs ∈ certified_mail
|] ⇒ em ∈ synth(analz(knows R evs))
```

7.6.1 Autenticazione TTPAuth e BothAuth, una esecuzione anomala

Il teorema 5 non asserisce semplicemente che un agente che invii una richiesta sia precedentemente venuto a conoscenza del messaggio cifrato, ma che possa essere stato lui stesso a generare tale messaggio. Il controllo della password garantisce TTP sul mittente del messaggio 2. In tal modo il server può controllare che chi richiede la chiave per un dato messaggio sia l'effettivo destinatario di tale messaggio. Tuttavia TTP continua a non avere alcuna evidenza su chi ha generato il ticket $S2TTP$. Esaminiamo infatti la struttura di tale ticket $S2TTP$. Esso contiene il nome del mittente, il nome del ricevente, una chiave ed uno hash. Tutto cifrato con $TTPEncKey$, una chiave pubblica. Per questo chiunque è in grado di generare un messaggio di tale tipo partendo da una chiave che già conosce ed indicando un mittente ed un ricevente arbitrari. In particolare, questa debolezza permette ad R di *ingannare* TTP , facendogli credere che un arbitrario agente S abbia deciso di eseguire il protocollo con lui.

In realtà questo non contraddice gli obiettivi del protocollo. Infatti, se R è in grado di spedire il messaggio 2, allora o l'agente S ha effettivamente iniziato il protocollo con lui, oppure R è in grado di sintetizzare tutti i componenti del messaggio 2, in particolare $h_r = \text{Hash}(\text{cleartext}, q, r, em)$ e la chiave k contenuta nel ticket.

Il lemma 3 afferma che un agente riesca a produrre un hash generato da un insieme di messaggi tra i quali figura em , implica che egli stesso conosca em . Quindi, se un agente invia una richiesta per farsi rilasciare la chiave per un dato messaggio, tale agente sicuramente già conosce il messaggio cifrato. Al passo successivo TTP invierà la chiave a colui il quale l'aveva richiesta (mettendolo in pratica a conoscenza del messaggio) e il certificato all'agente specificato in $S2TTP$ come mittente. L'esistenza di tale certificato indica quindi solo che il destinatario del messaggio indicato conosce tale messaggio. Il fatto che anche il mittente venga indicato nel certificato non aggiunge in realtà alcuna informazione.

7.6.2 Il certificato afferma che il destinatario conosce il messaggio criptato

Dall'autenticità del certificato, dal lemma appena dimostrato e dal fatto che TTP invia il certificato solo se stimolato da una opportuna richiesta, si deduce che l'esistenza del certificato implica che R conosce em .

teorema 6

```
[| Crypt TTPSigKey S2TTP ∈ parts(spies evs);  
  S2TTP=Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,  
  hs|};  
  hs = Hash {| Number cleartext, Nonce q, Nonce r, em |};  
  evs ∈ certified_mail  
|] ⇒ em ∈ synth(analz(knows R evs))
```

La dimostrazione del teorema si svolge secondo il seguente schema

1. Per il teorema 1, se il certificato compare sul traffico, esso è allora stato generato da *TTP*.
2. Dalla definizione del protocollo, e dal fatto che *TTP* è onesto, si deduce che *TTP* rilascia il certificato al passo 3 solo se ha ricevuto una richiesta in tal senso.
3. Sempre dalla definizione del protocollo, il server rilascia il certificato solo se alcuni controlli vanno a buon fine. Innanzitutto verifica che la password contenuta nella richiesta sia la password dell'agente indicato come destinatario nel ticket. Se questo controllo va a buon fine, per il teorema 2 il server ha la certezza che la richiesta ricevuta proviene effettivamente dall'agente indicato come mittente in *S2TTP*. Successivamente viene controllato che lo hash indicato nella richiesta corrisponda a quello contenuto nel ticket. Se questo avviene, poichè il ricevente non è in alcun modo in grado di leggere il contenuto di *S2TTP*, ma solo *TTP* può farlo, significa che mittente e ricevente concordano sull'insieme di messaggi necessari a generare tale hash.
4. Nell'ipotesi che *em* compaia tra i messaggi necessari a generare questo hash, il teorema 3 ci dice che *R* è sicuramente a conoscenza di tale *em*

7.7 Il certificato afferma una avvenuta ricezione

I risultati ottenuti permettono di dimostrare che la presenza di un certificato che indichi l'avvenuta ricezione di un messaggio m da parte di un agente R implica che R conosca o sia in grado di sintetizzare tale m .

teorema 7

```
[| evs ∈ certified_mail;  
  k ∈ symKeys;  
  S2TTP=Crypt TPEncKey {| Agent S, Number authoption, Key k, Agent R,  
  hs|};  
  hs = Hash {| Number cleartext, Nonce q, Nonce r, Crypt k m |};  
  Crypt TTPSigKey S2TTP ∈ parts(spies evs)  
|] ⇒ m ∈ synth(analz(knows R evs))
```

Un agente S che voglia dimostrare che un altro agente R conosce il messaggio m , produrrà quindi al giudice *cleartext*, q , r , k ed m , oltre ovviamente al certificato. Il giudice, a questo punto, sintetizza l'hash e lo confronta con quello contenuto nel certificato. Controlla nel certificato anche la chiave k e l'agente R . Se tutti i controlli vanno a buon fine, ha la certezza che R conosce o è in grado di sintetizzare il messaggio m .

Il teorema precedente parla di un messaggio generico. Nella nostra formalizzazione il messaggio viene rappresentata da una **Nonce**, un messaggio atomico che non può essere facilmente indovinato. Essendo un messaggio atomico, non può nemmeno essere sintetizzato. Per questo motivo, il teorema precedente può essere enunciato nella sua forma definitiva :

teorema 7a

```
[| evs ∈ certified_mail;  
  k ∈ symKeys;  
  S2TTP=Crypt TPEncKey {| Agent S, Number authoption, Key k, Agent R,  
  hs|};  
  hs = Hash {| Number cleartext, Nonce q, Nonce r, Crypt k Nonce m |};  
  Crypt TTPSigKey S2TTP ∈ parts(knows Spy evs)  
|] ⇒ Nonce m ∈ analz(knows R evs)
```

7.8 Il mittente non è in grado di sintetizzare il certificato

In realtà è possibile dimostrare una proprietà più forte di quella appena enunciata. La proprietà appena enunciata parte dall'ipotesi che il certificato appartenga al traffico. Tale ipotesi, per essere verificata da un eventuale giudice, presuppone che il giudice fosse *in ascolto* nel momento in cui tale certificato veniva spedito, e che fosse in grado di intercettare un messaggio che non era a lui diretto, poichè il server spedisce questo certificato al mittente e non ad un eventuale giudice. Il fatto che un agente sia in grado di presentare un certificato ad un giudice presuppone che egli stesso sia in grado di ottenere tale certificato dall'osservazione del traffico, o che sia in grado di generarlo *ex novo*. Nel primo caso egli avrebbe potuto ricavare tale certificato sia dalle informazioni ottenute attraverso il canale convenzionale, sia da quelle ottenute attraverso il canale confidenziale. I messaggi trasmessi attraverso il canale convenzionale entrano tutti a far parte della conoscenza della spia. Tutte le parti di questi, anche nel caso in cui la spia non sia in grado di ottenerle perchè cifrate usando una chiave che lei ignora, entrano a far parte dell'insieme $\text{parts}(\text{knowsSpyevs})$. Le informazioni invece delle quali l'agente viene a conoscenza attraverso il canale confidenziale non sono comprese in questo insieme. In oltre, come rilevato prima, nel momento in cui un agente presenta un certificato al giudice, potrebbe essere stato lui stesso a confezionarlo, senza averlo mai inviato sul canale convenzionale. Se quindi supponiamo che un agente sia in grado di sintetizzare un certificato nella forma richiesta e poi presentarlo ad un giudice, senza aver mai inviato tale certificato sulla rete, tale certificato non apparterrà all'insieme $\text{parts}(\text{knowsSpyevs})$. Per questo motivo il teorema appena enunciato non cattura la semantica di un evento del tipo S presenta al giudice un certificato C, ma solo quella di un evento Il certificato C è stato spedito sul traffico. Rafforziamo quindi le ipotesi del teorema, in maniera tale da modellare una situazione nella quale un agente presenti un certificato al giudice, indipendentemente se tale certificato sia stato spedito o meno sul traffico. Tutti i messaggi dei quali un agente A viene a conoscenza sono contenuti nell'insieme $\text{knows } A \text{ evs}$. Le parti di questi messaggi che tale agente è in grado di ricavare dai messaggi stessi sono contenute nell'insieme $\text{analz}(\text{knows } A \text{ evs})$. Tutti i messaggi che, per composizione o applicando una funzione crittografica, egli è in grado di sintetizzare sono rappresentati dall'insieme $\text{synth}(\text{analz}(\text{knows } A \text{ evs}))$. Il fatto che un agente sia in grado di presentare un messaggio X, può essere formalizzato dal predicato

$X \in \text{synth}(\text{analz}(\text{knows } S \text{ evs}))$

Alla luce di queste osservazioni, modifichiamo l'enunciato del teorema precedente in maniera tale formalizzi una situazione nella quale un agente presenta un certificato ad un giudice.

teorema 7b

```
[| evs ∈ certified_mail;  
  k ∈ symKeys;  
  S2TTP=Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,  
  hs|};  
  hs = Hash {| Number cleartext, Nonce q, Nonce r, Crypt k Nonce m |};  
  Crypt TTPSigKey S2TTP ∈ synth(analz(knows S evs))  
|] ⇒ Nonce m ∈ analz(knows R evs)
```

Per dimostrare questo teorema, è sufficiente dimostrare che se un agente è in grado di sintetizzare un certificato, allora tale certificato apparteneva al traffico, riconducendosi quindi alle ipotesi del teorema 7a. Affinchè un agente sia in grado di sintetizzare ex novo un documento firmato da TTP , è necessario che tale agente conosca la chiave $TTPSigKey$ tramite la quale tale firma viene apposta. Dimostriamo che tale chiave non entra mai a far parte dei messaggi ricevuti da un agente

lemma 1a

