IPsec e TLS a confronto: funzioni, prestazioni ed estensioni

di Davide Cerri

Ottobre 2001

Il materiale contenuto in questo documento è interamente tratto dalla tesi dal titolo *I protocolli IPsec e TLS per la sicurezza in rete: funzioni, prestazioni ed estensioni*, con la quale il 16 ottobre 2001 mi sono laureato in ingegneria delle telecomunicazioni presso il Politecnico di Milano. Le differenze rispetto alla tesi riguardano l'impaginazione, l'organizzazione in capitoli, e la mancanza qui di alcuni paragrafi che non ho ritenuto interessante riportare al di fuori di quell'ambito. Ringrazio ancora una volta tutte le persone che mi hanno aiutato nella stesura della tesi di laurea.

Indice

1 Introduzione: problematiche di sicurezza su Internet **5**

2 IPsec **7**
2.1 Descrizione 7
2.1.1 Architettura di IPsec 7
2.1.2 AH (Authentication Header) 8
2.1.3 ESP (Encapsulating Security Payload) 9
2.1.4 IKE (Internet Key Exchange) 10
2.1.5 Elaborazione dei pacchetti IPsec 13
2.2 Utilizzi 14
2.2.1 Reti private virtuali (VPN) 14
2.2.2 “Road warrior” 15
2.3 Implementazioni 16
2.3.1 Linux FreeS/WAN 16
2.3.2 Altre implementazioni 17

3 TLS **18**
3.1 Descrizione 18
3.1.1 TLS Record Protocol 18
3.1.2 TLS Handshake Protocol 19
3.1.2.1 Change cipher spec 20
3.1.2.2 Alert 20
3.1.2.3 Handshake 20
3.1.3 Differenze tra TLS 1.0 e SSL 3.0 22
3.2 Utilizzi 23
3.3 Implementazioni 23

4 Confronto funzionale e di impatto **24**
4.1 Differenze di funzionalità 24
4.2 Differenze di impatto 26

5 L’instaurazione della connessione IPsec **29**
5.1 Il DNS sicuro: DNSSEC 29
5.2 Handshake in sistemi centralizzati: il protocollo KINK 30
5.3 Connessioni gateway–to–gateway: la opportunistic encryption 30
5.4 Il protocollo HIP 32
5.4.1 Architettura e concetti generali 32
5.4.2 Formato e sequenza dei pacchetti 35

6 Valutazione delle prestazioni **38**
6.1 Architettura per le misurazioni . . . 38
6.2 Strumenti di misura . . . 39
6.3 Prestazioni di IPsec . . . 40
6.4 Prestazioni di TLS . . . 42
6.5 Sovrapposizione tra TLS e IPsec . . . 43
6.6 L'impatto della frammentazione sulle prestazioni di IPsec . . . 44
6.6.1 Analisi teorica . . . 44
6.6.2 Risultati sperimentali . . . 47
6.7 Tempi di handshake . . . 49
6.7.1 IKE . . . 51
6.7.2 IKE "opportunistic" . . . 52
6.7.3 TLS . . . 52

7 Conclusioni **54**

A Note di crittografia **58**
A.1 Crittografia simmetrica e asimmetrica . . . 58
A.2 Cifrari a blocco e loro modalità operative . . . 59
A.3 Diffie–Hellman . . . 60
A.4 RSA . . . 62
A.5 Funzioni di hash e MAC . . . 62

B Software utilizzato **64**

Elenco degli acronimi **66**

Elenco delle figure **69**

Elenco delle tabelle **71**

Bibliografia **72**

Indice analitico **75**

1 Introduzione: problematiche di sicurezza su Internet

Quando TCP/IP fu progettato, quella che diventerà poi la rete Internet collegava tra loro università e laboratori di ricerca, ed era uno strumento per la condivisione delle risorse e delle informazioni da parte della comunità scientifica: gli utenti della rete erano dunque una comunità ristretta e fidata, e per questo non si fece particolare attenzione alle problematiche relative alla sicurezza. Oggi, con l'esplosione di massa di Internet, la situazione è molto diversa, e il tema della sicurezza in rete assume una grande importanza.

Bisogna innanzitutto dire che definire una rete come "sicura" oppure no non ha molto senso. In primo luogo perché una rete completamente sicura non esiste (e forse non esisterà mai), per cui bisogna commisurare il grado di sicurezza offerto con quello richiesto dalla particolare applicazione. In secondo luogo, è importante notare come il termine "sicurezza" comprenda aspetti e problemi diversi, per risolvere i quali si usano approcci differenti. I principali servizi di sicurezza che si possono voler garantire sono i seguenti:

Autenticazione: garantire l'identità degli interlocutori.

Riservatezza: impedire letture non autorizzate, garantendo la segretezza del contenuto della comunicazione (detto anche servizio di *confidenzialità*).

Integrità: garantire che dati e informazioni non subiscano modifiche non autorizzate, in particolare durante il transito dal mittente al destinatario.

Non ripudio: impedire che le parti neghino di aver inviato (o ricevuto) dati che hanno effettivamente inviato (o ricevuto).

Autorizzazione: permettere l'accesso a determinate risorse o servizi solo a chi è autorizzato a farlo (basandosi su un servizio di autenticazione).

Disponibilità: garantire l'operatività di un servizio e la sua fruibilità da parte degli utenti autorizzati, evitando che azioni di disturbo possano compromettere la disponibilità del sistema.

Gli attacchi alla sicurezza di un sistema possono essere dunque molteplici, e colpire l'uno o l'altro di questi aspetti. Un intruso può ad esempio intercettare dei dati (colpendo quindi la riservatezza), modificare delle informazioni in transito (integrità), sostituirsi ad uno degli interlocutori (autenticazione), rendere indisponibile un servizio tramite un attacco di tipo *Denial of Service (DoS)*. Un attacco classico è il cosiddetto *man–in–the–middle* (MITM, ovvero "uomo in mezzo"), in cui un intruso si inserisce nella comunicazione impersonando ciascun interlocutore agli occhi dell'altro, e può così leggere tutti i dati scambiati, modificarli, inserirne altri.

In questa sede ci si occuperà della sicurezza delle comunicazioni su Internet, e in generale di come rendere sicura (secondo vari aspetti) la trasmissione di informazioni in rete. Anche se è ovvio, è

importante notare che la sicurezza in rete termina dove inizia la sicurezza degli host: qui ci si occupa essenzialmente di proteggere l'informazione durante il transito in rete, ma è abbastanza inutile approntare un sistema che permetta di comunicare in maniera sicura se poi le macchine e i dati ivi immagazzinati sono accessibili a chiunque.

Come detto, gli approcci possibili per fornire servizi di sicurezza sono molti, e la scelta dell'uno o dell'altro dipende dalle funzionalità richieste e dall'applicazione in questione. In particolare si possono evidenziare tre approcci, illustrati in figura 1.1, che corrispondono a diverse collocazioni delle funzionalità di sicurezza all'interno dello stack TCP/IP. Nel caso (a) la sicurezza è inserita a livello rete, ovvero all'interno del protocollo IP: è questo il caso di *IPsec*, che offre il vantaggio di una totale trasparenza rispetto ai livelli superiori[1]. Un'altra soluzione abbastanza generale è quella di *TLS*[2], mostrata in (b): in questo caso le funzionalità di sicurezza sono inserite al di sopra del protocollo di trasporto, ovvero a livello sessione secondo la nomenclatura OSI. Infine, come mostrato nel caso (c), è possibile includere i servizi di sicurezza a livello applicativo, con soluzioni specifiche e dedicate alla singola applicazione: esempi di questo tipo sono *SET* per le transazioni sicure e *PGP* per la posta elettronica. Nel seguito ci si concentrerà sui primi due casi perché costituiscono soluzioni più generali, analizzando quindi in particolare la suite di protocolli IPsec e il protocollo TLS. Per una trattazione generale delle tematiche associate alla sicurezza delle reti si può vedere [1], che tratta anche con un certo dettaglio IPsec e TLS.

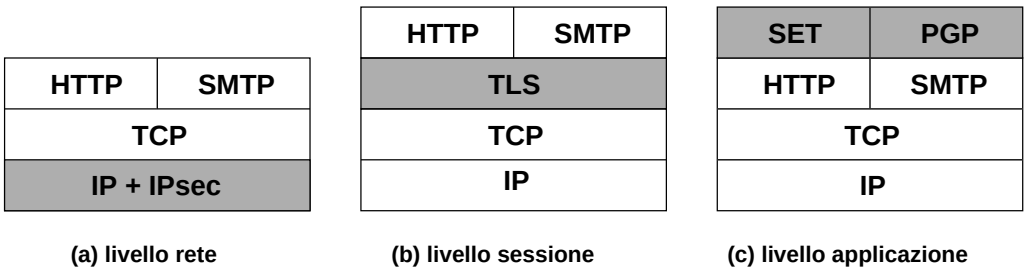

Figura 1.1: Diversi posizionamenti delle funzioni di sicurezza nello stack TCP/IP.

[1] In figura è indicato TCP, ma IPsec può essere utilizzato da qualunque protocollo di livello superiore.
[2] TLS è un'evoluzione del protocollo SSL standardizzata dalla IETF.

2 IPsec

2.1 Descrizione

IPsec è un insieme di protocolli che specificano un'architettura di sicurezza a livello del protocollo IP; è stato progettato insieme con IPv6, di cui è parte integrante, ma può opzionalmente essere utilizzato anche con IPv4 come estensione[1]. IPsec può essere implementato nei singoli host oppure in sistemi intermedi detti *security gateway*, e fornisce essenzialmente servizi di autenticazione, integrità e riservatezza; dato che tali servizi sono forniti a livello IP possono essere utilizzati da qualunque protocollo di livello superiore in modo totalmente trasparente.

2.1.1 Architettura di IPsec

La struttura di IPsec, specificata in [2], è piuttosto complessa in quanto IPsec non è un singolo protocollo ma piuttosto un'architettura di sicurezza a livello IP, composta da vari protocolli e da altri elementi. I protocolli principali che costituiscono IPsec sono tre: *AH* (Authentication Header) che fornisce servizi di autenticazione e integrità, *ESP* (Encapsulating Security Payload) che fornisce servizi di riservatezza, autenticazione e integrità, e *IKE*[2] (Internet Key Exchange) che gestisce lo scambio delle chiavi[3]. AH ed ESP non si preoccupano dello scambio delle chiavi e presumono che i due interlocutori si siano già accordati creando tra loro una *security association* (SA), ovvero un "contratto" che specifica quali meccanismi di sicurezza utilizzare e con quali chiavi. Il compito di negoziare e gestire le security association secondo delle politiche definite localmente è affidato appunto a IKE[4]. Sia AH che ESP possono essere utilizzati in modalità *trasporto* oppure in modalità *tunnel*: nel primo caso (non utilizzabile tra security gateway) si aggiungono gli header dei protocolli utilizzati (AH e/o ESP) tra l'header IP e l'header del protocollo di trasporto (TCP o UDP), mentre nel secondo il pacchetto IP originario viene interamente incapsulato in un nuovo pacchetto IP.

Un concetto fondamentale nell'ambito di IPsec, come accennato poco fa, è quello di security association. Una SA è una "connessione" tra due parti che comunicano tramite IPsec, e specifica quali meccanismi di sicurezza vengono utilizzati, quali algoritmi e quali chiavi, per proteggere il traffico che vi fluisce. Le SA sono unidirezionali, per cui sono necessarie due SA per permettere a due host di comunicare tra loro, inoltre una SA può essere associata ad AH o ad ESP, ma non ad entrambi. Tutte

[1] Nel seguito ci si riferirà generalmente all'uso di IPsec in ambito IPv4.

[2] In IPv6 AH ed ESP sono due extension header. IKE, sia in ambito IPv4 che IPv6, è invece un protocollo di livello applicazione.

[3] Per alcuni concetti base relativi alla crittografia si veda l'appendice A.1.

[4] In [2] si fa esplicito riferimento ad IKE come protocollo predefinito di IPsec, tuttavia si dice che è possibile adottare anche altre soluzioni. A rigore, si può dire che il "nucleo" di IPsec è costituito esclusivamente da AH, ESP e da alcuni elementi collegati (SA, SAD e SPD, che verranno illustrati in seguito), mentre il meccanismo di scambio delle chiavi, per quanto necessario, non fa propriamente parte di IPsec. Nel capitolo 5 si affronterà il problema dell'instaurazione della connessione IPsec, analizzando alcune possibili estensioni e alternative all'attuale protocollo IKE.

le security association attive[5] su un host (o security gateway) sono contenute in un database detto *SAD* (Security Association Database), mentre esiste un altro database detto *SPD* (Security Policy Database) che contiene le politiche di sicurezza: tramite esse il sistema decide se un pacchetto IP debba essere scartato, lasciato passare in chiaro oppure elaborato tramite IPsec, basandosi su parametri come l'indirizzo IP sorgente o destinazione, la porta sorgente o destinazione, il protocollo di trasporto. Le security association possono essere combinate tra loro, sia nel caso che i nodi terminali siano gli stessi sia nel caso siano diversi: per esempio si potrebbero avere due SA in modalità trasporto tra due host (una per AH e una per ESP), oppure una SA in modalità trasporto tra due host a cui si aggiunge una SA in modalità tunnel tra due security gateway che stanno tra i due host[6].

2.1.2 AH (Authentication Header)

Il protocollo AH (Authentication Header) fornisce servizi di autenticazione, integrità e protezione da attacchi di tipo *replay*[7]. AH autentica l'intero pacchetto IP, ad eccezione dei campi variabili dell'header IP[8] (ovvero type of service, flags, fragment offset, time to live, header checksum, più alcune delle opzioni) che, essendo modificabili dai nodi intermedi, non possono essere autenticati[9]. La posizione dell'header AH all'interno del pacchetto IP, nelle modalità trasporto e tunnel, è mostrata in figura 2.1.

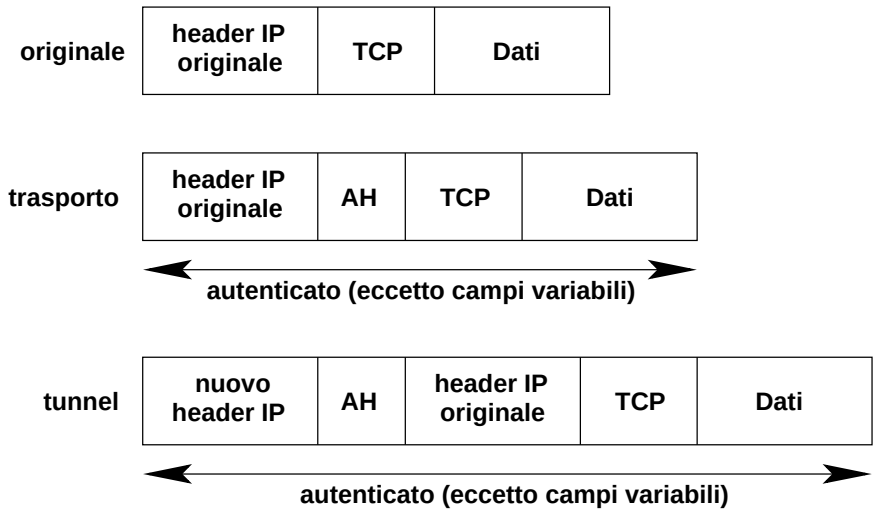

Figura 2.1: Posizionamento di AH all'interno del pacchetto IP.

L'header del protocollo AH è mostrato in figura 2.2; segue una breve descrizione dei campi che lo compongono.

Next header contiene il codice identificativo del protocollo dell'header successivo, per esempio TCP o ESP.

[5] Una SA ha un tempo di vita, che può essere specificato in termini temporali oppure come quantità di dati trasferiti.

[6] Non c'è limite al numero di tunnel sovrapposti, mentre per la modalità trasporto è possibile applicare solamente una volta ciascuno dei protocolli AH e ESP (e in questo caso bisogna necessariamente applicare prima ESP e poi AH).

[7] Un *replay attack* consiste nell'immissione in rete da parte dell'intruso di un pacchetto autentico precedentemente intercettato.

[8] Per il calcolo dei dati di autenticazione tali campi vengono posti a zero.

[9] Ci si riferisce all'header IP esterno. Nella modalità tunnel l'intero pacchetto originale (compreso l'header IP) viene incapsulato in un nuovo pacchetto, e quindi è interamente autenticato.

Payload length contiene la lunghezza dell'header AH, espressa in parole di 32 bit, meno 2.

Reserved riservato per usi futuri, deve essere posto a zero.

Security Parameters Index (SPI) contiene un valore numerico che, insieme con l'indirizzo IP di destinazione e il protocollo (ovvero AH, in questo caso) identifica la security association utilizzata. Viene stabilito dal destinatario quando la SA viene negoziata.