```
evs ∈ certified_mail ⇒  
∀ A. A ≠ TTP → Key TTPSigKey ∉ parts(knows A evs)
```

Tale lemma è chiaramente un lemma di regolarità. Per questo motivo la dimostrazione risulta molto simile a quella fornita per il lemma 1. Tale lemma ci viene inoltre in aiuto per i casi Fake, permettendoci di affermare che la spia non può aver comunicato $TTPSigKey$ al nostro agente A , poichè essa stessa non era a conoscenza di tale chiave.

Nessun agente, nemmeno la spia, è in grado di sintetizzare un messaggio apponendovi la firma di TTP . Quindi, se un agente presenta un documento di questo tipo, sicuramente tale documento gli è stato inviato dal server. Ed il server invia un documento sul quale appone la propria firma solo al passo 3, nel momento in cui invia su un canale confidenziale la ricevuta di avvenuta consegna al mittente. Avvenendo tale trasmissione sul canale a

consegna garantita, tale messaggio entra automaticamente a far parte della conoscenza della spia, quindi anche all'insieme $\text{knows Spy } evs$.

lemma 4

```
[| evs ∈ certified_mail;  
Crypt TTPSigKey X ∈ synth(analz(knows S evs)) |]  
⇒ Crypt TTPSigKey X ∈ parts(knows Spy evs)
```

Applicando questo lemma alle ipotesi del teorema 7b otteniamo esattamente le ipotesi del teorema 7a. Questo ci permette di concludere la dimostrazione affermando che :

Se un qualsiasi agente presenta un certificato firmato da TTP nel quale si attesta che un agente R ha richiesto di leggere un messaggio m e quindi gli sono stati forniti tutti gli strumenti necessari a leggere tale messaggio, allora effettivamente R ha affermato la volontà di leggere questo messaggio e successivamente ha appreso m .

Capitolo 8

Una possibile variante del protocollo

Il protocollo esaminato in questo documento, mentre offre ad un mittente un'*evidenza* per dimostrare che il destinatario ha effettivamente ricevuto il messaggio, non offre al destinatario alcuna garanzia. In particolare il destinatario non ha alcuna certezza su chi sia il mittente del messaggio. Questo accade perchè il protocollo non prevede nessun meccanismo per verificare l'autenticità del messaggio 1, ed in particolare del ticket contenuto in questo messaggio. La possibilità di una esecuzione anomala come quella indicata nel capitolo precedente preclude inoltre al destinatario la possibilità di dimostrare che il mittente ha iniziato una sessione con lui, avendo R la possibilità di iniziare il protocollo direttamente al passo 2 specificando un agente a suo piacimento come mittente. In questo capitolo verranno discusse delle varianti del protocollo che offrono maggiori garanzie al destinatario.

8.1 Il sistema postale : raccomandate con ricevuta di ritorno

Il protocollo di e-mail certificata si pone un obiettivo simile a quello che si pongono le raccomandate con ricevuta di ritorno nel sistema postale : dare al mittente la certezza che un messaggio sia stato recapitato e la possibilità di dimostrarlo. Le analogie tra i due sistemi sono molteplici. Entrambi si basano sulla collaborazione di un terzo *attore*, del quale non viene messa in dubbio l'integrità. Nell'ambito delle raccomandate tale ruolo è svolto dal sistema postale stesso, nel nostro protocollo da un server considerato sicuro. In entrambi i casi non viene garantito che il destinatario legga il

messaggio, ma che tale messaggio sia stato notificato e messo a disposizione di questo. Infatti, quando il postino consegna la raccomandata, non obbliga il destinatario a leggere la lettera, ma la consegna e si fa rilasciare da questi una dichiarazione di avvenuta consegna. Nel protocollo in questione accade qualcosa di lievemente diverso: è il destinatario che, con l'invio della richiesta per il rilascio della chiave, afferma di aver ricevuto il messaggio, ma di non poterlo leggere senza che il server rilasci la chiave. Egli comunque afferma di essere a conoscenza del fatto che tale messaggio gli è stato spedito, ed il server si occupa di metterglielo a disposizione consegnandogli la chiave.

Quando una persona riceve una raccomandata, le informazioni che tale destinatario ottiene riguardo il mittente del messaggio provengono innanzitutto dalla fiducia che ripone nel sistema postale. Per inviare tale raccomandata il mittente si è recato in un ufficio postale *autenticandosi* tramite un documento. Una ulteriore assicurazione sul mittente nasce da una eventuale firma apposta sul messaggio. In entrambi i casi viene offerta la possibilità al destinatario di affermare in sede legale di aver ricevuto un messaggio da uno specifico mittente. Nel primo caso chiamando in causa il servizio postale, nel secondo unicamente esibendo il messaggio. Esaminiamo due varianti del protocollo che offrano assicurazioni al destinatario paragonabili a quelle offerte al destinatario di una raccomandata, prendendo le mosse da come il sistema postale gestisce tali raccomandate con ricevuta di ritorno.

8.2 Prima variante: il mittente firma il messaggio

Supponiamo di disporre di una infrastruttura di chiavi pubbliche (PKI). Sia *SSigKey* la chiave privata dell'agente *S* utilizzata per la firma, ed *SVerKey* la chiave pubblica necessaria per la verifica della firma. Supponiamo che il mittente *S* spedisca al passo 1 il messaggio cifrato *em* firmato con *SSigKey*. L'evento al passo 1 diventa :

Says S R { | TTP, Crypt SSigKey(Crypt k m) , AuthMode, cleartext, q, S2TTP | }

Ricevendo questo messaggio *R* è in grado di ricavare *Crypt k m*, essendo pubblica la chiave necessaria per ottenerlo dal messaggio. In oltre, dopo aver verificato la firma, ha la certezza che tale *em* sia stato generato da *S*. Infine può esibire *Crypt SSigKey(Crypt k m)* per dimostrare che *Crypt k m* è stato generato da *S*. Questa soluzione darebbe quindi ad *R* la possibilità di dimostrare, esibendo il messaggio, che *S* lo abbia effettivamente spedito. Tuttavia non darebbe alcuna garanzia su chi sia il destinatario di tale

messaggio nelle intenzioni di S . Supponiamo infatti che S esegua il protocollo con R spedendogli un messaggio m usando la variante del protocollo in esame. Terminata l'esecuzione del protocollo R sarebbe in possesso della chiave k , del messaggio m e dello stesso messaggio firmato da S e cifrato con chiave k (dal passo 1 del protocollo). A questo punto R potrebbe eseguire il protocollo con un'altro agente R' , indicando S come mittente, e riutilizzando la chiave ed il messaggio della comunicazione precedente. Se R' decide di portare a termine il protocollo compare sul traffico un certificato che indica che S ha spedito il messaggio m ad R' . Questa imperfezione nasce dal fatto che il messaggio firmato ed il destinatario non sono collegati in alcun modo. Esaminiamo allora una ulteriore variante che preveda che il mittente citi esplicitamente il destinatario nel messaggio firmato. L'evento al passo uno, in questa nuova formulazione, diventa

Says S R { | TTP, Crypt SSigKey{ | R, Crypt k m | } , AuthMode, cleartext, q, S2TTP | }

Nessun agente oltre S sarebbe stato in grado di generare un messaggio del tipo

Crypt SSigKey{ | R, Crypt k m | }

essendo *SSigKey* una chiave privata. Per questo motivo l'esistenza di un messaggio di questo tipo implica necessariamente la volontà, da parte di S , di eseguire una sessione del protocollo con R al fine di trasmettergli il messaggio m .

Questa variante del protocollo raggiunge in pieno gli obiettivi che ci eravamo prefissi. Essa presuppone tuttavia l'esistenza di una infrastruttura di chiavi pubbliche, non prevista tra i prerequisiti originari del protocollo, aggiungendo quindi difficoltà e costo computazionale in una eventuale implementazione dello stesso.

8.3 Seconda variante: il mittente esibisce un documento all'ufficio delle postale

L'esecuzione anomala indicata nel paragrafo precedente è resa possibile dal fatto che non vengono fornite garanzie in merito a chi ha generato *S2TTP*. Per garantire l'autenticità di tale messaggio senza appesantire ulteriormente eventuali implementazioni potremmo utilizzare un elemento già previsto dal

protocollo: la **password** del mittente. Supponiamo allora che il mittente inserisca nel ticket oltre agli elementi già presenti anche la sua password.

$$S2TTP = \text{Crypt } TTP\text{EncKey } \{ | \text{Agent } S, \text{RPwd } S, \text{AuthMode}, \text{Key } k, \text{Agent } R, \text{hs} | \}$$

$S2TTP$ viene criptato con $TTP\text{EncKey}$. Per questo motivo, una volta *confezionato*, può essere aperto solo dal server. Questo garantisce che la password, inviata all'interno di un messaggio di questo tipo, resti confidenziale tra l'agente e TTP . Similmente a quanto avviene per la richiesta di rilascio della chiave (passo 2 del protocollo), poichè nessun agente oltre S conosce $\text{RPwd } S$, un ticket che contenga tale password è stato sicuramente generato da S . Per questo motivo, esibendo il ticket ricevuto al passo 1 il destinatario è in grado di dimostrare che S ha iniziato una sessione del protocollo con lui.

8.4 Il protocollo di e-mail certificata nella realtà

Utilizzando questa variante del protocollo, le garanzie offerte al destinatario provengono unicamente dalla fiducia riposta nel server. Per poter utilizzare tali garanzie in ambito giudiziario la fiducia nei confronti di TTP dovrebbe essere condivisa anche dall'intero sistema giudiziario al quale la controversia è affidata. Lo stesso problema si pone per le garanzie offerte al mittente. Per poter affermare che un agente R abbia ricevuto sia un determinato messaggio criptato sia la chiave per decifrarlo è necessario avere la assoluta certezza che

1. TTP rilascia la chiave insieme al certificato.
2. TTP rilascia il certificato perchè ha ricevuto una richiesta in tal senso da parte del destinatario.
3. L'esistenza una richiesta per il rilascio della chiave necessaria per decifrare un determinato messaggio m implica necessariamente che chi invia tale richiesta sia già a conoscenza del messaggio m cifrato con tale chiave.

In pratica bisogna avere la certezza che il server in questione si comporta seguendo le regole imposte dal protocollo. Affinchè tali requisiti possano essere sfruttati in sede legale è quindi di nuovo necessario che l'organo preposto

alla risoluzione della controversia riponga completa fiducia nel server utilizzato come intermediario nella comunicazione. Ottenere tale fiducia significa non solo offrire garanzie sull'onestà di chi gestisce tale server, ma riuscire anche a dimostrare l'efficienza e la sicurezza del sistema di autenticazione e del server stesso. Nel caso del sistema giudiziario di uno stato sono possibili due scenari : nel primo tali server sono gestiti direttamente dallo stato, nel secondo sono gestiti da enti privati certificati. Il secondo scenario implica comunque l'esistenza di un ente statale che svolga il ruolo di certificatore.

La variante del protocollo presa in esame differisce dal protocollo originario solo nel primo passo. Poichè le considerazioni appena svolte riguardano solo i passi successivi al primo, esse possono essere estese anche al protocollo di e-mail certificata nella sua versione originaria. Quindi nella versione originaria del protocollo e in questa seconda variante presentata, nonostante non sia prevista l'esistenza di una infrastruttura di chiavi pubbliche, nel caso in cui si voglia far assumere valenza legale alle comunicazioni effettuate utilizzando tali protocolli è previsto comunque un requisito la cui disponibilità nella realtà non è scontata : l'esistenza di uno o più server certificati, e quindi di una autorità di certificazione statale.

8.5 Terza variante : certificazione del mittente e del ricevente attraverso la firma

Supponiamo invece l'esistenza di una infrastruttura di chiavi pubbliche. Presentiamo una variante del protocollo nella quale non sia necessario un sistema di password per l'autenticazione degli utenti presso *TTP*, e la fiducia nei confronti di *TTP* stesso assuma un ruolo meno determinante.

8.5.1 Passo 1

Sia il primo passo quello indicato nella seconda formulazione della prima variante :

Says S R { | TTP, Crypt SSigKey{ | R, Crypt k m | } , AuthMode, cleartext, q, S2TTP | }

Come precedentemente rilevato, la ricezione di un messaggio di tale tipo permette al destinatario di affermare che *S* ha iniziato un protocollo con lui allo scopo di inviargli il messaggio *m*. Tale formulazione offre in oltre al destinatario garanzie sul mittente del messaggio.

8.5.2 Passo 2

Non avendo più a disposizione un sistema di password, l'unico modo col quale il ricevente possa autenticarsi presso TTP al momento di richiedere la chiave è di firmare tale richiesta. La necessità di far viaggiare le richieste su un canale sicuro nel protocollo originale nasce dalla necessità di mantenere confidenziale la password contenuta in tale richiesta, e necessaria a garantirne l'autenticità. Essendo in questo contesto la firma a garantire l'autenticità del messaggio, l'utilizzo di un canale confidenziale non è più necessario. Indichiamo con $RSigKey$ la chiave utilizzata da R per la firma, e con $RVerKey$ la corrispondente chiave per verificare tale firma. L'evento al passo 2 diventa :