Sequence number specifica il numero di sequenza del pacchetto all'interno della SA, per prevenire i replay attack. Il destinatario gestisce i numeri di sequenza (se il servizio anti-replay è abilitato) tramite un meccanismo a finestra.

Authentication data contiene il valore per il controllo dell'integrità (Integrity Check Value — ICV) del pacchetto. La lunghezza di questo campo è variabile ma deve essere un multiplo di 32 bit, per cui è possibile inserire un padding.

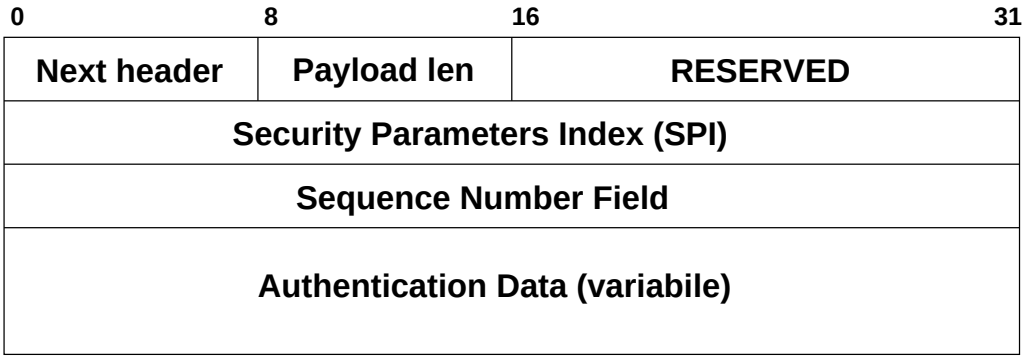

Figura 2.2: Formato dell'header AH.

AH presuppone che esista già una security association tra i due nodi, e non si preoccupa quindi di negoziarne i parametri. La specifica del protocollo AH si trova in [3].

2.1.3 ESP (Encapsulating Security Payload)

Il protocollo ESP (Encapsulating Security Payload) fornisce servizi di riservatezza, autenticazione, integrità e protezione anti-replay. È possibile utilizzare solo il servizio di riservatezza, oppure solo i servizi di autenticazione e integrità (ed eventualmente anti-replay), oppure tutti i servizi insieme. Per quanto riguarda l'autenticazione, questa differisce da quella fornita dal protocollo AH in quanto non copre l'header IP esterno. La posizione di ESP all'interno del pacchetto IP nelle modalità tunnel e trasporto è mostrata in figura 2.3.

Come si vede nella figura 2.3, ESP aggiunge sia un header sia un trailer, perché incapsula tutti i dati che protegge. Ecco nel dettaglio il formato del pacchetto, mostrato in figura 2.4.

Security Parameters Index (SPI) contiene un valore numerico che, insieme con l'indirizzo IP di destinazione e il protocollo (in questo caso ESP), permette di identificare la security association utilizzata. È analogo all'omonimo campo presente in AH.

Sequence number contiene il numero di sequenza del pacchetto nell'ambito della security association, come in AH.

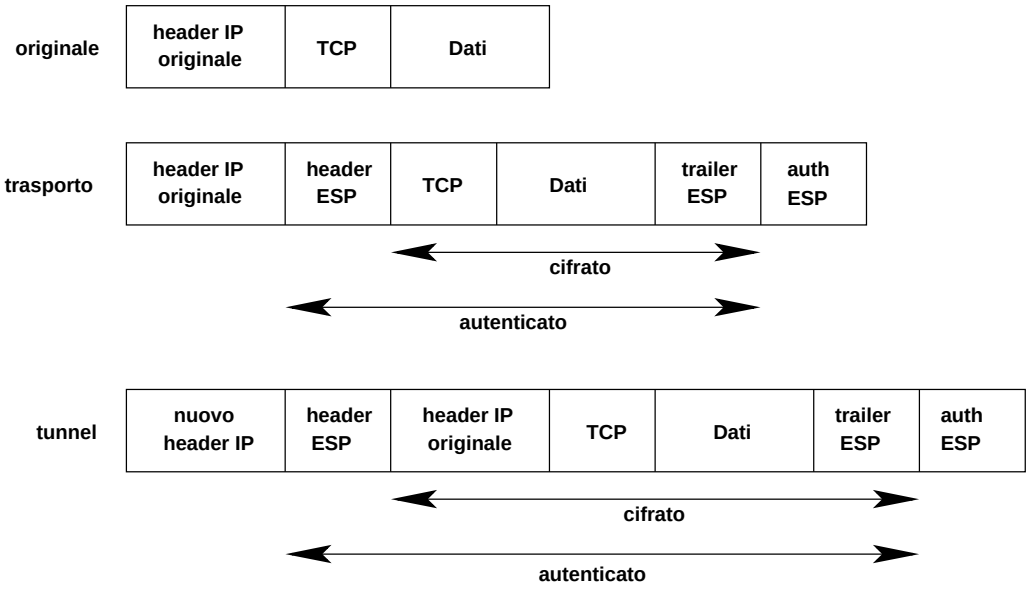

Figura 2.3: Posizionamento di ESP all'interno del pacchetto IP.

Payload data contiene il payload del pacchetto IP originale (se in modalità trasporto) oppure l'intero pacchetto IP originale (se in modalità tunnel), cifrato se si utilizza il servizio di riservatezza. Nel caso l'algoritmo di cifratura utilizzato necessiti di un vettore di inizializzazione (Initialization Vector — IV), questo viene inserito all'inizio del payload.

Padding il padding (variabile tra 0 e 255 byte) può essere necessario sia perché l'algoritmo di cifratura può richiedere che il testo in chiaro abbia una dimensione multipla di un certo valore, sia per assicurare il corretto allineamento dei campi successivi, come mostrato in figura 2.4. È anche possibile aggiungere un padding per limitare gli effetti di un'analisi del traffico basata sulla dimensione dei pacchetti.

Pad length contiene la lunghezza del padding.

Next header contiene il codice identificativo del protocollo per i dati contenuti nel payload, per esempio TCP o UDP. Si noti che, se si utilizza il servizio di riservatezza, questo campo è cifrato.

Authentication data contiene il valore di controllo integrità (ICV), calcolato sull'intero pacchetto ESP escluso questo campo. È presente solo se si utilizza il servizio di autenticazione/integrità.

Anche ESP, come AH, presuppone che esista già una security association tra i due nodi e non si preoccupa di negoziarne i parametri. La specifica del protocollo ESP è contenuta in [4].

2.1.4 IKE (Internet Key Exchange)

Come si è visto, nell'architettura di IPsec è centrale il concetto di security association, ma né AH né ESP si preoccupano della gestione delle SA. Le security association possono essere costruite manualmente o automaticamente; è chiaro però che una loro gestione manuale non è in genere praticabile se non in contesti molto limitati, per cui è necessario un meccanismo automatico: il protocollo *IKE* (Internet Key Exchange) risolve questo problema.

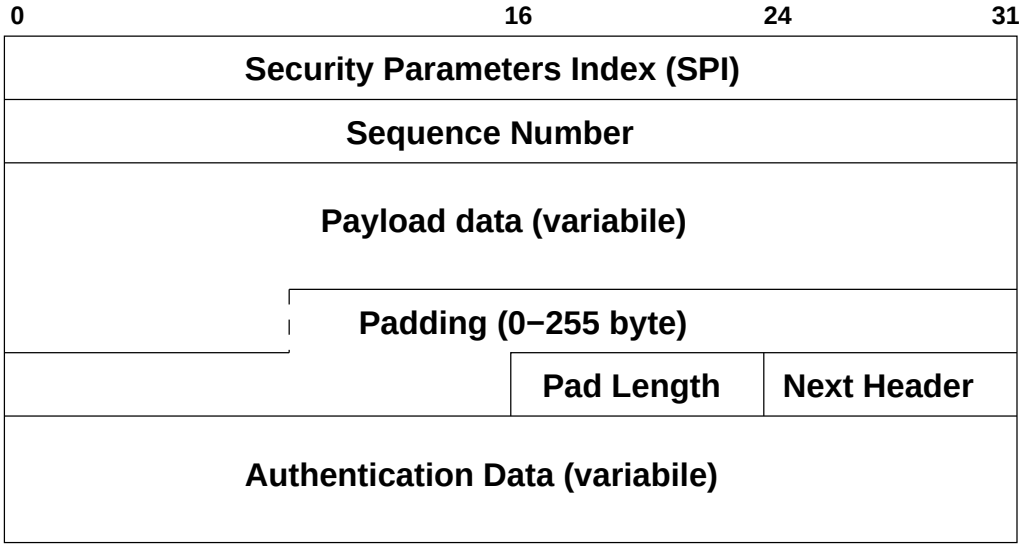

Figura 2.4: Formato del pacchetto ESP.

Il protocollo *ISAKMP* (Internet Security Association and Key Management Protocol, specificato in [5]) definisce le procedure e il formato dei pacchetti per la gestione (creazione, modifica, cancellazione) delle security association e per lo scambio e l'autenticazione delle chiavi, indipendentemente dalla tecnica di generazione delle chiavi stesse, dagli algoritmi di cifratura e dai meccanismi di autenticazione. Il significato preciso dei pacchetti ISAKMP dipende dal contesto (dominio interpretativo) utilizzato: nel nostro caso si tratta del dominio IPsec, definito in [6]. IKE è un protocollo ibrido, che integra ISAKMP con parte dei protocolli Oakley[10] e SKEME, ed è specificato in [8].

IKE ha due fasi: nella prima i due nodi creano una security association per IKE stesso (detta *ISAKMP SA*[11]), ovvero un canale sicuro da utilizzare per i messaggi di IKE, nella seconda fase utilizzano la ISAKMP SA per negoziare security association per altri protocolli. Nella prima fase si può usare il *main mode* oppure l'*aggressive mode*, mentre nella seconda fase si utilizza il *quick mode*.

La sequenza di messaggi del main mode è illustrata in figura 2.5. I primi due messaggi servono a negoziare i parametri della ISAKMP SA, nei due successivi avviene lo scambio dei valori per il protocollo Diffie–Hellman[12] (insieme con alcuni dati ausiliari), infine gli ultimi due servono ad autenticare lo scambio Diffie–Hellman. Il metodo di autenticazione negoziato nella prima parte influenza la composizione dei messaggi successivi, ma non il loro scopo; sono infatti possibili quattro forme di autenticazione: mediante firma digitale (lo scambio in figura 2.5 si riferisce a tale modalità), mediante cifratura a chiave pubblica[13] (con due diverse modalità) e mediante chiave segreta condivisa. Si faccia attenzione a non confondere l'autenticazione dell'*identità dell'interlocutore*, che è quella di cui si preoccupa IKE e alla quale ci si riferisce ora, con l'autenticazione della *provenienza dei dati*, che è quella garantita dal servizio di autenticazione di AH ed ESP: la prima, tramite l'uso di un certificato, di un segreto condiviso o della crittografia a chiave pubblica, è volta a garantire che l'interlocutore è chi effettivamente sostiene di essere, mentre la seconda è volta a garantire che i dati ricevuti provengono

[10] Oakley [7] è un protocollo di scambio delle chiavi basato su Diffie–Hellman, integrato con alcuni accorgimenti volti ad aumentarne la sicurezza.

[11] Al contrario di quelle di AH e ESP, la ISAKMP SA (detta anche IKE SA) è bidirezionale.

[12] Si veda l'appendice A.3.

[13] Le chiavi pubbliche devono essere scambiate in qualche modo prima dell'handshake di IKE.

effettivamente dall'interlocutore identificato dalla prima.

Ecco i messaggi nel dettaglio: nel primo l'iniziatore[14] oltre all'header (HDR) manda un insieme di proposte per la security association da negoziare (SA), il ricevitore replica indicando nel proprio messaggio la proposta che ha scelto. Nel terzo e quarto messaggio avviene lo scambio Diffie–Hellman, tramite il payload KE (che contiene i dati pubblici scambiati per effettuare il DH), e il payload N*x* (che contiene il "nonce", ovvero un numero casuale ausiliario utilizzato per la generazione della chiave e per l'autenticazione). Infine nel terzo e quarto messaggio (che sono cifrati con la chiave appena negoziata) i due interlocutori procedono all'autenticazione: si identificano con il payload ID*xx* (che contiene un identificativo come l'indirizzo IP, o il nome dell'host o altro — si veda [6] per i dettagli) fornendo eventualmente anche un certificato (payload CERT, opzionale) e inserendo una firma (payload SIG_*x*) calcolata su un hash.

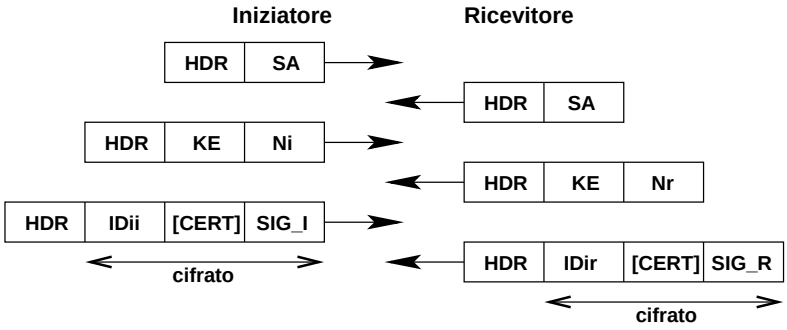

Figura 2.5: IKE main mode.

L'aggressive mode, la cui sequenza di messaggi è illustrata in figura 2.6 (sempre nel caso dell'autenticazione con firma digitale), ottiene lo stesso risultato del main mode ma con un numero inferiore di messaggi (tre anziché sei), al prezzo però di non proteggere le identità degli interlocutori: dato che i payload ID*xx* sono scambiati prima che sia terminato lo scambio Diffie–Hellman, questi viaggiano in chiaro e non cifrati come nel caso del main mode[15].

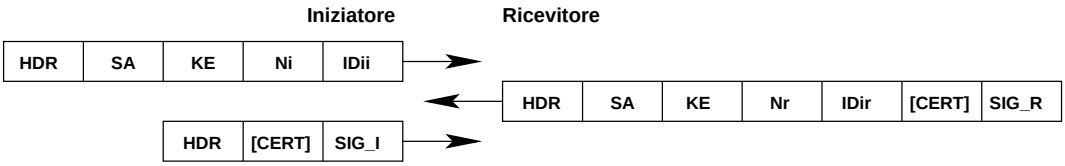

Figura 2.6: IKE aggressive mode.

Dopo aver terminato la fase 1, con il main mode o con l'aggressive mode, i due interlocutori hanno creato una ISAKMP SA, e quindi possono procedere alla fase 2 e creare security association per altri protocolli (non ISAKMP). Questa negoziazione avviene mediante il quick mode, ed è illustrata in figura 2.7. Al contrario di quanto avviene nella fase 1, qui tutti i messaggi sono cifrati perché sono protetti dalla ISAKMP SA. Nel primo messaggio l'iniziatore fa un insieme di proposte per i parametri della SA da negoziare (payload SA), inserisce un valore casuale (payload Ni) che sarà utilizzato per calcolare la

[14] I termini *iniziatore* (initiator) e *ricevitore* (responder) servono solo a distinguere chi dei due interlocutori ha iniziato l'handshake. IKE è un protocollo peer–to–peer, e non client–server.

[15] Se si usa l'autenticazione mediante cifratura a chiave pubblica è possibile proteggere le identità anche nel caso dell'aggressive mode.

chiave, e un hash (payload HASH) per autenticare il messaggio; opzionalmente è possibile includere due payload di tipo ID che contengono gli identificativi delle due parti che stanno negoziando la SA[16]. In questo modo il Diffie–Hellman viene effettuato con gli stessi valori scambiati nella fase 1, per cui non si ha la proprietà della *perfect forward secrecy*[17]: se si vuole la PFS è necessario effettuare un nuovo scambio DH includendo il payload KE. Nel secondo messaggio il ricevitore sceglie tra le proposte presentate (payload SA), comunica il proprio "nonce" (payload Nr), eventualmente inserisce i payload degli identificativi e il payload KE (quest'ultimo nel caso si voglia la PFS), e autentica il messaggio tramite il payload HASH. Lo scambio si conclude con un terzo messaggio di conferma, contenente un HASH calcolato, tra le altre cose, su entrambi i "nonce".

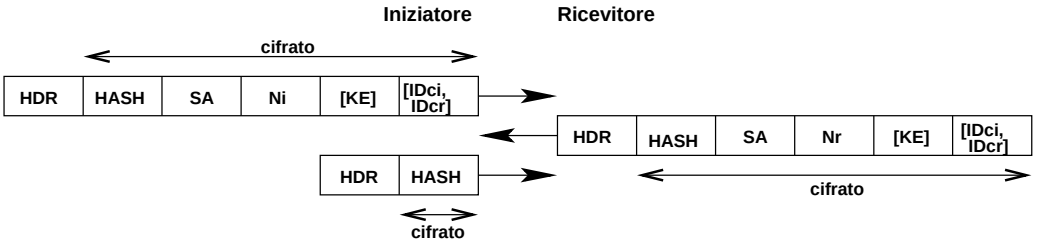

Figura 2.7: IKE quick mode.

Una singola negoziazione ha come risultato in realtà due security association: una dall'iniziatore al ricevitore e una in senso inverso[18]; è inoltre possibile fare più di una negoziazione con un solo scambio quick mode.

2.1.5 Elaborazione dei pacchetti IPsec

In una macchina che implementa IPsec, tutto il traffico è soggetto alle politiche di IPsec per cui è necessario consultare il Security Policy Database (SPD) per decidere se un pacchetto deve essere lasciato passare normalmente, deve essere passato all'elaborazione IPsec, oppure deve essere scartato[19]. Nell'analizzare l'elaborazione dei pacchetti IPsec si distingue il traffico in uscita (*outbound*, ovvero diretto verso un'interfaccia di rete) dal traffico in entrata (*inbound*, ovvero proveniente da un'interfaccia di rete). Si noti che un security gateway può elaborare lo stesso pacchetto due volte: prima in entrata, proveniente da un'interfaccia, e poi in uscita, diretto verso un'altra interfaccia.

Per il traffico in uscita, bisogna innanzitutto cercare nel SPD un selettore applicabile al pacchetto. Se questo richiede un'elaborazione IPsec bisogna associare il pacchetto ad una SA esistente (cercandola nel SAD) oppure invocare IKE per crearne una nuova, poi si esegue l'elaborazione necessaria (autenticazione, cifratura, incapsulamento del pacchetto in uno nuovo — che deve essere costruito — se è richiesta la modalità tunnel) e infine si passa il pacchetto al livello inferiore (figura 2.8-a). In un host (non in un security gateway) con un'implementazione basata sui socket, è possibile consultare il SPD

[16] Se non vengono inseriti, si assumono gli indirizzi IP delle due parti della ISAKMP SA. Se IKE sta agendo come client per conto di un'altra parte, gli ID devono necessariamente essere inclusi.

[17] La perfect forward secrecy (PFS) è la proprietà per cui la compromissione di una singola chiave permette l'accesso solo alle informazioni protette da quella singola chiave. Perché questa proprietà sia verificata è necessario che una chiave usata per proteggere la trasmissione dei dati non sia usata per derivare altre chiavi, e se la chiave usata per proteggere la trasmissione dei dati era stata derivata da un'altra chiave, quella chiave non deve essere usata per derivare altre chiavi.

[18] Le chiavi saranno diverse, perché ognuna è calcolata partendo anche dal SPI, che è diverso per le due SA.

[19] Quest'ultima è la scelta di default, se non è specificata una politica applicabile al pacchetto.

quando il socket viene creato, in modo da determinare una sola volta l'elaborazione IPsec necessaria per il traffico associato a quel socket.

Per il traffico in ingresso, bisogna innanzitutto ricomporre il datagramma IP nel caso sia stato frammentato, dopo di che si identificano i pacchetti che devono essere elaborati con IPsec tramite la presenza dei valori relativi a ESP oppure ad AH nel campo "protocol" dell'header IP. Per i pacchetti IPsec, si identifica la SA relativa grazie al valore SPI presente nell'header AH o ESP, si applica l'elaborazione IPsec richiesta, e si controlla il SPD per accertarsi che le operazioni effettuate corrispondano alle politiche specificate. Infine il pacchetto viene inoltrato verso la destinazione successiva oppure passato al livello superiore (figura 2.8-b).

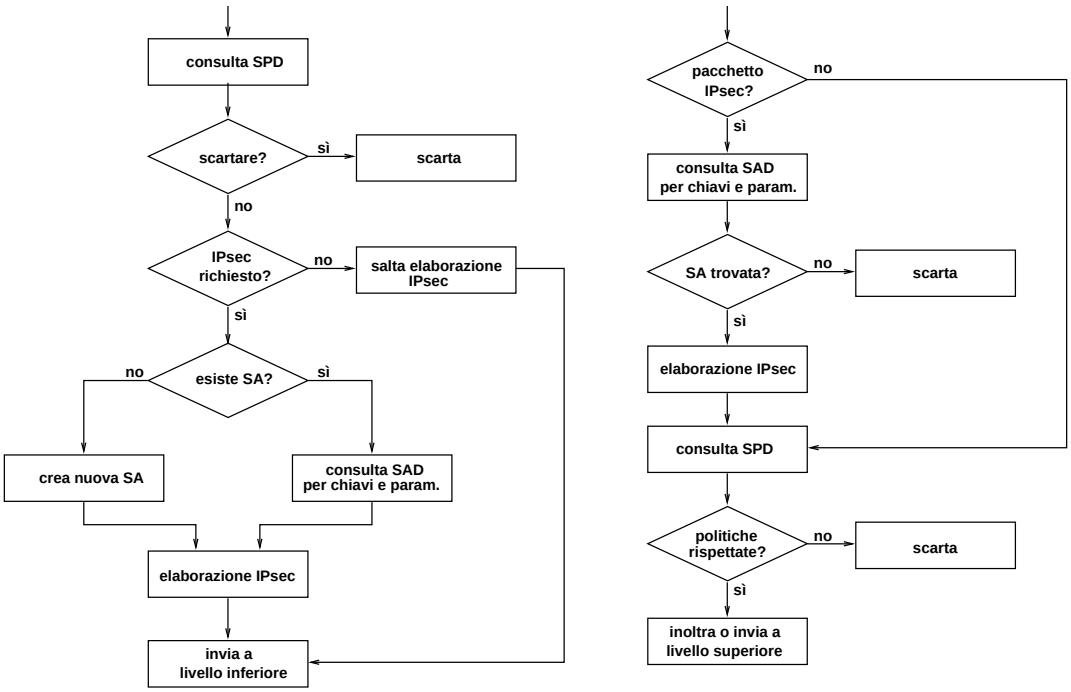

Figura 2.8: Elaborazione dei pacchetti IPsec.

2.2 Utilizzi

Dato che fornisce sicurezza a livello IP, IPsec è una soluzione molto generale e si presta ad essere usato per diverse applicazioni. Si vedranno ora le due attualmente più comuni, ovvero le reti private virtuali e i "road warrior".

2.2.1 Reti private virtuali (VPN)

Le *reti private virtuali* (VPN, Virtual Private Networks), permettono di collegare tra loro delle reti locali tramite la rete Internet, garantendo però sicurezza (riservatezza, integrità...) al traffico che viaggia sulla rete esterna. Utilizzando IPsec questo è possibile: è sufficiente inserire nelle reti locali dei security gateway attraverso i quali far passare tutto il traffico diretto alle altre reti, e creare dei tunnel IPsec tra i

security gateway. In questo modo il traffico diretto da una rete all'altra viaggia su Internet ma è protetto dal tunnel IPsec, il tutto senza che gli host né tantomeno le applicazioni debbano preoccuparsene. Si ha così, virtualmente, un'unica rete privata che sfrutta l'infrastruttura della rete pubblica. Un esempio di questa architettura è illustrato in figura 2.9.

Le VPN sono un caso tipico in cui si può avere la sovrapposizione di vari livelli di IPsec. Si Supponga che il tunnel tra i security gateway utilizzi ESP con cifratura e integrità: allora tutto il traffico che fluisce tra di essi verrà cifrato e autenticato. In qualche caso si potrebbe però volere anche sicurezza end–to–end, e impostare IPsec tra due host appartenenti a reti diverse, per esempio utilizzando ancora ESP. Allora tutto il traffico tra questi due host verrebbe elaborato da IPsec sia sugli host, dove verrebbe cifrato/decifrato una volta, sia sui gateway, dove sarebbe cifrato/decifrato un'altra volta. In questo caso si ha quindi una sovrapposizione di SA, con end-point diversi.

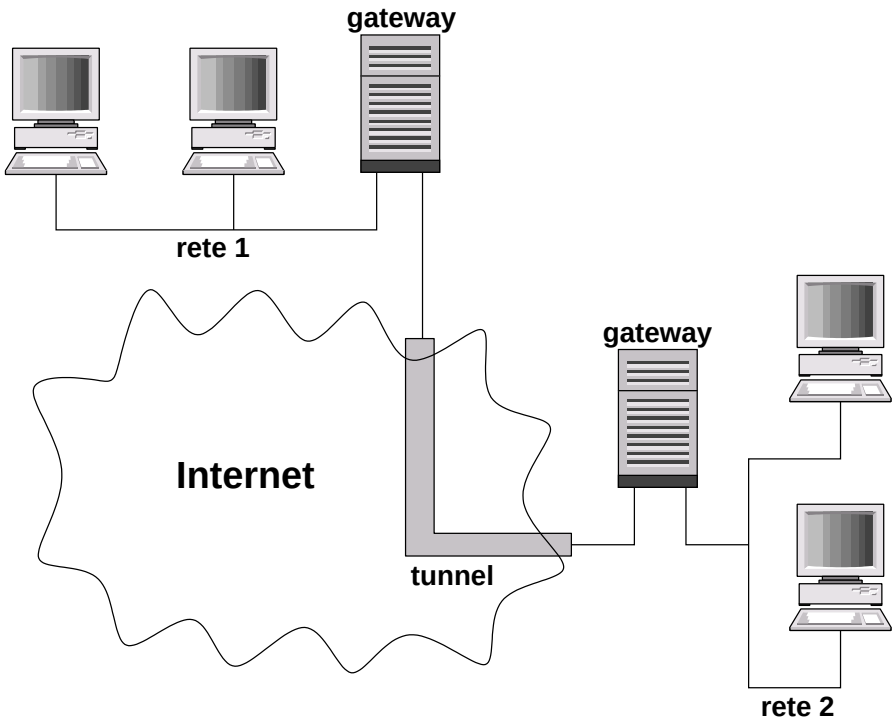

Figura 2.9: Rete privata virtuale.

2.2.2 "Road warrior"

Il "*road warrior*" si può vedere come un caso degenere della VPN, in cui da una parte si ha un singolo host anziché una rete (figura 2.10): un esempio tipico è quello in cui una persona vuole accedere alla rete aziendale trovandosi all'esterno, per esempio a casa propria oppure in viaggio, tramite una connessione sicura, in modo da essere autenticato e da proteggere le informazioni in transito. La particolarità del road warrior rispetto alla VPN è che in questo caso in genere l'host esterno ha un indirizzo IP dinamico e quindi non noto a priori, al contrario di quanto avviene con i security gateway che hanno di solito un indirizzo IP statico e noto in fase di configurazione.

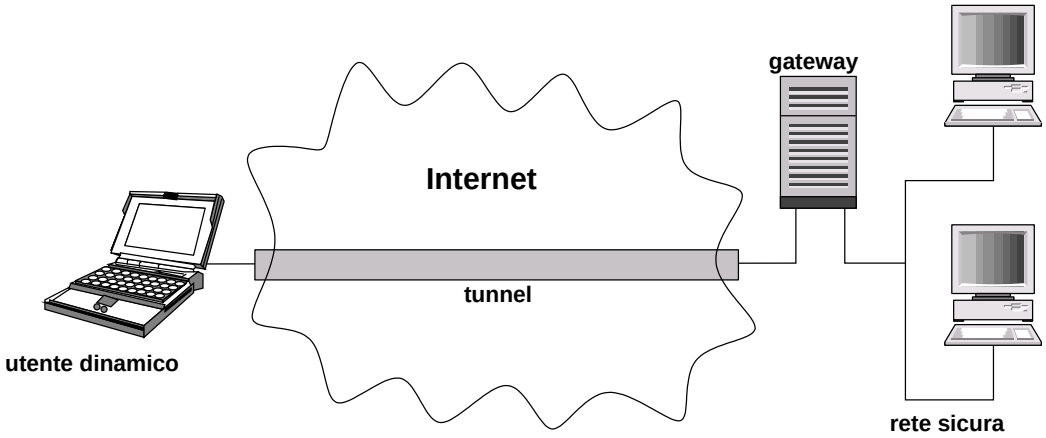

Figura 2.10: Road warrior.

2.3 Implementazioni

IPsec può essere implementato in vari modi (questa classificazione è fornita in [2]):

- tramite integrazione nell'implementazione nativa di IP. Questo richiede l'accesso al codice sorgente di IP, ed è applicabile sia agli host che ai security gateway.
- "bump–in–the–stack" (BITS), ovvero in mezzo tra IP e i driver di rete. Questo approccio non richiede accesso al codice sorgente di IP e può quindi essere utile per sistemi legacy. È di solito utilizzato negli host.
- "bump–in–the–wire" (BITW), ovvero tramite un dispositivo esterno che si occupi delle operazioni crittografiche, in genere dotato di un proprio indirizzo IP. Questa soluzione è utilizzata in particolare in ambienti militari.

IKE è un protocollo di livello applicazione, generalmente implementato come un demone in ascolto sulla porta UDP 500.

2.3.1 Linux FreeS/WAN

FreeS/WAN è un'implementazione libera (sotto licenza GNU GPL) di IPsec per Linux. La versione più recente è la 1.91 (uscita a giugno 2001), e si compone di due parti principali: *KLIPS* (kernel IPsec), che implementa AH, ESP e la gestione dei pacchetti all'interno del kernel, e il demone *Pluto* che implementa IKE. L'implementazione di IPsec è di tipo bump–in–the–stack. FreeS/WAN implementa come algoritmo di cifratura solamente il triplo DES[20], MD5 e SHA-1 come algoritmi di autenticazione e supporta IPComp [9] per la compressione del payload IP. L'implementazione di IKE non supporta l'aggressive mode, e fornisce l'autenticazione mediante segreto condiviso o mediante chiavi RSA (esiste una patch per poter utilizzare i certificati X.509). Una caratteristica particolare e interessante di FreeS/WAN è la "opportunistic encryption" (si veda al capitolo 5).

[20] Il DES semplice, pur essendo implementato in quanto necessario per fornire il triplo DES, non è utilizzabile nonostante sia l'unico algoritmo richiesto dallo standard. Questa è una scelta esplicita del gruppo che sviluppa FreeS/WAN, motivata dal fatto che il DES non è più considerato sicuro.

2.3.2 Altre implementazioni

Per quanto riguarda i sistemi operativi, è degno di nota il fatto che Microsoft abbia incluso IPsec in Windows 2000, integrandolo con la gestione dei domini Windows. Le funzionalità differiscono tra la versione Professional del sistema operativo e la versione Server.

Un'altra implementazione interessante di IPsec è PGPnet, inclusa in PGP per Windows (anche nella versione freeware). È semplice da installare e da configurare tramite interfaccia grafica, permette l'autenticazione dell'interlocutore tramite chiave PGP e permette connessioni IPsec dinamiche, per cui due macchine con PGPnet possono comunicare tramite IPsec senza che sia preconfigurata staticamente una connessione tra di esse.

3 TLS

3.1 Descrizione

TLS (Transport Layer Security) è un protocollo di livello 5 (sessione) che opera quindi al di sopra del protocollo di livello trasporto. È uno standard IETF, specificato in [10], e deriva dal protocollo *SSL* (Secure Socket Layer). SSL è stato originariamente proposto da Netscape ma è un protocollo aperto, l'ultima versione è la 3.0 e non è mai stato standardizzato; TLS 1.0 (la versione attuale) è molto simile a SSL 3.0[1] tuttavia non è compatibile con questo, le implementazioni comunque supportano generalmente sia TLS 1.0 sia SSL 3.0, ed è garantita l'interoperabilità.

Lo scopo di TLS è quello di permettere una comunicazione sicura *tra due applicazioni*, fornendo al flusso di dati tra di esse servizi di autenticazione, integrità e riservatezza. È composto da due livelli: il *TLS Record Protocol* e il *TLS Handshake Protocol*: il primo è il livello più basso, che opera direttamente al di sopra di un protocollo di trasporto affidabile come TCP ed è usato per incapsulare (offrendo servizi di sicurezza) protocolli di livello superiore tra cui l'Handshake Protocol stesso, il quale si occupa della fase di negoziazione in cui si autentica l'interlocutore e si stabiliscono le chiavi segrete condivise. TLS è indipendente dal protocollo di livello applicazione che ne utilizza i servizi, tuttavia lo standard non specifica come l'applicazione lo debba utilizzare, ovvero come decidere di iniziare un handshake TLS e come interpretare i certificati scambiati per l'autenticazione delle parti.