```
Says R TTP Crypt RSigKey {|S2TTP', hr|} #
```

Per considerazioni simili a quelle svolte riguardo al messaggio inviato al passo 1, l'esistenza di un messaggio firmato da R implica che necessariamente tale messaggio sia stato generato da R . Il messaggio inviato in questo passo è a tutti gli effetti un documento nel quale R afferma di aver ricevuto em , necessario per generare h_r , e di volerlo leggere ottenendone la chiave da TTP .

8.5.3 Passo 3

Non essendo l'integrità del server un requisito del protocollo, al mittente poco importa di ricevere un certificato nel quale il TTP afferma di aver effettivamente consegnato la chiave ad R , in modo che questi venisse a conoscenza di m . Ottenedo invece un documento firmato da R che ne manifesti l'intenzione di leggere il messaggio in questione, S avrebbe la possibilità di utilizzare tale documento in sede legale. Per quanto riguarda il destinatario, TTP deve spedirgli la chiave affinché possa leggere il messaggio. Nell'ipotesi che TTP abbia ricevuto una richiesta per il rilascio della chiave da un agente R' , l'evento al passo 3 diventa :

```
Notes TTP {| Key k', hr'|} #  
Notes R' {| Key k', hr'|} #  
Says TTP S Crypt RSigKey' {|S2TTP'', hr'|} #  
Gets S Crypt RSigKey' {|S2TTP'', hr'|}
```

8.5.4 Osservazioni

Eliminando l'ipotesi che il server sia sicuro, in realtà non è possibile dimostrare che il ricevente ha ottenuto la chiave, è solo possibile dimostrare che questo è venuto a conoscenza del fatto che S ha inteso spedirgli un messaggio. Paradossalmente, tutti gli obiettivi riguardanti le garanzie che si intendeva dare ad R sono stati raggiunti. Per garantire che l'esistenza del certificato implichi necessariamente la conoscenza di k da parte di R è necessario reintrodurre l'ipotesi che il server sia sicuro, anche se in una forma più debole di quella richiesta dal protocollo di Abadi et al. Riesaminiamo quale semantica per questa ipotesi era necessario assumere nella versione originale del protocollo affinché il certificato fosse realmente una garanzia della conoscenza del messaggio da parte di R .

1. TTP rilascia la chiave insieme al certificato.
2. TTP rilascia il certificato perchè ha ricevuto una richiesta in tal senso da parte del destinatario.
3. L'esistenza una richiesta per il rilascio della chiave necessaria per decifrare un determinato messaggio m implica necessariamente che chi invia tale richiesta sia già a conoscenza del messaggio m cifrato con tale chiave.

In questa formulazione l'ipotesi 2 non è più necessaria, in quanto S , al termine della sessione, è in possesso della richiesta inoltrata da R al fine di ottenere la chiave per leggere il messaggio. Tale richiesta, essendo firmata da R , rappresenta un documento sufficiente ad affermare la volontà di R di ottenere il messaggio. L'ipotesi 3 viene soddisfatta dalle proprietà del protocollo in tutte le versioni presentate. In particolare per la versione originale del protocollo questo predicato è stato dimostrato nel capitolo precedente. Per questo motivo questo predicato può essere considerato una proprietà del protocollo e non più un'ipotesi. L'unica ipotesi che rimane necessaria è quindi l'ipotesi 1. Non si richiede quindi il corretto comportamento del server in tutte le fasi del protocollo e nelle rispettive fasi di verifica, si richiede solo che questi invio del certificato al passo 3 solo se ha inviato ad R la chiave. Supponiamo adesso che si sia aperta una controversia tra i due agenti S ed R . S espone il certificato, la richiesta effettuata da R per il rilascio della chiave. Fornisce in oltre tutti gli elementi necessari per generare l'hash contenuto in tale richiesta, fra i quali em . Tale richiesta rappresenta una prova inoppugnabile che R è a conoscenza di em . A questo punto il giudice convoca TTP , e gli chiede se ha rilasciato una chiave ad R . Il giudice

confronta questa chiave con em . Se tale chiave permette di leggere il messaggio, allora la controversia si risolve in favore di S . In una situazione reale le tracce lasciate sulla rete dalla comunicazione attraverso cui TTP mette R a conoscenza della chiave potrebbero costituire un valido supporto alle dichiarazioni del server.

Capitolo 9

Conclusioni

Fornire la specifica di un protocollo complesso come protocollo di seconda generazione, ossia utilizzando la nozione di canale, ne aumenta notevolmente la leggibilità. Ne facilita inoltre l'implementazione, lasciando agli implementatori la scelta im merito a quali protocolli utilizzare per realizzare suddetti canali, sempre presupponendo che tali protocolli soddisfino i requisiti richiesti dal canale. Per quanto riguarda la formalizzazione e la verifica del protocollo, essa risulta notevolmente semplificata, nel caso in cui sia stata precedentemente approfondita la formalizzazione dei canali utilizzati. In questo documento sono state fornite le formalizzazioni di tre differenti canali : il canale convenzionale, il canale che garantisce la consegna, ed il canale confidenziale che garantisce la consegna. Di questi in realtà solo l'ultimo, e anche questo parzialmente, può essere considerato un canale di *seconda generazione*, in quanto presuppone l'utilizzo di un protocollo che raggiunga l'obiettivo di confidenzialità. Gli altri due canali riguardano più che altro requisiti che devono essere forniti dallo strato sottostante dell'architettura di rete. In particolare requisiti del livello di trasporto. Mentre il canale convenzionale è in realtà il canale *nativo* per le formalizzazioni nell'approccio induttivo, il canale con consegna garantita può essere considerato un'estensione di tale approccio che permette di catturare la semantica di un evento del tipo *un messaggio X viene inviato e sicuramente arriva a destinazione*. La nozione stessa da canale induce a suddividere i protocolli di comunicazione in una gerarchia, nella quale al primo livello vengono inseriti i protocolli di sicurezza, al secondo vengono inseriti i protocolli di seconda generazione, al terzo livello i protocolli che utilizzano al loro interno protocolli del livello sottostante e così via. In questa gerarchia i protocolli di trasporto si possono immaginare collocati al *livello 0*. In particolare possiamo considerare collo-

cati a questo livello tutti i protocolli che raggiungano *obbiettivi di trasporto*, ad esempio consegna di messaggi, consegna garantita, integrità dei messaggi. Esaminiamo nuovamente la definizione di protocollo di sicurezza di seconda generazione

Si definisce protocolli di seconda generazione un protocollo di sicurezza che preveda durante il suo svolgimento l'esecuzione di altri protocolli di sicurezza, specificando solo i requisiti che tali protocolli devono soddisfare, e non i dettagli del loro funzionamento.

Tale classe di protocolli rappresenta il secondo livello della gerarchia appena introdotta. Ci si può chiedere allora se è possibile introdurre una gerarchia simile per gli obiettivi di sicurezza, ogni livello della quale possa essere messo in corrispondenza di un livello della gerarchia dei protocolli di sicurezza. In particolare, è lecito chiedersi se esistono una classe di obiettivi di sicurezza che possono essere considerati *atomici*, ossia obiettivi di sicurezza che non possono essere ricondotti in alcun modo ad altri. Come è stato mostrato in questo documento, il requisito di autenticazione nella forma Aliveness si può ricondurre al requisito di autenticità. Non siamo invece in grado di determinare se il requisito di autenticazione nella forma Aliveness possa o meno essere raggiunto da un protocollo che non faccia alcun uso di autenticazione. Il requisito di confidenzialità invece non può essere ricondotto a nessun altro requisito di sicurezza e, come tale, è da considerarsi atomico. La definizione di questa gerarchia di obiettivi di sicurezza potrebbe essere la seguente :

livello 1 Tutti i requisiti di sicurezza che non possono essere ricondotti ad altri.

livello 1+1 I requisiti di sicurezza che possono essere soddisfatti solo da protocolli che soddisfino almeno un requisito di **livello 1**.

Questa classificazione di certo non è ancora matura, ed appare arduo riuscire a definire quali requisiti appartengano al primo livello ed ai successivi. La creazione di una gerarchia di questo genere semplificherebbe notevolmente lo sviluppo di protocolli che si pongano obiettivi *complessi*. Supponiamo ad esempio che si voglia creare un protocollo che soddisfi dei requisiti di non repudiabilità. Supposto che tali requisiti si trovino al secondo livello, chi intendesse affrontare questo problema saprebbe a priori di dover utilizzare protocolli che soddisfino requisiti di livello 1. Il protocollo di

e-mail certificata trattato in questo documento mostra in oltre come la *presentazione* e la formalizzazione stessa del protocollo risultino notevolmente semplificate.

Capitolo 10

Isabelle/Isar files

10.1 Message.thy

```
theory Message = Main
files ("Message_lemmas.ML"):

lemma [simp] : "A Un (B Un A) = B Un A"
by blast

types
  key = nat

consts
  invKey :: "key=>key"

axioms
  invKey [simp] : "invKey (invKey K) = K"

constdefs
  symKeys :: "key set"
  "symKeys == {K. invKey K = K}"

datatype
  agent = Server | Friend nat | Spy
```

```

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

syntax
  "@MTuple"      :: "'a, args] => 'a * 'b"      ("(2{|_,/ _|})")

syntax (xsymbols)
  "@MTuple"      :: "'a, args] => 'a * 'b"      ("(2{|_,/ _|})")

translations
  "{|x, y, z|}" == "{|x, {|y, z|}|}"
  "{|x, y|}"    == "MPair x y"

constdefs

  HPair :: "[msg,msg] => msg"                  ("(4Hash[_] /_)" [0,
1000])
  "Hash[X] Y == {| Hash{|X,Y|}, Y|}"

  keysFor :: "msg set => key set"
  "keysFor H == invKey ' {K.  $\exists X. \text{Crypt } K X \in H\}"

consts parts  :: "msg set => msg set"
inductive "parts H"
  intros
    Inj [intro]:      "X  $\in$  H ==> X  $\in$  parts H"
    Fst:              "{|X,Y|}  $\in$  parts H ==> X  $\in$  parts H"
    Snd:              "{|X,Y|}  $\in$  parts H ==> Y  $\in$  parts H"
    Body:             "Crypt K X  $\in$  parts H ==> X  $\in$  parts H"

lemma parts_mono: "G<=H ==> parts(G) <= parts(H)"$ 
```

```

apply auto
apply (erule parts.induct)
apply (auto dest: Fst Snd Body)
done

```

```

consts analz  :: "msg set => msg set"
inductive "analz H"
  intros
    Inj [intro,simp] : "X ∈ H ==> X ∈ analz H"
    Fst:  "{|X,Y|} ∈ analz H ==> X ∈ analz H"
    Snd:  "{|X,Y|} ∈ analz H ==> Y ∈ analz H"
    Decrypt [dest]:
      "[|Crypt K X ∈ analz H; Key(invKey K): analz H|] ==> X ∈
analz H"

```

```

lemma analz_mono: "G<=H ==> analz(G) <= analz(H)"
apply auto
apply (erule analz.induct)
apply (auto dest: Fst Snd)
done

```

```

consts synth  :: "msg set => msg set"
inductive "synth H"
  intros
    Inj [intro]: "X ∈ H ==> X ∈ synth H"
    Agent [intro]: "Agent agt ∈ synth H"
    Number [intro]: "Number n ∈ synth H"
    Hash [intro]: "X ∈ synth H ==> Hash X ∈ synth H"
    MPair [intro]: "[|X ∈ synth H; Y ∈ synth H|] ==> {|X,Y|} ∈ synth
H"
    Crypt [intro]: "[|X ∈ synth H; Key(K) ∈ H|] ==> Crypt K X ∈ synth
H"

```

```

lemma synth_mono: "G<=H ==> synth(G) <= synth(H)"
apply auto
apply (erule synth.induct)
apply (auto dest: Fst Snd Body)

```

done

```
inductive_cases Nonce_synth [elim!]: "Nonce n ∈ synth H"  
inductive_cases Key_synth [elim!]: "Key K ∈ synth H"  
inductive_cases Hash_synth [elim!]: "Hash X ∈ synth H"  
inductive_cases MPair_synth [elim!]: "{|X,Y|} ∈ synth H"  
inductive_cases Crypt_synth [elim!]: "Crypt K X ∈ synth H"
```

```
use "Message_lemmas.ML"
```

```
lemmas analz_into_parts = analz_subset_parts [THEN subsetD, standard]
```

```
lemma Fake_parts_insert_in_Un:
```

```
  "[|Z ∈ parts (insert X H); X: synth (analz H)|]  
  ==> Z ∈ synth (analz H) ∪ parts H"
```

```
by (blast dest: Fake_parts_insert [THEN subsetD, dest])
```

```
lemma parts_member:"[|x : parts X; y : parts {x}|] ==> y : parts X"
```

```
apply (rotate_tac 1)
```

```
apply (drule UnI1[of _ _ "parts X"])
```

```
apply (rotate_tac 1)
```

```
apply (erule rev_mp)
```

```
apply (subst parts_insert [THEN sym])
```

```
apply (drule parts_cut_eq)
```

```
apply (erule ssubst, simp)
```

```
done
```

```
method_setup spy_analz = {*
```

```
  Method.ctxt_args (fn ctxt =>
```

```
    Method.METHOD (fn facts =>
```

```
      gen_spy_analz_tac (Classical.get_local_claset ctxt,
```

```
        Simplifier.get_local_simpset ctxt) 1))
```

```
*)
```

```
  "for proving the Fake case when analz is involved"
```

```
method_setup atomic_spy_analz = {*
```

```
  Method.ctxt_args (fn ctxt =>
```

```
    Method.METHOD (fn facts =>
```

```
      atomic_spy_analz_tac (Classical.get_local_claset ctxt,
```

```
        Simplifier.get_local_simpset ctxt) 1))
```

```
*)
```

```
  "for debugging spy_analz"
```

```
method_setup Fake_insert_simp = {*
```

```

    Method.ctx_args (fn ctxt =>
      Method.METHOD (fn facts =>
        Fake_insert_simp_tac (Simplifier.get_local_simpset ctxt) 1))
  *}
  "for debugging spy_analz"
end

```

10.2 Message.ML

10.3 EventTraceability.thy

```

theory EventTraceability = Message
files ("EventTraceability_lemmas.ML"):

```

consts

```

  initState :: "agent => msg set"

```

datatype

```

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

```

consts

```

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

```

syntax

```

  spies :: "event list => msg set"

```

translations

```

  "spies"  => "knows Spy"

```

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))"

```

consts

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

primrec

```

used_Nil: "used [] = (UN B. parts (initState B))"
used_Cons: "used (ev # evs) =
  (case ev of
    Says A B X => parts {X} Un (used evs)
  | Gets A X => used evs
  | Notes A X => parts {X} Un (used evs))"

```

lemma Notes_imp_used [rule_format]: "Notes A X : set evs --> X : used evs"

```

apply (induct_tac evs)
apply (auto split: event.split)
done

```

lemma Says_imp_used [rule_format]: "Says A B X : set evs --> X : used evs"

```

apply (induct_tac evs)
apply (auto split: event.split)
done

```

lemma MPair_used [rule_format]:

```

  "MPair X Y : used evs --> X : used evs & Y : used evs"
apply (induct_tac evs)
apply (auto split: event.split)
done

```

```

use "EventTraceability_lemmas.ML"

method_setup analz_mono_contra = {*
  Method.no_args
  (Method.METHOD (fn facts => REPEAT_FIRST analz_mono_contra_tac))
*}
  "for proving theorems of the form  $X \notin \text{analz}(\text{knows Spy evs}) \rightarrow P$ "
end

```

10.4 Traceability.thy

```

theory Traceability = EventTraceability
files ("Traceability_lemmas.ML"):

```

axioms

```

Key_supply_ax : "finite KK ==>  $\exists K. K \notin KK \ \& \ \text{Key } K \notin \text{used evs} \ \& \ K \in \text{symKeys}$ "

```

consts

```

publicKey :: "[bool,agent] => key"

```

syntax

```

pubEK :: "agent => key"
pubSK :: "agent => key"
priEK :: "agent => key"
priSK :: "agent => key"

```

translations

```

"pubEK" == "publicKey False"
"pubSK" == "publicKey True"

```

```

"priEK A" == "invKey (pubEK A)"
"priSK A" == "invKey (pubSK A)"

```

axioms

```
injective_publicKey : "publicKey b A = publicKey b' A' ==> b=b' & A=A'"
```

```
privateKey_neq_publicKey : "invKey (publicKey b A) ≠ publicKey b' A'"
```

consts

```
shrK      :: "agent => key"
```

axioms

```
inj_shrK: "inj shrK"
```

```
sym_shrK: "shrK X ∈ symKeys"
```

primrec

```
initState_Server: "initState Server =  
  insert (Key (priEK Server))(  
  insert (Key (priSK Server))(  
  ((Key ' range pubEK) ∪ (Key ' range pubSK)  
  ∪ (Key ' range shrK))))"
```

```
initState_Friend: "initState (Friend i) =  
  insert (Key (priEK (Friend i)))(  
  insert (Key (priSK (Friend i)))(  
  insert (Key (shrK (Friend i))(  
  ((Key ' range pubEK) ∪ (Key ' range pubSK)))))"
```

```
initState_Spy: "initState Spy =  
  insert (Key (priEK Spy))(  
  insert (Key (priSK Spy))(  
  insert (Key (shrK Spy))(  
  ((Key ' range pubEK) ∪ (Key ' range pubSK)))))"
```

```
use "Traceability_lemmas.ML"
```

```
lemma knows_mono: "x : knows a l ==> x : knows a (b#l)"
```

```
apply(subst knows_Cons)
```

```
apply auto
```

```

apply(case_tac b, auto)
apply(case_tac b, auto)
done

```

```

lemma initState_subset_knows:"initState A <= knows A evs"
apply(induct_tac evs)
apply auto
apply(rule knows_mono)
apply(erule subsetD)
apply assumption
done

```

```

lemma privateKey_into_used : "Key (invKey (publicKey b A)) ∈ used evs"
apply(rule initState_into_used)
apply(rule privateKey_in_initStateCA[THEN parts.Inj])
done

```

```

lemma invKey_K [simp]: "K ∈ symKeys ==> invKey K = K"
by (simp add: symKeys_def)

```

```

lemma Crypt_imp_keysFor : "[|K ∈ symKeys; Crypt K X ∈ H|] ==> K ∈ keysFor H"
apply(drule Crypt_imp_invKey_keysFor, simp)
done

```

```

lemma spies_priK : "Key (invKey (publicKey b Spy)) : spies evs"
apply(induct_tac evs, simp)
apply(rule knows_mono)
apply assumption
done