3.1.1 TLS Record Protocol

Il TLS Record Protocol riceve i dati dal livello superiore, li suddivide in blocchi di dimensioni opportune, eventualmente li comprime, calcola un MAC[2], cifra il tutto e trasmette il risultato di questa elaborazione. I dati in ricezione vengono, nell'ordine, decifrati, verificati (verifica del MAC), decompressi, riassemblati e infine consegnati al livello superiore. I tipi di dato che possono essere consegnati al Record Protocol (e che quindi identificano il protocollo di livello superiore) definiti da [10] sono quattro, ovvero: "handshake protocol", "alert protocol", "change cipher spec protocol" e "application data protocol"; i primi tre fanno parte del TLS Handshake Protocol e saranno descritti successivamente, mentre l'ultimo indica che si tratta di dati del livello applicazione, ovvero del protocollo che sta sopra a TLS.

Il Record Protocol opera sempre all'interno di uno stato, che definisce gli algoritmi di compressione, cifratura e autenticazione e i parametri relativi (come le chiavi). Alla connessione sono sempre associati quattro di questi stati: gli stati correnti in lettura e in scrittura e i corrispondenti stati pendenti; i dati vengono elaborati secondo gli stati correnti, mentre il TLS Handshake Protocol può impostare i parametri per gli stati pendenti e rendere corrente uno stato pendente. In quest'ultimo caso lo stato pendente viene reinizializzato ad uno stato vuoto (che non può diventare corrente); lo stato iniziale specifica che non si utilizza nessun algoritmo di compressione, cifratura e autenticazione.

[1] Infatti si "presenta" come SSL 3.1 e usa le stesse porte di SSL.
[2] Si veda l'appendice A.5.

I parametri di sicurezza definiti per uno stato sono i seguenti:

- "connection end": specifica se l'entità in questione è il client o il server (in TLS c'è sempre questa distinzione);
- "bulk encryption algorithm": indica l'algoritmo di cifratura e i relativi parametri, come la lunghezza della chiave o il fatto che sia "esportabile" oppure no;
- "MAC algorithm": indica l'algoritmo di autenticazione e i relativi parametri;
- "compression algorithm": indica l'algoritmo di compressione e i relativi parametri;
- "master secret": sequenza segreta di 48 byte condivisa dai due interlocutori;
- "client random": 32 byte casuali forniti dal client;
- "server random": 32 byte casuali forniti dal server.

A partire da questi parametri vengono generati sei valori, ovvero le chiavi per la cifratura e per l'autenticazione e il vettore di inizializzazione (quest'ultimo solo per cifrari a blocco in modalità CBC[3]) sia per il client che per il server; ognuno dei due interlocutori utilizza i propri parametri (parametri in scrittura) in fase di trasmissione e quelli dell'altra parte (parametri in lettura) in fase di ricezione. Una volta che i parametri di sicurezza sono stati impostati e le chiavi sono state generate, gli stati della connessione possono essere istanziati rendendoli correnti. Gli stati devono essere aggiornati ad ogni record elaborato, e ciascuno comprende le seguenti informazioni: stato della compressione (stato corrente dell'algoritmo di compressione), stato della cifratura (comprende la chiave e altre informazioni necessarie a definire lo stato dell'algoritmo, per esempio l'ultimo blocco nel caso di un cifrario a blocco in modalità CBC), MAC secret e numero di sequenza.

Come detto in precedenza, il TLS Record Protocol riceve dei dati in modo opaco dal livello superiore, li suddivide in blocchi, applica un algoritmo di compressione, poi inserisce un MAC (calcolato tramite un hash sulla chiave segreta, il numero di sequenza, il frammento compresso e altri parametri) e infine cifra il tutto, MAC compreso (figura 3.1).

3.1.2 TLS Handshake Protocol

Il TLS Handshake Protocol è costituito da tre sotto-protocolli ("change cipher spec", "alert" e "handshake"), che permettono ai due interlocutori di autenticarsi, negoziare i parametri di sicurezza, istanziare i parametri negoziati e notificare condizioni di errore. Il TLS Handshake Protocol è responsabile della negoziazione di una sessione, che è costituita dai seguenti parametri: "session identifier" (identificatore della sessione scelto dal server), "peer certificate" (certificato X.509 dell'interlocutore — può mancare), "compression method" (algoritmo di compressione), "cipher spec" (algoritmi di cifratura e autenticazione e relativi parametri crittografici), "master secret", "is resumable" (flag che indica se la sessione può essere utilizzata per iniziare nuove connessioni). Da questi sono ricavati i parametri di sicurezza visti in precedenza, che vengono utilizzati dal Record Protocol. La stessa sessione, come suggerito dalla presenza del flag "is resumable", può essere utilizzata per creare più connessioni.

[3] Si veda l'appendice A.2.

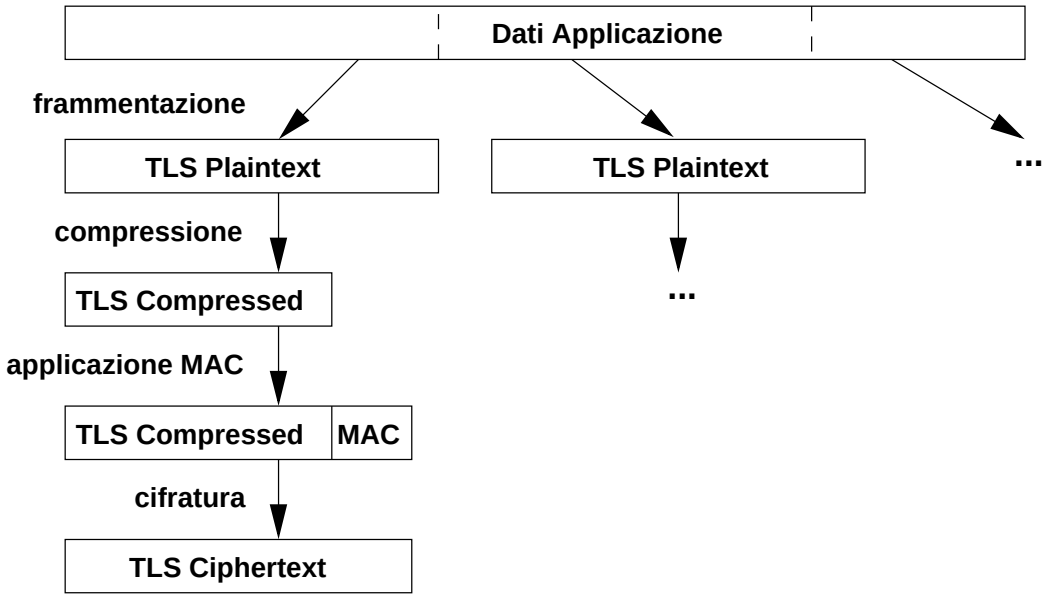

Figura 3.1: Sequenza delle trasformazioni di TLS.

3.1.2.1 Change cipher spec

Il change cipher spec protocol consiste in un solo messaggio, elaborato secondo lo stato corrente, che indica di passare al nuovo stato negoziato. Subito dopo aver mandato questo messaggio il mittente copia lo stato pendente in scrittura in quello corrente, mentre la ricezione del messaggio fa sì che l'attuale stato pendente in lettura divenga quello corrente.

3.1.2.2 Alert

TLS supporta una serie di messaggi di controllo ed errore, che possono essere inviati per notificare all'interlocutore una particolare situazione o evento. Sono presenti ad esempio messaggi per notificare la chiusura della connessione, un errore nella verifica del MAC o nella decifratura, vari tipi di errore relativi ai certificati (tutti i messaggi sono specificati in [10]). I messaggi di alert, come tutti i dati trasmessi, vengono elaborati secondo lo stato corrente, e a seconda della loro gravità sono classificati come "fatal" (i quali provocano sempre l'interruzione immediata della connessione[4]) oppure come "warning".

3.1.2.3 Handshake

L'handshake protocol è utilizzato per negoziare i parametri di sicurezza di una sessione. La specifica fa notare che le applicazioni non dovrebbero contare sul fatto che TLS stabilisca una connessione la più sicura possibile tra i due interlocutori, perché sono possibili alcuni attacchi che mirano a far accordare le due parti su un basso livello di sicurezza anche se sarebbe possibile utilizzarne uno maggiore. Per questo l'applicazione dovrebbe essere consapevole del livello di sicurezza richiesto, e in base a questo decidere come comportarsi rispetto a ciò che è stato negoziato.

[4] Altre connessioni in corso create a partire dalla stessa sessione possono continuare, ma l'identificatore della sessione viene marcato come non più valido impedendo di creare nuove connessioni sotto quella sessione.

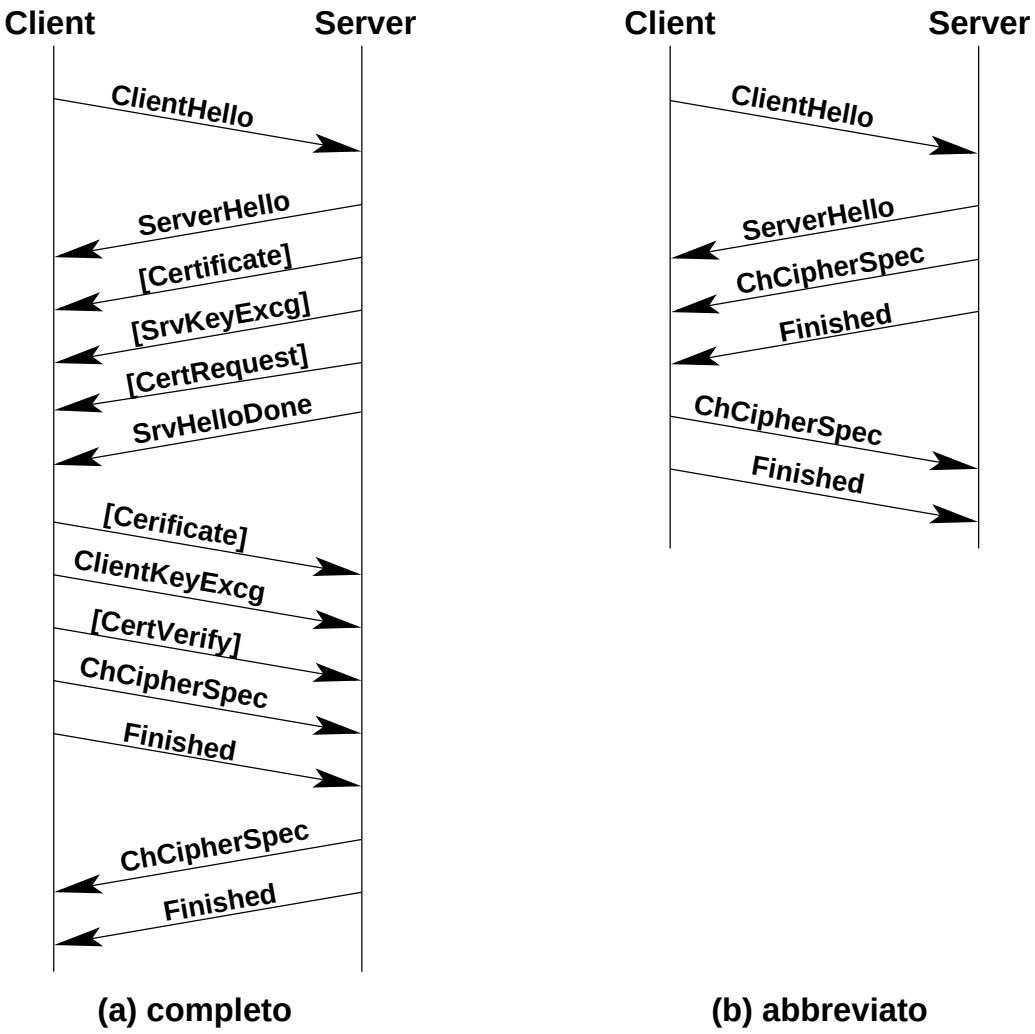

Figura 3.2: Handshake completo e abbreviato di TLS.

Si analizzerà ora nel dettaglio la sequenza di messaggi dell'handshake, illustrata in figura 3.2-a (i messaggi indicati tra parentesi quadre sono opzionali o dipendenti dalla situazione). Nel primo messaggio, ClientHello, il client indica quali algoritmi supporta in ordine di preferenza e inserisce un valore casuale; il server risponde con un ServerHello indicando quali algoritmi (compressione, cifratura, autenticazione) ha scelto, e fornendo il codice identificativo della sessione e un numero casuale. Se il metodo di autenticazione[5] scelto non è anonimo il server invia poi il proprio certificato (messaggio Certificate), se invece è anonimo oppure se il certificato è valido solo per la firma invia un messaggio ServerKeyExchange, che contiene una chiave pubblica RSA[6] oppure un valore per effettuare uno scambio Diffie–Hellman. Se il server non è anonimo[7] può poi richiedere un certificato al client con il messaggio CertificateRequest (nel quale può indicare i tipi di certificato e le certification authority che accetta), infine manda il messaggio ServerHelloDone, che indica al client di procedere a sua volta. A questo punto il client invia il proprio certificato (se richiesto), e poi il messaggio ClientKeyExchange, che contiene il premaster secret (generato casualmente dal client) cifrato con la chiave RSA fornita dal server, oppure un valore per completare lo scambio Diffie–Hellman la cui chiave sarà il premaster secret. Se il client ha mandato un certificato che permette la firma invia poi il messaggio CertificateVerify come verifica esplicita del proprio certificato. Ora le parti si sono autenticate e sono in possesso dei dati necessari per calcolare il master secret (generato a partire dal premaster secret e dai valori casuali precedentemente scambiati), quindi il client manda un messaggio ChangeCipherSpec[8] e copia lo stato pendente appena negoziato in quello corrente[9]. I parametri del nuovo stato sono utilizzati per la prima volta con il messaggio Finished, che contiene dei dati che permettono di verificare la correttezza dell'handshake; il server risponde a sua volta con un messaggio ChangeCipherSpec e un messaggio Finished. A questo punto l'handshake è terminato e le due parti possono iniziare a scambiarsi i dati (application data).

Se successivamente i due interlocutori decidono di riprendere una vecchia sessione oppure di duplicare la sessione esistente utilizzano un handshake abbreviato, illustrato in figura 3.2-b. Il client inizia con un messaggio ClientHello, in cui include l'identificatore della sessione in corso, il server risponde con un ServerHello e procede subito con i messaggi ChangeCipherSpec e Finished, infine anche il client invia i messaggi ChangeCipherSpec e Finished. Questa procedura può essere avviata anche dal server con un messaggio di HelloRequest, dopo di che procede come sopra.

3.1.3 Differenze tra TLS 1.0 e SSL 3.0

Le differenze tra SSL 3.0 e TLS 1.0 sono poche e riguardano aspetti secondari, tuttavia sono sufficienti a far sì che i due protocolli non siano tra loro compatibili. Comunque sono in genere implementati entrambi, per cui l'interoperabilità funziona come segue: se un server che non supporta TLS riceve una richiesta di connessione con numero di versione 3.1 (che, per ragioni storiche, identifica TLS 1.0) risponderà che la massima versione supportata è la 3.0, per cui l'handshake può continuare con SSL 3.0. Analogamente, un server che supporta TLS risponderà ad una richiesta SSL 3.0 con un handshake SSL.

[5] Anche qui si faccia attenzione a non confondere l'autenticazione dell'identità dell'interlocutore, che è effettuata dall'Handshake Protocol e che può mancare, con l'autenticazione della provenienza dei dati, che è effettuata dal Record Protocol e che solitamente è sempre presente.

[6] Si veda l'appendice A.4.

[7] TLS permette di autenticare solo il server, sia il server che il client, oppure nessuno dei due. Non è invece possibile autenticare solo il client.

[8] Questo messaggio, come visto, costituisce un protocollo a sé stante e non è propriamente un messaggio di handshake.

[9] Lo stato corrente precedente era quello iniziale predefinito, che non prevede alcuna trasformazione.

Rispetto a SSL 3.0, TLS definisce molti più messaggi di errore, inoltre presenta alcune piccole modifiche volte ad aumentarne la sicurezza dal punto di vista crittografico: ci sono per esempio delle differenze nel calcolo del MAC (sia nell'algoritmo che nei campi coinvolti) e nell'espansione del master secret per la generazione delle chiavi (TLS usa una funzione pseudocasuale).

3.2 Utilizzi

TLS è un protocollo generale, e potrebbe essere impiegato con qualunque applicazione che usa TCP (TLS non può essere utilizzato sopra UDP). Sono state standardizzate molte porte per l'uso di vari protocolli applicativi al di sopra di TLS[10], tra cui HTTP (443), SMTP (465), NNTP (563), FTP (989 e 990), TELNET (992), IMAP4 (993), IRC (994) e POP3 (995). L'uso più tipico di TLS è comunque legato ad HTTP[11], nella trasmissione sicura di pagine web e di dati via browser (un caso molto frequente è quello della trasmissione del numero di carta credito in applicazioni di commercio elettronico). In questi ambiti, vista la scarsa diffusione dei certificati, viene in genere utilizzato con l'autenticazione del solo server.