```

```

lemma Crypt_imp_keysFor:"[|K ∈ symKeys; Crypt K X ∈ H|] ==> K ∈ keysFor H"
apply(blast intro:Crypt_imp_keysFor)
done

```

```

lemma invKey_shrK_iff [iff]:

```

```

    "K ∈ symKeys ==> (Key (invKey K) ∈ X) = (Key K ∈ X)"
  by auto

lemma knows_subset_knows_Cons: "knows A evs <= knows A (e # evs)"
  by (induct e, auto simp: knows_Cons)

lemma knows_Says:"knows A (Says A' B X # evs) =
  (if (A=A' | A=Spy) then insert X (knows A evs) else knows A evs)"
  apply (simp add: knows_Cons)
  done

lemma knows_Gets:"knows A (Gets A' X # evs) =
  (if (A=A' & A~=Spy) then insert X (knows A evs) else knows A evs)"
  apply (simp add: knows_Cons)
  done

lemma knows_Notes:"knows A (Notes A' X # evs) =
  (if (A=A') then insert X (knows A evs) else knows A evs)"
  apply (simp add: knows_Cons)
  done

lemma knowsSSL : " (knows A (Notes S X # Notes R X # evs)) = (
  if (S=A | R=A) then insert X (knows A evs)
  else knows A evs)"
  apply (simp add: knows_Cons)
  done

lemma knowsReliable : "(knows A (Says S R X # Gets R X # evs)) = (
  if (S=A | R=A | A=Spy) then insert X (knows A evs)
  else knows A evs)"
  apply (case_tac "A=Spy")
  apply simp
  apply (simp add: knows_Cons)
  done

declare knows_Spy_Says [simp del]
declare knows_Spy_Gets [simp del]
declare knows_Spy_Notes [simp del]
declare knows_Says [simp]
declare knows_Gets [simp]
declare knows_Notes [simp]

```

```

consts
  allKnows  :: "agent set => event list => msg set"

defs
  allKnows_def : "allKnows S evs == UN x:S. knows x evs"

lemma knows_subset_allKnows : "x:S --> knows x evs <= allKnows S evs"

apply(simp add:allKnows_def)
apply blast
done

lemmas knows_subset_allKnowsD = knows_subset_allKnows[THEN mp]

lemma allKnowsD : "[| A : S; Y : knows A evs |] ==> Y : allKnows S evs"
apply(drule knows_subset_allKnowsD)
apply blast
done

lemma allKnows_subset_allKnows_Says : "
allKnows S evs <= allKnows S (Says A B X # evs)"
apply(simp del: knows_Says add: allKnows_def)
apply(blast intro:knows_subset_knows_Says[THEN subsetD])
done

lemma allKnowsSays_Spy:"Spy : S --> allKnows S (Says A B X # evs) = (insert
X)(allKnows S evs)"
apply(rule impI)
apply(simp add:set_eq_subset)
apply(simp_all add:allKnows_subset_allKnows_Says allKnowsD)
apply(simp add:allKnows_def UNION_def knows_Says)
apply blast
done

lemma allKnowsSays_Agent:"A : S --> allKnows S (Says A B X # evs) = (insert
X)(allKnows S evs)"
apply(rule impI)
apply(simp add:set_eq_subset)
apply(simp_all add: allKnows_subset_allKnows_Says allKnowsD knows_Says)
apply(simp add:allKnows_def UNION_def)
apply blast
done

```

```

lemma allKnowsSays_iff: "allKnows S (Says A B X # evs) = (
  if (Spy : S | A : S) then insert X (allKnows S evs)
  else allKnows S evs)"
apply(simp add:allKnowsSays_Spy allKnowsSays_Agent)

apply(simp del:knows_Says add:set_eq_subset allKnows_subset_allKnows_Says)
apply(simp_all del:knows_Says add:allKnows_def)
apply auto
apply(case_tac "xa=Spy | A = xa")
apply simp_all
apply blast
apply(erule conjE)
apply(drule not_sym)
back
apply simp
apply blast
done

lemma allKnowsSays_Spy:"Spy : S --> allKnows S (Says A B X # evs) = (insert
X)(allKnows S evs)"
apply(rule impI)
apply(simp add:set_eq_subset)
apply(simp_all add:allKnows_subset_allKnows_Says allKnowsD)
apply(simp add:allKnows_def UNION_def)
apply blast
done

lemma allKnowsSays_Agent:"A : S --> allKnows S (Says A B X # evs) = (insert
X)(allKnows S evs)"
apply(rule impI)
apply(simp add:set_eq_subset)
apply(simp_all add: allKnows_subset_allKnows_Says allKnowsD knows_Says)
apply(simp add:allKnows_def UNION_def)
apply blast
done

lemma knowsGets_iff : "knows A (Gets B X # evs) = (
  if (A=Spy | A~=B) then knows A evs
  else (insert X)(knows A evs)
)"
apply(simp add:knows_Cons)
done

```

```

lemma allKnowsGets_Spy : "A=Spy --> allKnows S (Gets A X # evs) = allKnows
S evs"
apply(simp add:allKnows_def)
done

```

```

lemma allKnows_subset_allKnows_Gets:" allKnows S evs <= allKnows S (Gets
A X # evs)"
apply(simp del:knows_Gets add: allKnows_def)
apply(blast intro:knows_subset_knows_Gets[THEN subsetD])
done

```

```

lemma allKnowsGets_Agent : "A : S & A~=Spy --> allKnows S (Gets A X #
evs) = insert X (allKnows S evs)"
apply(rule impI)
apply(erule conjE)
apply(simp add:set_eq_subset)
apply(simp_all add: allKnows_subset_allKnows_Gets allKnowsD knows_Gets)
apply(simp add:allKnows_def UNION_def)
apply blast
done

```

```

lemma allKnowsGets_iff : "allKnows S (Gets A X # evs)= (
  if (A~=Spy & A : S) then insert X (allKnows S evs)
  else allKnows S evs)"
apply(simp add:set_eq_subset allKnows_subset_allKnows_Gets
allKnowsGets_Agent allKnowsGets_Spy)
apply(simp_all del:knows_Gets add:allKnows_def)
apply auto
apply(case_tac "xa = A & xa ~= Spy")
apply blast
apply(simp add:split_if)
apply(erule conjE)
apply blast
done

```

```

lemma knowsNotes_iff : "knows A (Notes B X # evs) =(
  if (A=B) then insert X(knows A evs)
  else knows A evs)"
apply(simp add:knows_Cons)
done

```

```

lemma allKnows_subset_allKnows_Notes : "
  allKnows S evs <= allKnows S (Notes A X # evs)"
apply(simp del: knows_Notes add: allKnows_def)
apply(blast intro:knows_subset_knows_Notes[THEN subsetD])

```

```

done

lemma allKnowsNotes : "A : S -->
  allKnows S (Notes A X # evs) = insert X (allKnows S evs)"
apply(rule impI)
apply(simp add:set_eq_subset)
apply(simp_all add: allKnows_subset_allKnows_Notes allKnowsD knows_Notes)
apply(simp add:allKnows_def UNION_def)
apply blast
done

lemma allKnowsNotes : "allKnows S (Notes A X # evs) =(
  if (A : S) then insert X (allKnows S evs)
  else allKnows S evs)"
apply(simp add:set_eq_subset allKnows_subset_allKnows_Notes
  allKnowsNotes)
apply(simp_all add:allKnows_def)
apply auto
apply(case_tac "xa = A")
apply simp_all
apply blast
done

lemma allKnowsSSL : "allKnows S (Notes A X # Notes B X # evs) =(
  if (A : S | B : S) then insert X(allKnows S evs)
  else allKnows S evs)"
apply(simp add:allKnowsNotes)
done

lemma allKnowsReliable : "allKnows S (Says A B X # Gets B X # evs) =(
  if (A : S | Spy : S | B : S) then insert X(allKnows S evs)
  else allKnows S evs)"
apply(simp add:allKnowsSays_iff allKnowsGets_iff)
done

lemma allKnowsUNIV_Says [simp]:"
  allKnows UNIV (Says A B X # evs) = insert X(allKnows UNIV evs)"
apply(simp add:allKnowsSays_iff)
done

lemma allKnowsUNIV_Notes [simp]:"
  allKnows UNIV (Notes A X # evs) = insert X(allKnows UNIV evs)"
apply(simp add:allKnowsNotes)
done

```

```

lemma allKnowsUNIV_Gets:"A~=Spy -->
  allKnows UNIV (Gets A X # evs) = insert X(allKnows UNIV evs)"
apply(simp add:allKnowsGets_iff)
done

lemma allKnowsUNIV_Gets_iff : "
  allKnows UNIV (Gets A X # evs) =
  (if A=Spy then allKnows UNIV evs
   else insert X(allKnows UNIV evs))"
apply(simp add:allKnowsGets_iff)
done

lemma allKnowsUNIV_Reliable [simp]:"
  allKnows UNIV (Says A B X# Gets B X # evs) = insert X(allKnows UNIV
  evs)"
apply(simp add:allKnowsReliable)
done

lemma allKnows_mono : " S1<=S2 ==> allKnows S1 evs <= allKnows S2 evs"
apply(simp add:allKnows_def)
apply auto
done

lemma allKnows_knows : "knows A evs = allKnows {A} evs"
apply(simp add:allKnows_def)
done

lemma allKnows_Un [simp]:"
  allKnows S1 evs Un allKnows S2 evs = allKnows (S1 Un S2) evs"
apply(simp add:allKnows_def)
apply auto
done

lemma Notes_imp_allKnowsUNIV : "Notes A X : set evs ==> X : allKnows
UNIV evs"
apply(blast dest:Notes_imp_knows intro:allKnowsD)
done

lemma Says_imp_allKnowsUNIV : "Says A B X : set evs ==> X : allKnows
UNIV evs"
apply(blast dest:Says_imp_knows intro:allKnowsD)
done