3.3 Implementazioni

TLS è implementato sotto forma di librerie, oppure direttamente all'interno dell'applicazione che lo utilizza. Nella prima categoria ricade OpenSSL, un'implementazione open source di SSL e TLS sviluppata da un gruppo di volontari e giunta attualmente alla versione 0.9.6a, utilizzata tra l'altro dal modulo che aggiunge il supporto TLS/SSL al web server Apache. Un altro insieme di librerie open source che implementano TLS e SSL è Network Security Services (NSS) del progetto Mozilla. Per quanto riguarda i browser, TLS e SSL sono supportati sia da Microsoft Internet Explorer che da Netscape Navigator (quest'ultimo tramite NSS).

[10] Le stesse porte vengono utilizzate anche con SSL.
[11] Il protocollo HTTP protetto da TLS/SSL viene spesso indicato come HTTPS.

4 Confronto funzionale e di impatto

Nei capitoli precedenti si è vista la descrizione di IPsec e TLS accennando anche agli utilizzi attualmente più comuni dell'uno e dell'altro. Le due soluzioni presentano comunque alcuni aspetti simili e sono entrambe sufficientemente generali (ovvero non dedicate ad una specifica applicazione), per cui può essere utile confrontarle anche in una prospettiva generica, indipendentemente dai rispettivi casi d'uso tipici. Quest'analisi sarà suddivisa in due parti: la prima sarà dedicata alle differenze di funzionalità per quanto riguarda la sicurezza, ovvero ciò per cui i protocolli sono stati progettati, mentre la seconda sarà dedicata alle differenze di "impatto", ovvero alle difficoltà e agli "effetti collaterali" che possono derivare dall'uso dell'uno o dell'altro. Le principali differenze sono poi riassunte nella tabella 4.1.

4.1 Differenze di funzionalità

IPsec e TLS sono strutturati in maniera molto diversa: TLS è un singolo protocollo descritto da un unico RFC, ed ha una struttura lineare e relativamente semplice, IPsec al contrario ha un'architettura molto complessa e formata da elementi diversi (i protocolli AH, ESP e IKE[1], i database SPD e SAD) specificati in documenti diversi. Questa grande complessità determina probabilmente varie difficoltà nell'implementazione e nell'utilizzo di IPsec, ed è secondo alcuni (si veda [11]) una debolezza anche dal punto di vista della sicurezza crittografica.

Le funzionalità "ad alto livello" sono abbastanza simili: sia IPsec che TLS offrono servizi di riservatezza, integrità e autenticazione, e nessuno dei due offre il non ripudio[2]. Inoltre, sono entrambi flessibili per quanto riguarda la scelta degli algoritmi di cifratura e di hash, ma IKE di IPsec offre diversi meccanismi di autenticazione dell'identità dell'interlocutore mentre TLS utilizza solo i certificati. D'altra parte TLS permette anche un'autenticazione asimmetrica dell'identità, ovvero permette di autenticare il server senza autenticare il client, mentre IKE di IPsec, essendo un protocollo peer–to–peer e non client–server, non fornisce questa possibilità per cui o si autenticano entrambi gli interlocutori oppure non si autentica nessuno dei due. Descrivendo la sequenza delle trasformazioni applicate da TLS si è visto inoltre che è esplicitamente prevista la compressione dei dati prima della cifratura: questa funzione non è propriamente presente in IPsec, ma è comunque realizzabile attraverso il protocollo IPComp [9]. Vale la pena di notare che la sequenza delle operazioni di cifratura e autenticazione è diversa nei due casi: in TLS prima si calcola il MAC e poi si cifra il tutto, MAC compreso, mentre in IPsec prima si cifra e poi si aggiungono i dati di autenticazione, che vengono quindi calcolati sul messaggio cifrato e non su quello in chiaro. Quello di TLS è in effetti l'ordine usuale in cui vengono applicate autenticazione e cifratura; la ragione per cui IPsec compie queste operazioni in ordine inverso è legata alle prestazioni: in questo modo infatti un pacchetto con un'autenticazione non valida viene

[1] Si veda la nota 4 a pagina 7.

[2] In entrambi i casi tutti i messaggi possono essere generati indifferentemente da uno qualsiasi dei due interlocutori, poiché entrambi sono in possesso delle chiavi segrete necessarie. Questo avviene perché l'uso della crittografia a chiave pubblica, per motivi di prestazioni, è limitato all'autenticazione delle parti nella fase di handshake, mentre per il resto della comunicazione si utilizzano algoritmi simmetrici.

scartato subito, senza doverlo prima decifrare. Quest'approccio è anche più robusto dal punto di vista degli attacchi di denial of service, anche se è controverso dal punto di vista crittografico (si veda ancora [11]).

Le differenze tra IPsec e TLS sono per la maggior parte una diretta conseguenza della loro diversa posizione all'interno dello stack TCP/IP. Innanzitutto bisogna considerare il diverso "punto terminale" della comunicazione sicura: TLS offre un canale sicuro tra due applicazioni, poiché opera al livello dei socket, mentre i terminali della comunicazione IPsec sono due macchine. Questo non significa che le security association di IPsec siano associate esclusivamente alle macchine: possono avere granularità più o meno fine, e in particolare possono essere associate ai socket e quindi alle applicazioni. Tuttavia l'elaborazione IPsec avviene comunque a livello IP, e quindi in un ambito relativo all'host e non all'applicazione: quando il traffico è arrivato nell'host di destinazione, IPsec ha finito il suo compito e quindi i dati non sono più protetti. Il discorso è analogo per quanto riguarda l'entità che viene autenticata: con TLS è naturale che sia l'applicazione o l'utente, mentre questo è un po' più complicato (per quanto possibile) con IPsec, attualmente più adatto ad autenticare la macchina.

Per quanto riguarda i protocolli di livello superiore, IPsec protegge tutto ciò che sta sopra IP, quindi ad esempio TCP, UDP, ICMP. TLS invece si deve appoggiare su un protocollo di trasporto affidabile, e quindi può essere utilizzato solo per proteggere traffico TCP; questa è una limitazione importante perché esclude ad esempio molte applicazioni real-time che viaggiano su UDP. TLS è adatto a proteggere la comunicazione tra due applicazioni, mentre IPsec può facilmente rendere sicuro tutto il traffico tra determinati host, o tra intere sottoreti.

IPsec cifra tutto ciò che segue l'header IP, e quindi anche l'header TCP, e questo può prevenire alcuni attacchi a basso livello (attacchi al TCP) o alcune analisi del traffico (in quanto non sono visibili dati come i numeri di sequenza, o le porte sorgente e destinazione). D'altre parte però, come si vedrà nel prossimo paragrafo, questo fatto può comportare anche alcuni problemi. Un'altra caratteristica di IPsec utile contro gli attacchi a basso livello è quella di proteggere i messaggi ICMP.

Riguardo ai numeri di sequenza, un semplice attacco a TLS è descritto in [12], ed è una conseguenza del fatto che questo opera al di sopra di TCP, il quale quindi non partecipa alle funzioni di sicurezza e non è al corrente delle operazioni crittografiche. Se un intruso inserisce nella comunicazione un pacchetto con un numero di sequenza valido, in ricezione TCP non si accorgerà di nulla per cui invierà il riscontro e passerà i dati a TLS. TLS si accorgerà che i dati non sono validi ma non potrà comunicarlo a TCP, il quale scarterà poi il pacchetto vero considerandolo un duplicato: a questo punto sarà necessario abbattere la connessione.

IPsec può garantire in una certa misura l'anonimato degli interlocutori rispetto a chi intercetta il traffico cifrato: sia durante l'handshake (se si utilizza IKE main mode) che durante il trasferimento gli unici dati resi pubblici sono gli indirizzi IP dei security gateway, e non c'è modo di sapere quali siano i due interlocutori. Il caso host–to–host, se si utilizza la modalità tunnel, non è distiguibile da quello gateway–to–gateway da parte di chi intercetti il traffico tra i due host (a meno di sapere con certezza che un dato indirizzo IP non corrisponde ad un security gateway). TLS è invece esclusivamente host–to–host, e dunque sono noti gli indirizzi IP dei due interlocutori e, come poco fa ricordato, al contrario di quanto avviene con IPsec qui sono note anche le porte, per cui sono in una certa misura noti i processi che stanno comunicando, e non semplicemente gli host. A ciò si aggiunge il fatto che durante l'handshake di TLS i certificati delle due parti (che rivelano molte più informazioni sull'identità rispetto al solo indirizzo IP) viaggiano in chiaro, mentre con IKE main mode sono cifrati.

Secondo alcuni (si veda [13]) SSL/TLS non è sicuro ed è vulnerabile ad attacchi di tipo man–

in–the–middle (MITM), tant'è che esistono programmi come dsniff[3] ed ettercap[4] che permettono di realizzarli. Tali attacchi si fondano però su comportamenti erronei dell'utente più che su debolezze del protocollo, in quanto si basano sul fatto che l'utente tenda ad ignorare degli avvertimenti che l'applicazione gli presenta, ad esempio nel caso in cui il certificato del server è firmato da un'autorità sconosciuta oppure quando il nome dell'host non coincide con quello indicato nel certificato. È però vero che TLS richiede una grossa partecipazione all'applicazione (che deve espressamente richiedere una connessione sicura, deve preoccuparsi di controllare il certificato fornito dall'interlocutore, deve controllare che i parametri negoziati siano abbastanza forti, eccetera), la quale in genere la richiede all'utente (sia in fase di configurazione dell'applicazione stessa che in fase di utilizzo): questo fatto, pur non essendo un difetto dal punto di vista della sicurezza crittografica, può comunque costituire un punto di debolezza del protocollo.

La situazione di IPsec da questo punto di vista è in parte differente, per almeno due ragioni. In primo luogo, IPsec opera al livello del sistema operativo e non dell'applicazione e in genere non interagisce con l'utente, per cui nel caso di sistemi non "casalinghi" (in cui cioè il sistema è configurato da un amministratore che si presume più esperto, e non dall'utente) il problema di un'errata configurazione e di politiche di sicurezza troppo deboli è minore. Inoltre, gli attacchi di tipo MITM a TLS/SSL colpiscono come detto la fase di scambio delle chiavi, la quale come già ricordato non fa propriamente parte di IPsec. Se si utilizza IKE secondo una modalità simile all'handshake di TLS, in cui ciascuna delle due parti invia un proprio certificato all'altra, tendenzialmente si è vulnerabili ad attacchi dello stesso tipo (tenendo però presente l'osservazione fatta poco fa sul rapporto con l'utente); tuttavia ci sono altre possibilità, quali l'utilizzo di DNSSEC [14] per reperire le chiavi pubbliche, oppure l'utilizzo di un diverso protocollo di scambio delle chiavi che si appoggi su un sistema di autenticazione robusto (quale ad esempio Kerberos[5], il cui uso non è però ipotizzabile nel caso generale di due host che non hanno alcun rapporto tra di loro).

In teoria si potrebbe utilizzare DNSSEC anche con TLS (anche se lo standard attuale del protocollo non lo prevede) ed è prevista un'estensione di TLS che permette l'uso di Kerberos [16], ma questo graverebbe comunque sull'applicazione, che dovrebbe essere riscritta per poterlo fare e dovrebbe preoccuparsi da sé di gestire il tutto. L'accoppiata IPsec–DNSSEC è generalmente considerata più sicura rispetto a TLS (si veda [17] o le "frequently asked questions" di dsniff), ma la sua diffusione su larga scala è ancora lontana, sia per DNSSEC in sé sia per IPsec [18], soprattutto per problemi relativi all'attuale protocollo IKE e alla sua grande complessità. Alcune possibili soluzioni attualmente allo studio relativamente all'instaurazione della connessione IPsec saranno trattate al capitolo 5.

4.2 Differenze di impatto

Se escludiamo implementazioni hardware, di cui non ci si occuperà in questa sede, l'utilizzo di IPsec richiede in genere l'aggiornamento del sistema operativo su tutte le macchine che devono fare elaborazioni IPsec, e questa può essere una difficoltà; tuttavia IPsec è parte integrante di IPv6 e sarà quindi in futuro integrato nell'implementazione TCP/IP dei vari sistemi operativi. TLS, da questo punto di vista, crea meno problemi perché è implementato all'interno dell'applicazione o come libreria. D'altra parte, come già evidenziato, TLS ha basso impatto sul sistema ma maggiore impatto sulle applicazioni, perché richiede esplicite modifiche in tutte i programmi che lo vogliono utilizzare; IPsec può essere invece

[3] Disponibile presso `http://www.monkey.org/~dugsong/dsniff`.

[4] Disponibile presso `http://ettercap.sourceforge.net`.

[5] A questo proposito, è attualmente allo studio il protocollo *KINK* [15], che permette di instaurare security association IPsec appoggiandosi appunto su Kerberos.

utilizzato in maniera totalmente trasparente, senza cioè dover modificare le applicazioni. Il problema di quest'ultimo approccio, come evidenziato in [12], è che in molti casi l'applicazione potrebbe non volere questa trasparenza perché potrebbe voler comunicare ad IPsec le proprie esigenze di sicurezza e controllare che siano soddisfatte dalle security association negoziate. Questo è più che altro un problema che riguarda le attuali implementazioni di IPsec, in quanto una tale interfaccia verso l'applicazione è espressamente prevista da [2], seppure solo come possibilità.

Come si è visto, con IPsec l'intero header TCP o UDP è cifrato, e non è nemmeno possibile sapere se il protocollo di trasporto è TCP o UDP: questo impedisce qualsiasi elaborazione da parte di sistemi intermedi che necessitino di accedere a dati che si trovano oltre l'header IP[6], quali ad esempio i proxy e i firewall di tipo proxy server, oppure in alcune applicazioni che riguardano la qualità del servizio. TLS non presenta questo problema, e può essere utilizzato con i proxy. Una proposta che tenta di risolvere questo inconveniente è quella del *Multi-Layer IPsec* [19], che suddivide il pacchetto in varie "zone" cifrate con chiavi diverse. La soluzione proposta è tuttavia molto complessa e richiederebbe molte operazioni ai sistemi intermedi che vi partecipano (mantenere un complesso stato per ogni connessione, effettuare operazioni crittografiche di cifratura/decifratura e autenticazione, partecipare a protocolli di scambio delle chiavi — che tra l'altro diventano di tipo multicast, e non sono analizzati in [19]), contrastando in pieno una certa tendenza a spostare l'intelligenza ai bordi esterni della rete. Oltre a ciò, la proposta comporta la modifica dei protocolli del nucleo di IPsec (AH ed ESP, ma anche IKE), e questo al momento non è probabilmente molto realistico.

Un altro inconveniente di IPsec da cui TLS è invece immune, discusso in [20], riguarda il *NAT* (Network Address Translation): infatti AH autentica anche l'header IP che non può quindi essere modificato, ESP cifra l'header TCP rendendo inaccessibili i numeri di porta (utilizzati dal NAPT[7]), e IKE, per come utilizza gli indirizzi IP, non può attraversare il NAT. Per questo le operazioni del NAT devono necessariamente essere effettuate prima dell'elaborazione IPsec, per cui non deve essere applicato IPsec sugli host al traffico che verrà sottoposto al NAT, e la macchina che effettua il NAT deve stare anch'essa (come gli host) dietro il security gateway, oppure coincidere con esso. Con l'avvento di IPv6 comunque la principale motivazione del NAT, ovvero l'attuale scarsità di indirizzi IP con IPv4, verrà meno e quindi questo non dovrebbe costituire più un grosso problema, ma bisogna considerare il fatto che il NAT è utilizzato anche per ragioni di sicurezza, ovvero per nascondere la struttura interna di una rete e renderla inaccessibile dall'esterno. Il protocollo HIP, di cui si parlerà nel capitolo 5, risolve anche questo problema nell'ambito di comunicazioni host–to–host.

[6] Se si usa la modalità tunnel non è nemmeno possibile accedere all'header IP originale.

[7] Nel NAPT (Network Address Port Translation) si utilizzano i numeri di porta per distinguere le connessioni. Questo è il modo in cui viene effettuato il *masquerading* (nome utilizzato in ambiente Linux), in cui un certo numero di host viene mascherato dietro un solo indirizzo IP (quello dell'host che effettua il masquerading).

IPsec	TLS
architettura complessa	singolo protocollo
peer–to–peer	client–server
livello rete	livello sessione
canale tra due macchine	canale tra due applicazioni
può proteggere tutto il traffico IP	può proteggere solo il traffico TCP
protegge tutto ciò che segue l'header IP	protegge i dati del livello applicazione
impatto maggiore sul sistema operativo	impatto maggiore sulle applicazioni
poco consolidato	consolidato e diffuso

Tabella 4.1: Principali differenze tra IPsec e TLS.