```

```

lemma singleton_subset_insert : " {x} <= insert x S"
apply simp
done

lemma Spy_to_UNIVD : "x : knows Spy evs ==> x : allKnows (UNIV -{Server})
evs"
apply (simp add:allKnows_def)
apply blast
done

lemma Spy_to_UNIVD_parts : "x : parts(knows Spy evs) ==>
x : parts(allKnows (UNIV -{Server}) evs)"
apply (simp add:allKnows_def)
apply blast
done

lemma Spy_to_UNIVD_analz : "x : parts(knows Spy evs) ==>
x : parts(allKnows (UNIV -{Server}) evs)"
apply (simp add:allKnows_def)
apply blast
done

lemma allKnowsSays_otherThanTTPS : " S~=Spy ==>
allKnows (UNIV-{S,Server}) (Says A B X # evs) =
insert X (allKnows (UNIV-{S,Server}) evs)"
apply (simp add:allKnowsSays_iff)
apply blast
done

lemma allKnowsReliable_otherThanTTPS: " S~=Spy ==>
allKnows (UNIV-{S,Server}) (Says A B X # Gets B X # evs) =
insert X (allKnows (UNIV-{S,Server}) evs)"
apply (simp add:allKnowsReliable)
apply blast
done

lemma Fake_analz_allKnows_insert : "X : synth(analz(knows Spy evs)) ==>
analz(insert X(allKnows S evs)) <= synth(analz(allKnows (insert Spy
S) evs))"
apply auto
apply (drule Fake_analz_insert[of _ _ "allKnows S evs"])
apply (simp add:allKnows_knows)

```

```

apply(blast intro: singleton_subset_insert[THEN allKnows_mono[THEN analz_mono[THEN
synth_mono[THEN subsetD]]]])
done

```

```

lemma Spy_in_set:"S~=Spy ==> insert Spy (UNIV -{S,Server}) = UNIV - {S,
Server}"apply blast
done

```

```

lemma Fake_analz_allKnows_insert_lemma : "
[| S~=Spy;
  X : synth(analz(knows Spy evs))
|] ==>
  analz(insert X(allKnows (UNIV - {S,Server}) evs))
  <= synth(analz(allKnows (UNIV - {S,Server}) evs))"
apply(drule Fake_analz_allKnows_insert)
apply auto
apply(drule subsetD, simp)
apply(drule Spy_in_set, simp)
done

```

```

lemma Crypt_not_init [simp]: "Crypt K X ~: Key ' A"
apply auto
done

```

```

method_setup possibility = {*
  Method.ctx_args (fn ctxt =>
    Method.METHOD (fn facts =>
      gen_possibility_tac (Simplifier.get_local_simpset ctxt)))
*}
  "for proving possibility theorems"

```

```

method_setup analz_freshK = {*
  Method.no_args
  (Method.METHOD
    (fn facts => EVERY [REPEAT_FIRST (resolve_tac [allI, impI]),
      REPEAT_FIRST (rtac analz_image_freshK_lemma),
      ALLGOALS (asm_simp_tac analz_image_freshK_ss)]))
*}

```

"for proving the Session Key Compromise theorem"

end

10.5 CertifiedMail.thy

theory CertifiedMail = Traceability :

syntax

TTP :: agent
RPwd :: "agent => key"
TTPDecKey :: key
TTPEncKey :: key
TTPSigKey :: key
TTPVerKey :: key

translations

"TTP" == "Server"
"RPwd" == "shrK"
"TTPDecKey" == "priEK Server"
"TTPEncKey" == "pubEK Server"
"TTPSigKey" == "priSK Server"
"TTPVerKey" == "pubSK Server"

consts

NoAuth :: nat
TTPAuth :: nat
SAuth :: nat
BothAuth :: nat
ack :: "nat => nat"

syntax

"SSLTransmit" :: "agent => agent => msg => event list => event list"

translations

"SSLTransmit A B X evs" => "Notes A X # Notes B X # evs"

consts certified_mail :: "event list set"

inductive "certified_mail"

intros

```

Nil : "[[] ∈ certified_mail"

Fake : "[| evsf : certified_mail; X : synth(anzl(knows Spy evsf))|] ==>

  Says Spy A X #evsf : certified_mail"

FakeSSL : "[| evsfssl : certified_mail; X : synth(anzl(knows Spy evsfssl))|]
==>  Notes Spy X # Notes A X #evsfssl : certified_mail"

CM1 : "[| evs1 : certified_mail;
  Nonce m ~: used evs1;
  Key k ~: used evs1;
  k : symKeys;
  r = ack q;
  m~=q;
  Nonce q ~: used evs1;

  hs = Hash {| Number cleartext, Nonce q, Nonce r, Crypt k (Nonce m) |};
  S2TTP = Crypt TTPEncKey {| Agent S, Number BothAuth, Key k, Agent R,
hs|}
|] ==>
  Says S R {| Agent TTP, Crypt k (Nonce m), Number BothAuth, Number cleartext,
Nonce q, S2TTP |}#evs1 : certified_mail"

CM2 : "[| evs2 : certified_mail;
  Gets R {|Agent TTP, em', Number BothAuth, Number cleartext', Nonce q',
S2TTP'|} : set evs2;
  r'=ack q;
  hr = Hash {| Number cleartext', Nonce q', Nonce r', em' |}
|] ==>
  Notes R {|S2TTP', Key(RPwD R), hr|} #
  Notes TTP {|S2TTP', Key(RPwD R), hr|} # evs2 : certified_mail"

CM3 : "[| evs3 : certified_mail;
  Notes TTP {|S2TTP'', Key(RPwD R'), hr'|} : set evs3;
  S2TTP''=Crypt TTPEncKey {| Agent S, Number BothAuth, Key k', Agent R',
hs' |};
  hs'=hr';
  pwd''=RPwD R'
|] ==>
  Notes TTP {| Key k', hr'|} #
  Notes R' {| Key k', hr'|} #
  Says TTP S (Crypt TTPSigKey S2TTP'') #
  Gets S (Crypt TTPSigKey S2TTP'') # evs3 : certified_mail"

```

```

reception : "[| evsr : certified_mail; Says A B X : set evsr |]==> Gets
B X#evsr : certified_mail"

lemma Crypt_not_init [simp]: "Crypt K X ~: Key ' A"
apply auto
done

lemma Hash_not_init [simp] : "Hash X ~: Key ' A"
apply auto
done

lemma Gets_imp_Says [dest!]: "[| Gets B X ∈ set evs; evs ∈ certified_mail
|] ==> ∃A. Says A B X ∈ set evs"
apply (erule rev_mp)
apply (erule certified_mail.induct)
apply auto
done

declare sym_shrK [simp]
lemma priK_neq_shrK[iff, simp]: "RPwd A ~= invKey (publicKey b C)"
apply (simp add: symKeys_neq_imp_neq)
done

lemma pubK_neq_shrK[iff, simp]: "RPwd A ~= publicKey b C"
apply (simp add: symKeys_neq_imp_neq)
done

declare priK_neq_shrK[THEN not_sym, simp]
declare pubK_neq_shrK[THEN not_sym, simp]

lemma Fake_analz_eq[simp]: " X : synth(analz H) ==> synth(analz ((insert
X) H)) = synth(analz H) "
apply (drule Fake_analz_insert[of _ _ "H"])
apply (erule rev_mp)
apply simp
apply (subst synth_increasing[THEN Un_absorb2])
apply (rule impI)
apply (drule synth_mono)
apply (rotate_tac 1)
apply (erule rev_mp)
apply (subst synth_idem)
apply (rule impI)
apply auto
apply (rule subset_insertI[THEN analz_mono[THEN synth_mono[THEN subsetD]])

```

```
apply assumption
done
```

```
lemma Gets_imp_parts_knows_Spy : "[| evs : certified_mail; Gets A X :
set evs|] ==> X : parts(spies evs)"
apply (drule Gets_imp_Says, simp)
apply (erule exE)
apply (drule Says_imp_knows_Spy, simp)
apply (drule parts.Inj, simp)
done
```

```
lemma CM2_S2TTP_analz_knows_Spy : "[| evs : certified_mail;
  Gets R {|Agent TTP, em, Number BothAuth, Number cleartext, Nonce q,
S2TTP|} :
  set evs |] ==>
  S2TTP : analz(knows Spy evs)"
apply (drule Gets_imp_Says, simp)
apply (erule exE)
apply (drule Says_imp_knows_Spy)
apply (drule analz.Inj)
apply (drule analz.Snd)+
apply assumption
done
```

```
lemmas CM2_S2TTP_parts_knows_Spy = CM2_S2TTP_analz_knows_Spy[THEN analz_subset_parts[THEN
subsetD]]
```

```
lemma CM2_em_analz_knows_Spy : "[| evs : certified_mail;
  Gets R {|Agent TTP, em, Number BothAuth, Number cleartext, Nonce q,
S2TTP|} :
  set evs |] ==>
  em : analz(knows Spy evs)"
apply (drule Gets_imp_Says, simp)
apply (erule exE)
apply (drule Says_imp_knows_Spy)
apply (drule analz.Inj)
apply (drule analz.Snd)
apply (drule analz.Fst)
apply assumption
done
```

```
lemmas CM2_em_parts_knows_Spy = CM2_em_analz_knows_Spy[THEN analz_subset_parts[THEN
subsetD]]
```

```

lemma CM3_S2TTP_parts_knows_Spy : "[| evs : certified_mail;
  Notes R {|S2TTP, pwd, hr|} : set evs
  |]==>
  S2TTP : parts(spies evs)"
apply(rotate_tac 1)
apply(erule rev_mp)
apply(erule certified_mail.induct)
prefer 3
apply(rule impI)
apply(simp)
apply(case_tac "{|S2TTP, pwd, hr|}=X")
apply(drule sym)
prefer 6
apply(drule Gets_imp_parts_knows_Spy, simp)
apply(drule parts.Snd)+
apply(rule impI)
apply(simp_all)
apply(blast dest:parts.Snd intro:parts_insertI dest:parts.Body)+
done

lemma CM3_k_parts_knows_Spy : "[| evs : certified_mail;
  Notes TTP {|Crypt TPEncKey {| Agent S, Number BothAuth, Key k, Agent
R, hs|}, Key (RPwd R), hs|} : set evs
  |]==>
  Key k: parts(spies evs)"
apply(drule CM3_S2TTP_parts_knows_Spy, simp)
apply(blast dest:parts.Body dest:parts.Snd)
done

lemma CM3_hs_parts_knows_Spy : "[| evs : certified_mail;
  Notes TTP {|Crypt TPEncKey {| Agent S, Number BothAuth, Key k, Agent
R, hs|}, Key (RPwd R), hs|} : set evs
  |]==>
  hs : parts(spies evs)"
apply(drule CM3_S2TTP_parts_knows_Spy, simp)
apply(blast dest:parts.Body dest:parts.Snd)
done