5 L’instaurazione della connessione IPsec

IPsec è un protocollo versatile ed è parte integrante della nuova versione del protocollo IP (IPv6), per cui si può considerare come la soluzione più generale per la protezione (cifratura/integrità/autenticazione) dei dati in transito su Internet. Tuttavia, come già detto nel capitolo 2, attualmente IPsec è utilizzato più che altro per costruire reti private virtuali, con configurazioni essenzialmente statiche. È importante notare che parlando di connessioni configurate staticamente non ci si riferisce qui ad un’impostazione manuale delle chiavi *segrete* usate da AH ed ESP (senza quindi alcun bisogno di un handshake), ma al caso in cui viene preimpostata la connessione (tipicamente un tunnel) tra due macchine, configurando su ciascuna di esse l’indirizzo IP (o il nome) e la chiave *pubblica* dell’interlocutore (dopo di che si utilizza IKE per effettuare l’handshake e negoziare le chiavi segrete). IKE è un protocollo versatile e offre molte possibilità e opzioni, tuttavia è anche un protocollo molto complesso, e forse poco adatto a situazioni in cui le connessioni devono essere create in maniera dinamica.

Il problema dell’instaurazione della connessione IPsec è quindi ancora aperto, per cui si analizzeranno di seguito alcune proposte che lo affrontano in diversi scenari di utilizzo: uno che coinvolge un server centrale ed uno peer–to–peer; nell’ultimo paragrafo verrà poi descritto un approccio diverso e molto più ampio al problema, ovvero il protocollo *HIP (Host Identity Payload)*. Per prima cosa si descriveranno però brevemente le estensioni di sicurezza del DNS (*DNSSEC*), che hanno un ruolo importante in alcuni delle proposte di seguito analizzate.

5.1 Il DNS sicuro: DNSSEC

Le estensioni di sicurezza del Domain Name System, note come *DNSSEC* [14], sono pensate per rendere sicure (ovvero autenticate) le risposte che si ottengono dai server DNS, per evitare attacchi di tipo DNS spoofing. Con DNSSEC si costruisce una “catena di fiducia” utilizzando la crittografia a chiave pubblica, per cui ogni zona firma le chiavi delle proprie zone di livello inferiore (ovvero *it* può firmare *xyz.it*, che a sua volta può firmare *abc.xyz.it*).

DNSSEC aggiunge alcuni resource record al DNS, ovvero:

KEY: contiene una chiave pubblica, associata ad una zona DNS, ad un host o ad un utente (ogni risorsa può essere associata a più di un record KEY). Oltre alla chiave, il record contiene dei campi che indicano il relativo algoritmo crittografico e il protocollo per cui la chiave può essere utilizzata; attualmente gli utilizzi possibili sono DNSSEC, IPsec, TLS e la posta elettronica.

SIG: permette di immagazzinare firme digitali, ottenute mediante crittografia a chiave pubblica. Un server DNSSEC, rispondendo ad un’interrogazione, oltre ai record richiesti fornirà anche i corrispondenti record SIG, in modo da autenticare i dati forniti.

NXT: permette di autenticare l’inesistenza di una risorsa. Questo record è associato ad un nome esistente e fornisce il successivo nome esistente, dichiarando l’inesistenza di qualsiasi nome tra i due (ciò implica che i record debbano essere ordinati).

CERT: permette di immagazzinare dei certificati, ad esempio X.509 e PGP. Il record KEY (che contiene solo una chiave pubblica, non un certificato) non è destinato ad immagazzinare chiavi pubbliche "personali", per le quali si utilizza quindi CERT (questo record è definito in [21]).

Al di là dell'uso connesso alla sicurezza del DNS, ciò che importa ai fini dell'instaurazione della connessione IPsec è il fatto che con DNSSEC si può utilizzare il DNS come sistema per la distribuzione di chiavi pubbliche, immagazzinate nei record KEY e autenticate tarmite i record SIG.

DNSSEC è implementato nella versione attuale del server DNS BIND[1] (la 9.1), ma il suo utilizzo non è ancora diffuso, anche perché risolve il problema della sicurezza ma ne apre di nuovi. Infatti comporta un maggiore carico computazionale (per via della crittografia a chiave pubblica) e una maggiore quantità di dati da immagazzinare, inoltre non è ancora ben chiaro come gestire tutte le chiavi pubbliche. Per una discussione su DNSSEC e sulle problematiche relative si veda [22].

5.2 Handshake in sistemi centralizzati: il protocollo KINK

Il protocollo IKE, come già ricordato, è un protocollo peer–to–peer, e potrebbe non essere la soluzione più adatta in ambienti (quali ad esempio una intranet) dove esiste qualche forma di gestione centralizzata. Con un protocollo peer–to–peer è infatti necessario che ogni host conosca e implementi le politiche di sicurezza locali, inoltre IKE utilizza meccanismi quali Diffie–Hellman e le firme digitali, che sono computazionalmente piuttosto onerosi. Essenzialmente, come evidenziato in [23], si potrebbe dire che l'uso di un protocollo peer–to–peer comporta problemi con complessità $O\left(n^2\right)$, mentre utilizzando un server centrale la complessità è $O\left(n\right)$.

È attualmente allo studio all'IETF il protocollo *KINK* (Kerberized Internet Negotiation of Keys) ([23] e [15]), che utilizza Kerberos [24] per costruire e gestire delle security association per IPsec. KINK è un protocollo centralizzato di gestione delle chiavi e può quindi essere utilizzato al posto di IKE, ma è importante notare che non è in grado di sostituire completamente (cioè in ogni situazione) IKE, il quale permette a due interlocutori qualsiasi di autenticarsi vicendevolmente senza l'intervento di una terza parte. È progettato per essere più veloce e leggero rispetto ad IKE, sia per il numero di messaggi scambiati sia per il carico computazionale (riducendo al minimo le onerose operazioni di crittografia asimmetrica), e permette una gestione centralizzata delle politiche di sicurezza.

KINK utilizza i dati forniti da Kerberos per effettuare uno scambio simile alla fase 2 di IKE, in modo da negoziare una security association in soli due o tre messaggi, mentre non prevede nulla di analogo ad uno scambio IKE fase 1. Ha una struttura simile ad ISAKMP (che anzi in parte utilizza) nel prevedere un insieme di payload con diverse funzioni che vengono poi combinati per costruire i messaggi del protocollo. KINK non richiede alcuna modifica né ad IPsec né ad IKE; attualmente non esistono sue implementazioni, se non a scopi sperimentali connessi alla sua definizione.

5.3 Connessioni gateway–to–gateway: la opportunistic encryption

La *opportunistic encryption* è una caratteristica dell'implementazione FreeS/WAN per Linux di IPsec, ed ha lo scopo di proteggere la maggior parte possibile del traffico Internet. È ancora in fase sperimentale, sia per quanto riguarda l'implementazione che per quanto riguarda la definizione e la specifica [25].

[1] Con eccezione del record CERT.

Questa procedura permette a due host qualsiasi che la supportano di instaurare una connessione IPsec, senza che essi debbano avere una qualsiasi conoscenza reciproca precedente, utilizzando DNSSEC per reperire la chiave pubblica dell'interlocutore. Anche se può essere utilizzata in un ambito end–to–end, la opportunistic encryption sembra adatta soprattutto ad uno scenario di tipo gateway–to–gateway, instaurando sostanzialmente delle connessioni dinamiche simili a VPN.

L'algoritmo è descritto in dettaglio in [26], comunque il funzionamento in linea generale è il seguente:

1. L'iniziatore (security gateway della sorgente) intercetta il pacchetto in uscita, e fa un'interrogazione al DNS sull'indirizzo di destinazione chiedendo un record TXT che contenga l'indirizzo del security gateway (ricevitore) associato alla destinazione[2]. L'iniziatore chiede poi al DNS un record KEY associato al ricevitore, ottenendone così la chiave pubblica[3].

2. L'iniziatore inizia uno scambio IKE fase 1 con il ricevitore.

3. Il ricevitore riceve il primo messaggio, risponde, e nel frattempo interroga il DNS chiedendo un record KEY associato all'iniziatore per ottenerne la chiave pubblica (che verrà utilizzata nel corso dello scambio IKE fase 1).

4. Al termine dello scambio IKE fase 1, l'iniziatore inizia uno scambio IKE fase 2 (quick mode) con il ricevitore.

5. Il ricevitore, dal primo messaggio del quick mode, viene a conoscenza dell'indirizzo della sorgente. A meno che questo non coincida con quello dell'iniziatore, il ricevitore interroga il DNS chiedendo un record TXT associato alla sorgente, per verificare che l'iniziatore sia autorizzato a rappresentarla.

6. Al termine dello scambio IKE fase 2, la security association IPsec è stata negoziata e la comunicazione protetta può procedere.

La opportunistic encryption non è implementata nella versione 1.9 di FreeS/WAN, ed è stata inclusa solo a partire dalla versione 1.91 (giugno 2001): per le prove di laboratorio che sono state condotte per verificarne le funzionalità si sono utilizzati alcuni snapshot sperimentali di sviluppo successivi alla release 1.9 (per le misurazioni sull'handshake, che verrano illustrate nel capitolo 6, si è utilizzata la versione 1.91). Si sono utilizzate solamente due macchine (quindi in un'ottica end–to–end e non gateway–to–gateway), installando su una di esse anche un server DNS; le macchine sono state lasciate collegate alla rete locale per vedere l'effetto di altro traffico in entrata e uscita.

Un problema evidenziato è il seguente: con la opportunistic encryption il sistema intercetta ogni pacchetto in uscita e invoca il demone IKE per cercare di instaurare una connessione IPsec. Per questo deve però interrogare il server DNS, e quindi generare altri pacchetti in uscita, i quali vengono intercettati e così via. Il problema è risolvibile ad esempio installando manualmente una route in chiaro verso il server DNS[4], oppure inserendo una regola che esclude dall'elaborazione IPsec il traffico DNS[5] (per

[2] L'uso del record TXT è considerato temporaneo, in attesa che venga definito un record apposito.

[3] La chiave può essere inserita anche nel record TXT, per evitare una doppia interrogazione al DNS.

[4] Questo se la macchina è configurata per interrogare un certo server DNS. Se, come nel caso esaminato in laboratorio, il server DNS è sulla macchina stessa questo dovrà comunicare con *molti* altri server DNS e dunque la situazione appare più problematica.

[5] Questo è fattibile discriminando in base al numero di porta. Tale possibilità è prevista da [2], ma non è attualmente implementata in FreeS/WAN.

autenticare le risposte del DNS si usa DNSSEC[6], mentre la riservatezza in questo caso potrebbe non essere importante).

Un altro problema riscontrato durante i test è questo: quando un pacchetto viene intercettato, si installa una regola di tipo "hold" per la destinazione, il che significa che il pacchetto viene tenuto in attesa finché la negoziazione opportunistic non è terminata (con esito positivo o negativo), mentre in caso di arrivo di altri pacchetti per la stessa destinazione viene tenuto in attesa soltanto il più recente, scartando i pacchetti precedenti. Si è visto però che in condizioni di alto carico è possibile che non si ottenga risposta dal DNS, il che comporta che la regola di "hold" rimane al suo posto bloccando qualsiasi comunicazione con la destinazione. Il problema è stato segnalato al team di sviluppo di FreeS/WAN, ed è stato risolto inserendo un timeout associato a tali regole.

Quello che appare probabilmente il problema maggiore di questo approccio è comunque legato agli attacchi di *denial of service*: è infatti sufficiente inviare ad un gateway con opportunistic encryption dei pacchetti ICMP echo request (ovvero dei ping) con indirizzi mittente fasulli per costringere il gateway ad una query DNS *sincrona* per ognuno di questi indirizzi, e ad iniziare degli handshake IKE (con tutto il carico computazionale che comportano) nel caso da tali interrogazioni si ottengano le informazioni sufficienti per farlo. Questo rischio è notevole, anche in considerazione del fatto che ci si occupa di protocolli di sicurezza, e purtroppo il problema non sembra di facile soluzione a meno di cambiare profondamente i protocolli che entrano in gioco.

Dal punto di vista della definizione del protocollo, il formato dei pacchetti di IKE e la struttura dell'handshake non vengono intaccati. L'impatto di questa proposta, più che sulla definizione di IKE, ricade sulla sua implementazione: un demone IKE "opportunistic" deve infatti essere in grado di gestire le interrogazioni e le risposte DNS coinvolte nel processo (con i record KEY e TXT) e deve essere pronto a negoziare security association con qualunque altra macchina (il che può comportare grandi necessità di risorse per mantenere gli stati associati alle connessioni e grandi necessità di entropia per generare dati casuali quali i nonce). Per quanto riguarda quest'ultimo aspetto, che può rientrare anch'esso negli svantaggi di questo approccio dal punto di vista del denial of service, si vedrà nel prossimo paragrafo che il protocollo HIP è invece pensato con una maggiore attenzione in proposito.

5.4 Il protocollo HIP

Nei paragrafi precedenti si sono viste due possibili soluzioni per l'instaurazione di connessioni IPsec non preconfigurate; si tratta di proposte o di prototipi pensati per casi più o meno particolari, ma comunque non di approcci rivoluzionari, per cui l'impatto è abbastanza limitato. Una proposta più ampia, che permetterebbe tra le altre cose di risolvere alcuni problemi di IPsec in ambito end–to–end, è invece il protocollo *HIP* (Host Identity Payload), specificato in tre draft ([27], [28] e [29]). Nel prossimo paragrafo verranno illustrati i concetti base realtivi al protocollo, che verrà poi descritto in maniera un po' più particolareggiata (senza tuttavia entrare troppo nei dettagli, dato che si tratta di draft e quindi di specifiche soggette a cambiamenti).

5.4.1 Architettura e concetti generali

Come discusso in [30], negli ultimi anni a causa di diversi fattori quali la diffusione di indirizzi IP dinamici, l'uso di indirizzi privati e del NAT, si è persa la trasparenza end–to–end delle connessioni, e questo crea problemi tra l'altro ad IPsec.

[6] Questo non è stato però ancora implementato in FreeS/WAN.

Attualmente in Internet esistono due spazi di nomi: gli indirizzi IP e i nomi di dominio. I secondi sono collocati a livello applicazione e vengono perlopiù utilizzati prima di effettuare una connessione per ottenere il corrispondente indirizzo IP (per cui sono riferimenti indiretti agli indirizzi IP), dopo di che non ne rimane traccia nella comunicazione. Inoltre con i nomi di dominio non si può avere anonimato e la dinamicità è limitata per via dei tempi di aggiornamento del DNS. Gli indirizzi IP hanno invece una caratteristica che, se in passato è stata un punto di forza, oggi si rivela un elemento di debolezza: essi "confondono" due informazioni essenzialmente diverse, ovvero l'identità di un interfaccia di rete e il percorso da seguire per raggiungerla, mettendo cioè insieme le informazioni relative all'identità con quelle relative all'instradamento. A ciò si aggiunge il fatto, ricordato poco fa, che oggi l'informazione di identità associata agli indirizzi IP è resa "debole" dal largo uso di indirizzi dinamici (o di indirizzi privati mascherati tramite NAT), il che ha delle conseguenze negative anche in virtù del fatto che i livelli dello stack di rete superiori al terzo (che non hanno nulla a che fare con il routing) utilizzano tali indirizzi.

Con HIP si propone di inserire un nuovo spazio di nomi in Internet, chiamato *Host Identity* (HI), che permetterebbe di *disaccoppiare* il routing (ovvero il livello 3, dove si continuerebbe ad usare gli indirizzi IP) dai livelli superiori (dal trasporto all'applicazione, dove si userebbe la host identity). Questo nuovo spazio di nomi avrebbe caratteristiche crittografiche (trattandosi essenzialmente di chiavi pubbliche) e quindi potrebbe essere utilizzato per autenticare gli host con IPsec[7]. Uno stesso host può avere più HI, alcune pubbliche (ovvero registrate nel DNS[8]) e altre anonime; può inoltre "autocertificare" la propria identità oppure utilizzare un meccanismo per garantirla, quale DNSSEC, PGP, o i certificati X.509. Dato che la HI può avere formato e lunghezza variabili, è previsto l'uso di un *Host Identity Tag* (HIT) di 128 bit, costituito essenzialmente da un hash della HI[9] e anch'esso immagazzinato nel DNS (se pubblico). La HI non deve mai essere espressamente utilizzata dai protocolli di rete: viene esclusivamente immagazzinata (eventualmente) in server DNS o LDAP e passata durante l'handshake di HIP, mentre tutti i protocolli utilizzano esclusivamente lo HIT, come rappresentazione della HI[10].

La host identity potrebbe essere utilizzata con IKE come chiave pubblica, prendendo gli HIT anziché gli indirizzi IP come identità: in questo modo IKE potrebbe attraversare il NAT, e si potrebbe dunque utilizzare IPsec (con ESP in modalità trasporto) in presenza di NAT. Questo è possibile, e va tenuto presente perché comunque IKE mette a disposizione molte più opzioni, tuttavia ciò che si propone è un *nuovo protocollo*, che utilizzando la HI permetterebbe di sostituire IKE in un ambito end–to–end (ma *non* gateway–to–gateway) fornendo una procedura di handshake più leggera e veloce per instaurare security association IPsec ESP.

L'handshake del protocollo HIP è lungo solo quattro pacchetti (contro i nove di un handshake IKE con main mode e quick mode), e nel progettarlo si sono tenuti in considerazione i possibili attacchi di denial of service, cercando di "rovesciare" l'onere computazionale sull'iniziatore. Questo avviene permettendo al ricevitore di riutilizzare un pacchetto precostituito come risposta al primo messaggio, e dandogli la possibilità di formulare una specie di "indovinello", di difficoltà variabile, che l'iniziatore

[7] In [27] si dice che la host identity potrebbe essere in teoria anche qualcosa di diverso da una chiave pubblica, tuttavia soluzioni di questo tipo sono espressamente sconsigliate e non sono prese in considerazione dai draft, per cui in questa sede verranno ignorate.

[8] In un record KEY.

[9] È previsto anche un formato gerarchico dello HIT che, sempre all'interno dei 128 bit, comprende un campo *Host Assigning Authority*, a sua volta suddiviso in due livelli. Tale formato è raccomandato per le identità pubbliche.

[10] È prevista anche un'altra rappresentazione della HI, detta *Local Scope Identity* (LSI). Si tratta di un numero casuale di 32 bit che, come indicato dal nome stesso, ha un significato esclusivamente locale e non globale come lo HIT. La LSI ricopre sostanzialmente un ruolo simile a quello del SPI, con la differenza che ha un "tempo di vita" più lungo rispetto a quest'ultimo.

dovrà risolvere (quest'idea è esposta in [31]). È anche prevista una modalità "opportunistic" per HIP, con lo svantaggio di un rischio maggiore per il denial of service (anche se appare molto minore rispetto a quello evidenziato per la "opportunistic encryption" esaminata in precedenza).

Il protocollo HIP è pensato inoltre con attenzione alle problematiche relative alla mobilità, che con il disaccoppiamento dei livelli rete e trasporto possono essere affrontate più facilmente: la mobilità dell'iniziatore (o di uno qualsiasi dei due interlocutori a connessione stabilita) non presenta problemi particolari, e il disaccopiamento permette di mantenere le connessioni (TCP ed ESP) anche dopo un cambiamento di indirizzo, mentre per gestire quella del ricevitore si fa uso di un "rendezvous server" (si veda [27] per i dettagli). Infine, HIP non presenta problemi in presenza di NAT, ed è possibile fare masquerading utilizzando HIT e SPI anziché i numeri di porta, per cui si può utilizzare il masquerading anche con IPsec.

Ricapitolando, HIP comporta l'inserimento di un nuovo livello nello stack TCP/IP, posto tra i livelli rete e trasporto, come mostrato in figura 5.1. Con riferimento all'uso con IPsec end–to–end[11], i vantaggi di HIP si possono così riassumere:

- L'handshake è *breve*, poiche è composto da soli quattro messaggi anziché i nove di IKE main mode + quick mode (sei si utilizza aggressive mode + quick mode). Il numero di messaggi è pari a quello dell'handshake di TLS.

- Come nel caso di IKE, l'handshake è in parte *cifrato*, mentre quello di TLS è in chiaro.

- Il protocollo è piuttosto *semplice* e progettato per permettere il riutilizzo del codice da altri protocolli, ad esempio per quanto riguarda i resource record del DNS. Una delle più frequenti critiche ad IKE riguarda proprio la sua grande complessità.

- Come IKE è un protocollo *peer–to–peer* e non client–server, il che facilita l'instaurazione della connessione tra due macchine qualsiasi.

- Pur essendo peer–to–peer ammette l'esistenza di *identità "anonime"*, e dunque permette connessioni con autenticazioni "asimmetriche" (in qualche modo simili a quelle realizzabili con TLS) che possono essere utili in ambito client–server.

- L'introduzione della host identity, con il disaccoppiamento dei livelli rete e trasporto, permette di superare i problemi relativi all'*attraversamento del NAT* presenti in IKE.

- Il protocollo è progettato cercando di limitare i possibili attacchi di *denial of service*, che invece possono essere un problema per IKE (soprattutto nel caso "opportunistic").

- Pur avendo un impatto molto grande su tutto lo stack TCP/IP, HIP permette una *diffusione "dal basso"*, in quanto è utilizzabile da due sistemi che lo supportano indipendentemente dal fatto che sia supportato o meno dai sistemi intermedi che si trovano fra di essi (fatta eccezione per i NAT).

Ancora una volta, è comunque importante notare che, essendo molto più semplice di IKE, HIP non ha tutte le caratteristiche di questo, e dunque non lo può sostituire in ogni caso di utilizzo. In particolare, come già detto, non può essere utilizzato in ambito gateway–to–gateway. L'introduzione della host identity, ovvero il livello aggiuntivo tra IP e TCP, può comunque portare dei benefici anche ad IKE, ad esempio per quanto riguarda l'attraversamento del NAT.

[11] L'introduzione di HIP può portare benefici in altri campi, che però qui non verranno trattati.

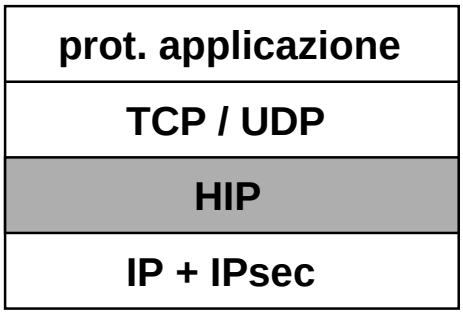

Figura 5.1: Posizione di HIP nello stack TCP/IP.

5.4.2 Formato e sequenza dei pacchetti

Il payload di HIP segue direttamente l'header IP[12]. Teoricamente potrebbe comparire in ogni pacchetto IP, tuttavia, dopo il completamento dell'handshake, le informazioni incluse in ESP sono sufficienti a mantenerne lo stato, per cui viene usato solamente nei pacchetti che servono a stabilire o modificare lo stato della connessione HIP.

Il formato del payload è illustrato in figura 5.2. Il primo campo indica il "next header" (con il significato usuale), il secondo contiene la lunghezza del "key payload", il terzo serve ad identificare il tipo di pacchetto HIP; seguono un campo che contiene il numero di versione e un campo riservato per usi futuri. I 128 bit successivi contengono lo HIT del mittente, segue il "key payload", e infine l'header ESP seguito dal payload IP[13].

Il key payload contiene le informazioni necessarie per portare a termine l'handshake, e il suo contenuto varia a seconda del tipo di messaggio. Il formato, illustrato in figura 5.3, è una semplificazione di un messaggio DNS e contiene infatti una sequenza di resource record (RR). Il campo RCOUNT contiene il numero di record presenti, FQDNLGTH fornisce la lunghezza del nome di dominio (Fully Qualified Domain Name), che è contenuto nel campo successivo insieme con un eventuale padding. Segue la serie dei resource record[14] (RDLENGTH contiene la lunghezza della parte RDATA e TYPE ne indica il tipo), che possono essere di tipo KEY (contengono le chiavi o gli HIT), SIG (contengono le firme) oppure OPT (contengono informazioni di altro tipo, quali cookie, SPI, LSI, informazioni sulle trasformazioni[15]).

L'handshake HIP, come già detto, è costituito da quattro pacchetti, e consente ai due interlocutori di scambiarsi le informazioni relative alle identità (intese come HI) e di stabilire una security association IPsec ESP. L'handshake comprende uno scambio di "cookie", che contrariamente a quanto avviene solitamente è iniziato dal ricevitore e non dall'iniziatore: questo aumenta di un messaggio la durata dell'handshake, ma offre il vantaggio di una maggiore robustezza verso attacchi di denial of service. Il primo pacchetto inviato dal ricevitore può essere infatti sostanzialmente precalcolato (in quanto contiene informazioni che non devono necessariamente cambiare ogni volta) e necessita di pochissime informazioni di stato da mantenere (ambedue le cose non sono vere per IKE), inoltre permette al ricevitore di formulare un "indovinello" di difficoltà variabile all'iniziatore: il ricevitore fornisce infatti un numero I di 64 bit, un numero K (che dovrebbe essere compreso tra 1 e 8) e un target, e l'iniziatore

[12] In alcuni casi, corrispondenti a pacchetti con funzioni particolari successive all'handshake, può seguire l'header ESP.
[13] HIP supporta ESP in modalità trasporto e non in modalità tunnel.
[14] Per i dettagli su di essi si vedano [32], [33] e [14].
[15] Queste ultime sono inserite utilizzando un formato preso in prestito da ISAKMP.

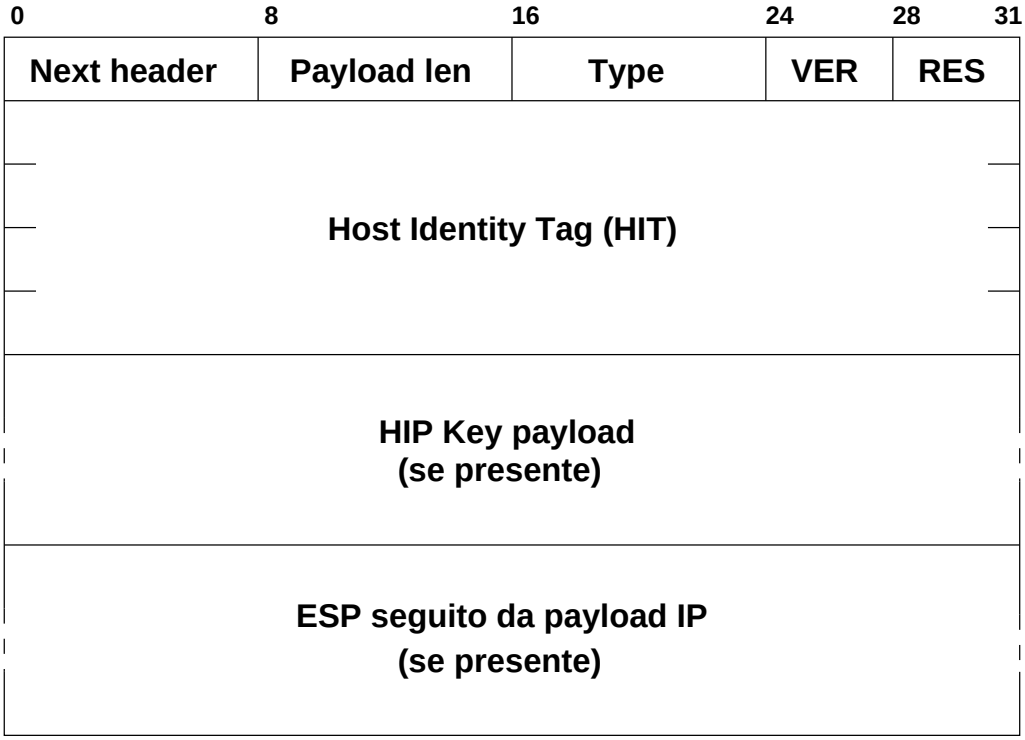

Figura 5.2: Formato del payload HIP.

deve trovare un numero J tale per cui i K bit meno significativi ottenuti dall'hash del concatenamento di I e J coincidano con i K bit meno significativi del target[16].

La struttura dell'handshake è la seguente: l'iniziatore ottiene indirizzo IP, HI e HIT del ricevitore dal DNS[17] e invia il primo pacchetto. Il ricevitore replica fornendo, tra l'altro, il valore per lo scambio Diffie–Hellman, il cookie (con l'indovinello) e le proprie proposte per le trasformazioni crittografiche per HIP ed ESP, e firmando il tutto. Nel terzo pacchetto della sequenza l'iniziatore inserisce il cookie (con la risposta all'indovinello), LSI e SPI che ha assegnato al ricevitore, il proprio valore per lo scambio Diffie–Hellman, la proposta scelta per le trasformazioni HIP e, cifrate, la proposta scelta per le trasformazioni ESP e la propria HI; infine firma il tutto. Nell'ultimo pacchetto (anch'esso firmato e in parte cifrato) il ricevitore inserisce tra l'altro LSI e SPI che ha assegnato all'iniziatore, completando così l'handshake: i pacchetti successivi saranno protetti dalla ESP SA negoziata.

Oltre ai quattro pacchetti dell'handshake, HIP ne prevede altri per cambiare chiavi e SPI, per notificare il cambio di indirizzo, e per il "bootstrap".

[16] Questo potrebbe richiedere all'iniziatore 2^K operazioni di hash prima di trovare un numero adatto, mentre la verifica da parte del ricevitore necessita di una sola operazione di hash.

[17] Per cui anche per HIP è importante DNSSEC. Eventualmente è possibile ottenere i dati anche da una tabella locale, il che in ambiti limitati può essere utile ma non è ovviamente praticabile su larga scala.

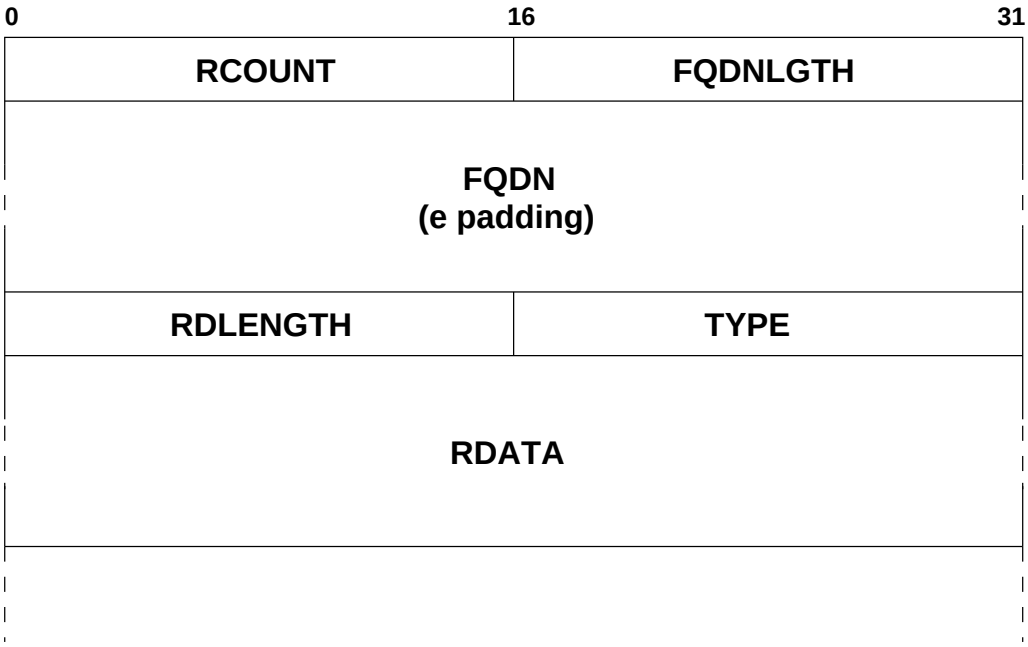

Figura 5.3: Formato dello HIP key payload.

6 Valutazione delle prestazioni

In questo capitolo verranno analizzate le prestazioni di IPsec e TLS, sulla base di considerazioni teoriche e soprattutto di risultati sperimentali. Per prima cosa si sono valutate le prestazioni di varie modalità d'uso di IPsec, successivamente si è effettuato il confronto tra le prestazioni di IPsec e quelle di TLS, e si sono valutati gli effetti della sovrapposizione tra i due. Si è poi considerato l'impatto della frammentazione dei pacchetti IP sulle prestazioni di IPsec, infine si sono valutati i tempi di handshake.

6.1 Architettura per le misurazioni

Per le misurazioni sui due protocolli (non quelle in presenza di frammentazione) si sono utilizzate due macchine (H1 e H2) con le seguenti caratteristiche hardware:

- CPU Intel Pentium III 450 MHz
- RAM 256 MB
- scheda di rete RealTek RTL-8139 (10/100 Mbit/s)

Le due macchine sono state collegate mediante un hub a 10 Mbit/s oppure mediante cavo incrociato (per le prove a 100 Mbit/s). Su tutte le macchine si è utilizzato il sistema operativo *Linux*, precisamente la distribuzione *Red Hat* 7.0 (kernel 2.2.16). Le implementazioni utilizzate sono *FreeS/WAN* versione 1.9 per IPsec e *OpenSSL* versione 0.9.6a per TLS.