```

```

(*****
  LEMMA 1
  *****)

lemma Spy_dont_know_private_keys:"evs : certified_mail ==> A~=Spy -->
Key (invKey (publicKey b A)) ~: parts(spies evs)"
apply(erule certified_mail.induct)
prefer 5
apply(drule CM2_S2TTP_parts_knows_Spy, simp)
prefer 6
apply(frule CM3_hs_parts_knows_Spy, simp)
apply(drule CM3_k_parts_knows_Spy, simp)
apply simp_all
apply(blast dest:Fake_parts_insert[THEN subsetD]
        dest:analz_subset_parts[THEN contra_subsetD]
        intro:privateKey_in_initStateCA[THEN parts.Inj
        [THEN initState_into_used]])+
done

lemma Spy_dont_know_RPw:"evs : certified_mail ==> A~=Spy -->
  Key (RPw A) ~: parts(spies evs)"
apply(erule certified_mail.induct)
prefer 5
apply(drule CM2_S2TTP_parts_knows_Spy, simp)
prefer 6
apply(frule CM3_hs_parts_knows_Spy, simp)
apply(drule CM3_k_parts_knows_Spy, simp)
apply simp_all
apply(blast dest:Fake_parts_insert[THEN subsetD]
        dest:analz_subset_parts[THEN contra_subsetD]
        intro:shrK_in_initState[THEN parts.Inj
        [THEN initState_into_used]])+
done
(*****
  LEMMA 2
  *****)

lemma Spy_doesnt_know_RPw_eq [simp]:"[|
  Key (RPw A) : parts(knows Spy evs);
  evs : certified_mail|] ==>
  (A=Spy)"
apply(case_tac "A=Spy")
apply simp
apply(drule Spy_dont_know_RPw[THEN mp], simp)

```

```
apply simp
done
```

```
(*****
  Teorema 1
  *****)
```

```
lemma CertAutenticity : "[| Crypt TTPSigKey X : parts(spies evs);
  evs : certified_mail
|] ==>
  ∃ A.(Says TTP A (Crypt TTPSigKey X) : set evs)"
apply(erule rev_mp)
apply(erule certified_mail.induct)
prefer 5
apply(drule Gets_imp_parts_knows_Spy, simp)
apply(drule parts.Snd)+
prefer 6
apply(drule CM3_hs_parts_knows_Spy, simp)
apply simp_all
apply(blast dest:Spy_dont_know_private_keys
  dest:Fake_parts_insert[THEN subsetD]
  dest:analz_subset_parts[THEN contra_subsetD]
  intro:privateKey_in_initStateCA[THEN parts.Inj[THEN initState_into_used]])+
done
```

```
lemma Gets_imp_knows : "[| evs : certified_mail;Gets A X : set evs |]
==> X : knows A evs"
apply(case_tac "A=Spy")
apply(drule Gets_imp_Says, simp)
apply(erule exE)
apply(drule Says_imp_knows_Spy, simp)
apply(rule Gets_imp_knows_agents, simp_all)
done
```

```
lemma knows_SSL_sender [simp]: " knows A (Notes A X # Notes B X # evs)
= insert X (knows A evs)"
apply(simp add: knows_Cons)
done
```

```
lemma knows_SSL_receiver [simp]: " knows B (Notes A X # Notes B X # evs)
= insert X (knows B evs)"
apply(simp add: knows_Cons)
done
```

```

lemma knows_SSL : "R=A | R=B ==> knows R (Notes A X # Notes B X # evs)
= insert X (knows R evs)"
apply(erule disjE)
apply simp_all
done

lemma Hash_synth_insert : "(Hash X : synth H) = (X : synth H | Hash X
: H)"
apply auto
done

lemma Spy_dont_know_TTPDecKey [simp]: "evs : certified_mail ==> Key TTPDecKey
~: parts(knows Spy evs)"
apply(drule Spy_dont_know_private_keys[of _ "TTP" "False"])
apply simp
done

lemma Spy_dont_know_TTPDecKey_analz[simp]:"evs : certified_mail ==>
Key TTPDecKey ~: analz(knows Spy evs)"
apply(rule analz_subset_parts[THEN contra_subsetD], simp)
done

(*****
TEOREMA 4
*****)

lemma RequestAuthenticity[dest]:"[| evs : certified_mail;
Notes TTP {|
Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R, hr|},
Key (RPwd R),
hr|} : set evs |] ==>
Notes R {|
Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R, hr|},

Key(RPwd R),
hr|} : set evs"
apply(rotate_tac 1)
apply(erule rev_mp)
apply(erule certified_mail.induct)
apply simp_all
apply(rule impI)
apply(erule conjE)

```

```

apply (drule sym)
back
apply simp
apply (erule conjE)+
apply (drule analz_subset_parts [THEN subsetD])
apply (drule Spy_dont_know_RPwd_eq, simp)
apply simp_all
apply blast
done

(*****
  LEMMA 3
  *****)

lemma Spy_dont_know_hr : "[| evs : certified_mail ;
  Hash {/Number cleartext, Nonce q, Nonce r, em|} : parts (knows Spy evs) |]
  ==> em : synth (analz (knows Spy evs))"

apply (rotate_tac 1)
apply (erule rev_mp)
apply (erule certified_mail.induct)
apply simp
apply spy_analz
apply spy_analz

apply (rule impI)
apply simp
apply (erule disjE)
apply (blast intro: subset_insertI [THEN synth_mono [THEN subsetD]]
  intro: subset_insertI [THEN analz_mono [THEN synth_mono [THEN subsetD]]])

apply simp
apply (rule conjI)
apply (rule impI)
apply (rule subset_insertI [THEN synth_mono [THEN subsetD]])+
apply assumption
apply (rule impI)
apply (rule subset_insertI [THEN synth_mono [THEN subsetD]])+
apply assumption

apply (frule CM2_S2TTP_parts_knows_Spy, simp)
apply (frule CM2_em_analz_knows_Spy, simp)
apply (drule synth.Inj)
back

```

```

apply simp
apply (blast intro:subset_insertI[THEN synth_mono[THEN subsetD]]
      intro:subset_insertI[THEN analz_mono[THEN synth_mono[THEN subsetD]]])

apply (frule CM3_hs_parts_knows_Spy, simp)
apply simp
apply (blast intro:subset_insertI[THEN synth_mono[THEN subsetD]]
      intro:subset_insertI[THEN analz_mono[THEN synth_mono[THEN subsetD]]])
apply simp
done

(*****
  TEOREMA 5
  *****)

lemma R_knows_encrypted_message: "[|
  Notes R' {|
    Crypt TPEncKey {| Agent S, Number authoption, Key k, Agent R,
      Hash {| Number cleartext, Nonce q, Nonce r, em |}
    |},
    Key (RPwd R),
    Hash {| Number cleartext, Nonce q, Nonce r, em |}|} : set evs;
  evs : certified_mail |]==>
  em : synth(analz( knows R evs))"

apply (erule rev_mp)
apply (erule certified_mail.induct)
prefer 3
apply (rule impI)
apply (rule knows_subset_knows_Notes[THEN analz_mono[THEN synth_mono[THEN
subsetD]]])+)
apply (case_tac "X={|Crypt TPEncKey
  {|Agent S, Number authoption, Key k, Agent R,
    Hash {|Number cleartext, Nonce q, Nonce r, em|}|},
  Key (RPwd R), Hash {|Number cleartext, Nonce q, Nonce r, em|}|}")
apply (rotate_tac 2)
apply (erule rev_mp)
apply (erule ssubst)
apply simp
apply (rule impI)

apply (drule conjunct2)
apply (frule conjunct1)
apply (drule analz_subset_parts[THEN subsetD])

```

```

apply(drule Spy_dont_know_RPw_eq, simp)
apply simp

apply(drule conjunct2)
apply(simp add:Hash_synth_insert)
apply(erule disjE)
back
apply simp
apply(drule analz_subset_parts[THEN subsetD])
apply(drule Spy_dont_know_hr, simp)
apply simp

apply(drule not_sym)
apply simp

apply simp

prefer 3
apply(drule Gets_imp_knows, simp)
apply(drule analz.Inj)
apply(drule analz.Snd)
apply(drule analz.Fst)
apply(rule impI)
apply(simp_all del:knows_Notes knows_Says knows_Gets
  add:knows_Spy_Notes knows_Spy_Says knows_Spy_Gets)

apply(blast intro: knows_subset_knows_Notes[THEN analz_mono[THEN synth_mono[THEN
subsetD]]]
  intro:knows_subset_knows_Says[THEN analz_mono[THEN synth_mono[THEN subsetD]]]
  intro:knows_subset_knows_Gets[THEN analz_mono[THEN synth_mono[THEN subsetD]])+
done

(*****
  TEOREMA 6
  *****)

lemma R_knows_encrypted_message1 : "[|
  Crypt TTPSigKey S2TTP : parts(spies evs);
  S2TTP=Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,
hs|};
  hs = Hash {| Number cleartext, Nonce q, Nonce r, em |};
  evs : certified_mail |]
=> em : synth(analz(knows R evs))"

```

```

apply(drule CertAutenticity, simp)
apply(erule exE)
apply(rotate_tac 3)
apply(erule rev_mp)
apply(erule ssubst)+
apply(erule certified_mail.induct)
prefer 6
apply(rule impI)
apply(simp del:knows_Notes knows_Says knows_Gets
  add:knows_Spy_Notes knows_Spy_Says knows_Spy_Gets)
apply(erule disjE)
apply(drule conjunct2)
apply(drule conjunct2)
apply(drule conjunct2)
apply(drule conjunct2)
apply(frule conjunct1)
apply(drule conjunct2)
apply(drule sym)
back
back
back
back
apply(rule knows_subset_knows_Notes[THEN analz_mono[THEN synth_mono[THEN
subsetD]]])+
apply(rule knows_subset_knows_Says[THEN analz_mono[THEN synth_mono[THEN
subsetD]]])+
apply(rule knows_subset_knows_Gets[THEN analz_mono[THEN synth_mono[THEN
subsetD]]])+
apply(simp_all del:knows_Notes knows_Says knows_Gets
  add:knows_Spy_Notes knows_Spy_Says knows_Spy_Gets)
apply(blast intro:knows_subset_knows_Notes[THEN analz_mono[THEN synth_mono[THEN
subsetD]]]
  intro:knows_subset_knows_Says[THEN analz_mono[THEN synth_mono[THEN subsetD]]]
  intro:knows_subset_knows_Gets[THEN analz_mono[THEN synth_mono[THEN subsetD]]]
  dest:R_knows_encrypted_message)+
done

```

```

(*****
  TEOREMA 2
  *****)

```

```

lemma TTP_sends_both : "[| evs : certified_mail;
  S2TTP=Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,
hr|}];

```

```

    Crypt TTPSigKey S2TTP : parts(spies evs) []
      ==> Notes R {|Key k, hr|} : set evs"
  apply(drule CertAutenticity, simp)
  apply(erule exE)
  apply(rotate_tac 2)
  apply(erule rev_mp)
  apply(erule ssubst)
  apply(erule certified_mail.induct)
  apply simp_all
done
(*****
  TEOREMA 3
  *****)

lemma R_knows_key : "[| evs : certified_mail;
  S2TTP=Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,
  hr|};
  Crypt TTPSigKey S2TTP : parts(spies evs) []
  ==> Key k : analz(knows R evs)"
  apply simp
  apply(drule TTP_sends_both)
  apply simp
  apply simp
  apply(drule Notes_imp_knows[THEN analz.Inj[THEN analz.Fst]])
  apply assumption
done

lemma Fake_parts_insert2: "X : synth(analz (H)) ==>
  parts(insert X G) <= synth(analz H) Un parts(H) Un parts(G)"
  apply(drule parts_insert_subset_Un)
  apply simp
  apply auto
done

lemma aux : " x : H ==> {x} <= H"
  apply simp
done
(*****
  LEMMA 1a
  *****)

lemma Agent_doesnt_knows_TTPSigKey : "evs : certified_mail ==>

```

```

    ALL R . R ~= TTP --> Key TTPSigKey ~: parts(knows R evs)"
apply(erule certified_mail.induct)
apply(rule allI)
apply(case_tac "R")
apply simp
apply simp
apply simp
apply(simp add:all_simps)
apply spy_analz
apply(simp add:all_simps)
apply(blast dest:Spy_dont_know_private_keys
           dest:Fake_parts_insert2[THEN subsetD]
           dest:analz_subset_parts[THEN contra_subsetD]
           intro:privateKey_in_initStateCA[THEN parts.Inj[THEN initState_into_used]])
apply(simp add:all_simps)
apply(blast dest:Spy_dont_know_private_keys
           dest:Fake_parts_insert2[THEN subsetD]
           dest:analz_subset_parts[THEN contra_subsetD]
           intro:privateKey_in_initStateCA[THEN parts.Inj[THEN initState_into_used]])
apply(drule Gets_imp_knows, simp)
apply(drule parts.Inj)
apply(drule parts.Snd)+
apply simp
apply(frule CM3_hs_parts_knows_Spy, simp)
apply(frule CM3_k_parts_knows_Spy, simp)
apply(frule RequestAutenticity, simp)
apply(frule Notes_imp_knows[THEN parts.Inj])
back
apply(frule parts.Fst)
apply(drule parts.Snd)+
apply(simp del:knows_Says knows_Gets knows_Notes add:knowsSSL knowsReliable
       all_simps)
apply(rule allI)
apply(rule impI)
apply(rule conjI)
apply(rule impI)
apply(rule conjI)
apply(rule impI)+
apply(rule conjI)
apply blast
apply simp
apply blast
apply(rule impI)+
apply(rule conjI)
apply blast

```

```

apply simp
apply(subst parts_insert)
apply simp
apply(drule aux[THEN parts_mono])
apply(rule contra_subsetD, simp)
apply blast
apply(drule aux[THEN parts_mono])
apply(rule conjI)
apply blast
apply(rule impI)+
apply(rule conjI)
apply blast
apply(subst parts_insert, simp)
apply(rule contra_subsetD, simp)
apply blast

apply(drule Says_imp_knows_Spy[THEN parts.Inj])
apply simp
apply(subst parts_insert)
apply simp
apply(blast dest:aux[THEN parts_mono])
done

lemma Agent_doesnt_know_TTPSigKeyD : "[| evs : certified_mail;
  R ~= TTP |] ==> Key TTPSigKey ~: parts(knows R evs)"
apply(rotate_tac 1)
apply(erule rev_mp)
apply(rule_tac x="R" in spec)
apply(rule Agent_doesnt_knows_TTPSigKey)
apply assumption
done

lemma parts_insert_imp: "[| x : parts(A) --> x : parts(B); x : parts(insert
Y A) |] ==>
  x : parts(insert Y B)"
apply(rotate_tac 1)
apply(erule rev_mp)
apply(subst parts_insert)
apply(subst parts_insert)
back
apply simp
done

```

```

(*****
  LEMMA 4
  *****)

lemma CertInTr:"[| evs : certified_mail;
  R~=TTP;
  Crypt TTPSigKey X : synth( analz( knows R evs)) |] ==>
  Crypt TTPSigKey X : parts(knows Spy evs)"
apply(drule analz_subset_parts[THEN synth_mono[THEN subsetD]])
apply(frule Agent_doesnt_know_TTPSigKeyD[THEN Crypt_synth_eq])
apply simp
apply(rotate_tac 2)
apply(erule rev_mp)
apply(erule ssubst)
apply(erule certified_mail.induct)
apply(case_tac "R")
prefer 7
apply(drule Gets_imp_knows[THEN parts.Inj], simp)
apply(drule parts.Snd)+
apply simp
prefer 8
apply(frule RequestAutenticity, simp)
apply(frule Notes_imp_knows[THEN parts.Inj])
back
apply(frule parts.Fst)
apply(drule parts.Snd)+
prefer 9
apply(drule Says_imp_knows_Spy[THEN parts.Inj])
apply simp_all
apply(blast dest:parts_cut_eq intro:parts_insert_imp parts_insertI)+
done
(*****
  TEOREMA 7a
  *****)

lemma Judge_Says_received : "[| evs : certified_mail;
  k : symKeys;
  S2TTP=Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,
hs|};
  hs = Hash {| Number cleartext, Nonce q, Nonce r, Crypt k (Nonce m) |};
  Crypt TTPSigKey S2TTP : parts(spies evs) |]
  ==> Nonce m : analz(knows R evs)"
apply simp
apply(frule R_knows_encrypted_message1)
apply blast

```

```

apply blast
apply blast
apply (drule R_knows_key)
apply simp
apply simp
apply (frule Crypt_synth_analz)
apply simp
apply blast
done
(*****
  TEOREMA 7b
  *****)

lemma Judge_Says_received2 : "[| evs : certified_mail;
  k : symKeys;
  S2TTP=Crypt TTPEncKey {| Agent S, Number authoption, Key k, Agent R,
  hs|};
  hs = Hash {| Number cleartext, Nonce q, Nonce r, Crypt k (Nonce m) |};
  R~=TTP;
  Crypt TTPSigKey S2TTP : synth(analz(knows R evs)) |]
  ==> Nonce m : analz(knows R evs)"
apply (frule CertInTr)
apply simp
apply simp
apply (rule Judge_Says_received)
apply simp_all
apply blast
done

end

```

Bibliografia

- [1] M. Abadi and B. Blanchet.
Computer-assisted verification of a protocol for certified email.
In R. Cousot, editor, *Static Analysis, 10th International Symposium (SAS'03)*, volume 2694 of *Lecture Notes in Comp. Sci.*, pages 316–335. Springer Verlag, 2003.
- [2] M. Abadi, N. Glew, B. Horne, and B. Pinkas.
Certified email with a light on-line trusted third party: Design and implementation.
In *Proceedings of the 11th International Conference on World Wide Web (WWW-02)*. ACM Press and Addison Wesley, 2002.
- [3] N. Asokan, V. Shoup, and M. Waidner.
Asynchronous protocols for optimistic fair exchange.
In *Proc. of the 17th IEEE Sym. on Sec. and Privacy*. IEEE Comp. Society Press, 1998.
- [4] T. Aura.
Distributed access-rights management with delegation certificates.
In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues with Distributed Mobile Objects*, volume 1603 of *Lecture Notes in Comp. Sci.*, pages 211–235. Springer-Verlag, 1999.

- [5] G. Bella.
Inductive verification of smart card protocols.
J. of Comp. Sec., 11(1):87–132, 2003.
- [6] G. Bella, F. Massacci, and L. C. Paulson.
Verifying the SET registration protocols.
IEEE J. of Selected Areas in Communications, 21(1):77–87,
2003.
- [7] G. Bella and L. C. Paulson.
Mechanical proofs about a non-repudiation protocol.
In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in
Higher Order Logics: TPHOLs 2001*, volume 2152 of *Lecture Notes in
Comp. Sci.*, pages 91–104. Springer, 2001.
- [8] B. Blanchet.
An efficient cryptographic protocol verifier based on prolog rules.
In *Proc. of the 14th IEEE Comp. Sec. Found. Workshop*. IEEE
Comp. Society Press, 1998.
- [9] M. Burrows, M. Abadi, and R. M. Needham.
A logic of authentication.
Proceedings of the Royal Society of London, 426:233–271, 1989.
- [10] J. Clark and J. Jacob.
A survey of authentication protocol literature: Version 1.0.
Technical report, University of York, Department of Computer Science,
November 1997.
Available on the web at <http://www-users.cs.york.ac.uk/~jac/>.
A complete specification of the Clark-Jacob library in CAPSL is
available at
<http://www.cs.sri.com/~millen/capsl/>.

- [11] E. Cohen.
TAPS: A first-order verifier for cryptographic protocols.
In *Proc. of the 13th IEEE Comp. Sec. Found. Workshop*, pages
144–158. IEEE Comp. Society Press, 2000.
- [12] F. J. T. Fábrega, 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.
- [13] G. Lowe.
Breaking and fixing the Needham-Schroeder public-key protocol
using CSP and FDR.
In T. Margaria and B. Steffen, editors, *Tools and Algorithms for
the Construction and Analysis of Systems: second international
workshop*,
TACAS '96, volume 1055 of *Lecture Notes in Comp. Sci.*, pages
147–166. Springer, 1996.
- [14] L. C. Paulson.
The inductive approach to verifying cryptographic protocols.
J. of Comp. Sec., 6:85–128, 1998.
- [15] T. Nipkow, L. C. Paulson and M. Wenzel.
A Proof Assistant for High-Order Logic
2002.
- [16] L. C. Paulson
Introduction to Isabelle.
2002.
- [17] L. C. Paulson.
Inductive analysis of the internet protocol TLS.
ACM Trans. on Inform. and Sys. Sec., 2(3):332–351, 1999.

- [18] 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.
- [19] V. Shmatikov and J. C. Mitchell.
Analysis of a fair exchange protocol.
In *Network and Distributed System Security Symposium (NDSS-00)*,
2000.
- [20] J. Zhou and D. Gollmann.
A fair non-repudiation protocol.
In *Symposium on Security and Privacy*. IEEE Computer Society,
1996.
- [21] G.Bella, C.Longo and L.C. Paulson
Verifying Second Level Security Protocols
In *International Workshop on Security Protocols*. Sidney Sussex
College, Cambridge, England
2003.