Per le misurazioni concernenti gli effetti della frammentazione si sono usate anche altre due macchine (G1 e G2) con le seguenti caratteristiche hardware:

G1:

- CPU Intel Pentium 200 MHz
- RAM 128 MB
- scheda di rete Intel 82557 Ethernet Pro 100 (10/100 Mbit/s)
- scheda di rete 3Com 3c590 Vortex (10 Mbit/s)

G2:

- CPU Cyrix 6x86MX 200 MHz
- RAM 64 MB
- scheda di rete Intel 82557 Ethernet Pro 100 (10/100 Mbit/s)
- scheda di rete 3Com 3c590 Vortex (10 Mbit/s)

Le quattro macchine sono state collegate come mostrato in figura 6.1, ovvero H1 è stata connessa a G1 tramite cavo incrociato, e così pure H2 e G2, poi G1 e G2 sono state connesse tramite hub a 10 Mbit/s. Il sistema operativo utilizzato è stato ancora una volta Linux, con la distribuzione Red Hat 7.1 e il kernel 2.4.2 su H1, H2 e G1 e 2.4.6 su G2. Per IPsec si è utilizzato FreeS/WAN versione 1.9; si è poi utilizzato *NIST Net* versione 2.0.10 su G2, per poter simulare la presenza di una rete con una data probabilità di perdita.

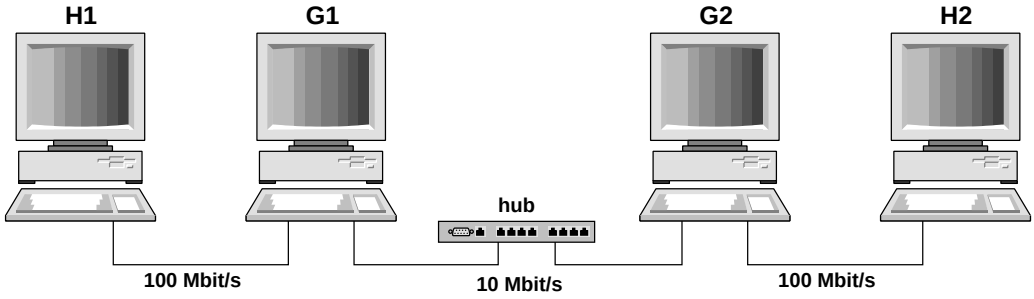

Figura 6.1: Rete di laboratorio utilizzata per i test sulla frammentazione.

6.2 Strumenti di misura

Le misurazioni sono state effettuate tramite il programma *netperf* (versione 2.2 alpha). Netperf permette di effettuare vari test sulle prestazioni di rete, relativamente a vari protocolli; si utilizza facendo partire su una macchina un demone (`netserver`) che rimane in ascolto in attesa di connessioni, e su un'altra il client (`netperf`), fornendo a quest'ultimo i parametri relativi al test da condurre. Qui si sono utilizzati i test "TCP stream" e "UDP stream", che funzionano in maniera analoga rispettivamente su TCP e UDP. Questi test trasmettono un flusso di dati tra le due macchine tale da saturare il sistema, continuando per un tempo dato[1]. Ciò che viene trasmesso è un blocco di dati che rimane in memoria, per evitare che i risultati siano influenzati dalle prestazioni dei dischi fissi. Netperf permette anche di specificare dei parametri di confidenza, e in base ad essi uno stesso test viene automaticamente ripetuto più volte (in numero compreso tra un valore minimo e un valore massimo che si forniscono al programma) fino a che i risultati non li soddisfano[2]. Con netperf si è specificata una confidenza del 99% con un ampiezza dell'intervallo pari al 2%, tuttavia in alcuni casi (in particolare in alcuni dei test sulla frammentazione) l'ampiezza dell'intervallo è risultata maggiore. Ogni test ha la durata di un minuto, ed è stato iterato da un minimo di cinque volte ad un massimo di trenta.

I parametri presi in considerazione sono il *throughput* (di seguito indicato con X), l'*utilizzo del processore* (U_{CPU}) e la *domanda di servizio del processore* (D_{CPU}), ovvero il tempo di CPU richiesto per l'elaborazione di un'unità di dati. Il throughput viene calcolato da netperf semplicemente dividendo il numero di byte trasferiti per la durata del test. L'utilizzo del processore viene determinato misurando l'occupazione di un processo nullo a bassa priorità che occupa tutto il tempo in cui la CPU rimane inattiva, e facendone poi il complementare. Non è possibile misurare direttamente l'occupazione della CPU da parte del processo che interessa, perché IPsec è implementato all'interno del kernel del sistema operativo e non è dunque un normale processo; per avere la massima affidabilità sui dati relativi all'u-

[1] Per il test su TCP è anche possibile specificare una dimensione totale del trasferimento anziché un tempo.

[2] Il programma effettua il test di Student. Se non si raggiunge la confidenza desiderata con il massimo numero di iterazioni, il programma fornisce comunque i risultati specificando la confidenza raggiunta.

tilizzo del processore i test sono stati condotti con le macchine in *single user mode*[3]. La domanda di servizio del processore è calcolata da netperf come $D_{CPU} = U_{CPU}/X$.

I test su IPsec non presentano problemi per via della trasparenza di IPsec alle applicazioni, per cui è suffciente instaurare manualmente la connessione IPsec tra le due macchine e poi effettuare i normali test "TCP stream" e "UDP stream". Per poter effettuare dei test su TLS è stato invece necessario modificare netperf, introducendo nel programma un test "TLS stream", che è sostanzialmente il test "TCP stream" modificato in modo da utilizzare TLS. Si è quindi modificata la funzione del programma[4] che effettua il test di flusso TCP, introducendovi le opportune chiamate di libreria ad OpenSSL per effettuare un handshake TLS all'inizio del test, e sostituendo tutte le normali chiamate in lettura/scrittura sui socket con quelle di OpenSSL.

6.3 Prestazioni di IPsec

Per prima cosa si sono analizzate le prestazioni di varie modalità di IPsec (per quanto riguarda la fase di trasferimento dei dati), ovvero:

- ESP in modalità trasporto
- ESP in modalità tunnel
- AH più ESP in modalità trasporto
- AH più ESP in modalità tunnel
- AH in modalità trasporto (trasmissione non cifrata)
- AH in modalità tunnel (trasmissione non cifrata)

Gli algoritmi utilizzati sono il triplo DES per la cifratura (con chiavi di 192 bit e vettore di inizializzazione di 64 bit) e SHA1 per l'autenticazione (che produce una stringa di 160 bit, poi troncata a 96 bit).

In tabella 6.1 sono illustrati i risultati relativi a TCP su IPsec su rete a 10 Mbit/s. Dai dati nella terza colonna si può notare che, in termini di carico sul processore, non vi è una grande differenza tra le varie modalità di IPsec, ovvero tra tunnel oppure trasporto e tra l'autenticazione con AH oppure con ESP, infatti lo scostamento tra il valore minimo e quello massimo, considerando anche il margine di approssimazione, è sostanzialmente nullo. Si ha invece una differenza notevole, come ci si poteva attendere, tra una connessione cifrata e una connessione solo autenticata: l'overhead sul processore nel primo caso è infatti circa triplo rispetto al secondo. In questo caso il collo di bottiglia è costituito dalla rete, per cui il throughput riflette il carico su di essa: i risultati ottenuti non danno grosse sorprese, e riflettono più o meno il diverso overhead sul pacchetto dato dalle diverse modalità, che precisamente è:

ESP trasporto: header ESP: 8 byte, IV: 8 byte, trailer ESP: 2 byte, autenticazione: 12 byte. Totale: 30 byte più padding.

ESP tunnel: come il precedente più 20 byte per l'header IP. Totale: 50 byte più padding.

[3] Nella modalità "a utente singolo" in Linux vengono eliminati pressoché tutti i servizi e i processi di sistema, ad eccezione di quelli fondamentali.

[4] Netperf è scritto in linguaggio C.

AH + ESP trasporto: header AH: 12 byte; autenticazione: 12 byte; header ESP: 8 byte; IV: 8 byte; trailer ESP: 2 byte. Totale: 42 byte più padding.

AH + ESP tunnel: come il precedente più 20 byte per l'header IP. Totale: 62 byte più padding.

AH trasporto: header AH: 12 byte; autenticazione: 12 byte. Totale: 24 byte.

AH tunnel: come il precedente più 20 byte per l'header IP. Totale: 44 byte.

Le differenze di overhead di rete tra le varie modalità, sia a livello teorico che a livello sperimentale, non sono molto grandi. Per una connessione non cifrata la scelta ovvia appare AH in modalità trasporto (dato che il tunnel non aggiunge nessuna funzione ad AH) ma nel caso gateway–to–gateway è necessario il tunnel. Nel caso cifrato la cosa più semplice è utilizzare il solo ESP, magari in modalità tunnel anche nel caso host–to–host in modo da autenticare anche l'header IP.

Test	X [KB/s]	U_{CPU} [%]	D_{CPU} [μs/KB]
chiaro	888,30	3,52	39,616
ESP trasporto	841,45	40,70	483,714
ESP tunnel	817,16	40,10	490,702
AH+ESP trasporto	821,14	39,59	482,112
AH+ESP tunnel	814,75	40,90	501,945
AH trasporto	862,02	15,53	180,112
AH tunnel	843,22	15,40	182,625

Tabella 6.1: Prestazioni TCP su IPsec con 3DES e SHA1 a 10 Mbit/s

Test	X [KB/s]	U_{CPU} [%]	D_{CPU} [μs/KB]
chiaro	1159,0	4,57	39,431
ESP trasporto	1136,4	48,47	426,546
ESP tunnel	1124,3	49,99	450,674
AH+ESP trasporto	1126,7	49,29	437,485
AH+ESP tunnel	1126,3	49,98	459,494
AH trasporto	1142,8	17,37	152,006
AH tunnel	1126,5	17,39	154,345

Tabella 6.2: Prestazioni UDP su IPsec con 3DES e SHA1 a 10 Mbit/s

In tabella 6.2 sono invece mostrati i dati relativi al test su UDP nelle stesse condizioni: anche in questo caso non si notano grosse differenze, per cui valgono le considerazioni del caso precedente.

Il test su UDP è tuttavia interessante perché in questo caso non entrano in gioco i meccanismi di ritrasmissione e controllo di flusso di TCP, per cui il sistema si può modellare semplicemente come la serie di due code rappresentanti il processore e la rete, come illustrato in figura 6.2. Per come funzionano i test di netperf che sono stati utilizzati il sistema viene saturato, per cui $\max(U_{CPU}, U_{NET}) = 1$. Questo significa che, se D_{CPU} e D_{NET} sono le domande di servizio del processore e della rete, e $D_{MAX} = \max(D_{CPU}, D_{NET})$, si ha che $X \simeq 1/D_{MAX}$. In questo caso il collo di bottiglia è la rete, per cui si può ipotizzare che $D_{NET} \simeq 1/X$. Si è allora provato a ripetere i test su rete a 100 Mbit/s (solo per alcune modalità): i risultati sono illustrati in tabella 6.3.

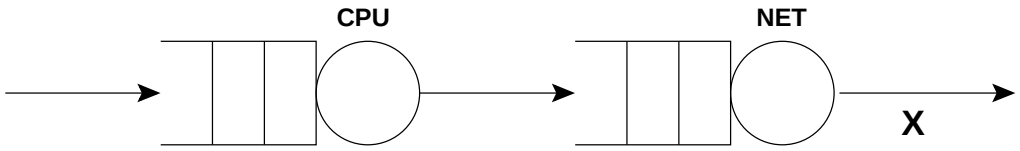

Figura 6.2: Modello a code per i test con UDP.

Test	X [KB/s]	U_{CPU} [%]	D_{CPU} [μs/KB]
chiaro	11619,5	32,09	27,623
ESP trasporto	2384,74	99,90	418,928
AH+ESP trasporto	2324,30	97,89	421,154

Tabella 6.3: Prestazioni UDP su IPsec con 3DES e SHA1 a 100 Mbit/s

Come si può vedere, in questo caso (a parte per la connessione in chiaro) il collo di bottiglia è costituito dal processore, infatti si è misurato $U_{CPU} \simeq 100\%$. A conferma di quanto detto in precedenza, si può verificare che in questo caso $X \simeq 1/D_{CPU}$. La domanda di servizio è uguale a quella misurata nel caso a 10 Mbit/s (tenendo conto delle imperfezioni nella misura che probabilmente si accentuano in caso di saturazione del processore), il che è corretto e convalida il metodo di misurazione del carico sul processore (in quanto in questo caso si ha una misura indiretta di D_{CPU} calcolabile dal throughput, la cui misurazione non presenta problemi).

Anche i dati sui test a 100 Mbit/s per TCP, riportati in tabella 6.4, confermano una domanda di servizio sul processore uguale a quella del caso a 10 Mbit/s. In questo caso però, per la presenza dei meccanismi di TCP citati sopra, non si può adottare il semplice modello prima proposto per cui il throughput risulta inferiore all'inverso della domanda di servizio del processore.

Test	X [KB/s]	U_{CPU} [%]	D_{CPU} [μs/KB]
chiaro	11489,81	47,92	41,712
ESP trasporto	2018,46	97,48	482,957
AH+ESP trasporto	2004,45	96,60	481,950

Tabella 6.4: Prestazioni TCP su IPsec con 3DES e SHA1 a 100 Mbit/s

6.4 Prestazioni di TLS

I test su TLS sono stati anch'essi effettuati con netperf, modificando il test "TCP stream" in modo da avere una connessione protetta con TLS, con la possibilità di avere una connessione cifrata e autenticata (usando triplo DES e SHA1) oppure solo autenticata (usando SHA1). Il test inserito in netperf è stato denominato come detto "TLS stream".

Nelle tabelle 6.5 e 6.6 sono illustrati i risultati dei test su TLS, rispettivamente su rete a 10 Mbit/s e 100 Mbit/s. Si nota che per quanto riguarda il carico sul processore TLS risulta più leggero di IPsec, in particolare per una connessione non cifrata. Quest'ultima si può considerare molto leggera addirittura rispetto ad una connessione in chiaro, tant'è vero che anche su rete a 100 Mbit/s il collo di bottiglia è dato ancora dalla rete e non dal processore. TLS si rivela leggero anche per quanto riguarda il carico sulla rete, infatti nel caso a 10 Mbit/s (dove è la rete il collo di bottiglia) il throughput è sostanzialmente uguale a quello della connessione in chiaro. Questo è dovuto al basso overhead in termini di

byte trasmessi imposto da TLS, essenzialmente per il fatto che opera su blocchi più grandi rispetto ai pacchetti IP. Questo tema, insieme con l'analisi di possibili miglioramenti alle prestazioni di IPsec in questo ambito, sarà discusso al paragrafo 6.6.

Test	X [KB/s]	U_{CPU} [%]	D_{CPU} [μs/KB]
chiaro	888,30	3,52	39,616
cifratura e autenticazione	889,93	31,43	353,136
solo autenticazione	886,10	6,64	74,984

Tabella 6.5: Prestazioni TLS con 3DES e SHA1 a 10 Mbit/s

Test	X [KB/s]	U_{CPU} [%]	D_{CPU} [μs/KB]
chiaro	11489,81	47,92	41,712
cifratura e autenticazione	2804,87	99,70	355,461
solo autenticazione	11465,58	84,03	73,286

Tabella 6.6: Prestazioni TLS con 3DES e SHA1 a 100 Mbit/s

6.5 Sovrapposizione tra TLS e IPsec

Si è provato a sovrapporre TLS ad IPsec: si è effettuato un test "TLS stream" (con e senza cifratura) sopra una connessione protetta da IPsec (tunnel ESP con cifratura 3DES e autenticazione SHA1). Il motivo di questo test è che i due protocolli potrebbero realisticamente sovrapporsi, ad esempio nel caso di una connessione TLS che nel transito in rete passa attraverso un tunnel IPsec. I risultati sono riportati in tabella 6.7.

Test	X [KB/s]	U_{CPU} [%]	D_{CPU} [μs/KB]
chiaro	888,30	3,52	39,616
cifratura+autenticazione	822,88	66,25	805,132
solo autenticazione	825,84	43,61	528,117

Tabella 6.7: Prestazioni TLS con 3DES e SHA1 sopra IPsec (tunnel ESP con 3DES e SHA1) a 10 Mbit/s

Per quanto riguarda il carico di rete, si conferma il bassissimo impatto di TLS da questo punto di vista, infatti il throughput è sostanzialmente quello di IPsec. Per quanto riguarda il carico sul processore i risultati ottenuti indicano che le domande di servizio dei due protocolli si sommano: se infatti si aggiunge alla domanda di servizio di IPsec quella di TLS e si sottrae quella della connessione in chiaro (che altrimenti verrebbe contata due volte) si ottengono i risultati qui osservati. Infatti per una connessione TLS cifrata e autenticata si può calcolare $D_{CPU} \simeq 490{,}7 + 353{,}1 - 39{,}6 = 804{,}2$ KB/s (con un dato sperimentale pari a $805{,}1$ KB/s), mentre per una connessione TLS solo autenticata si può calcolare $D_{CPU} \simeq 490{,}7 + 73{,}3 - 39{,}6 = 524{,}4$ KB/s (con un dato sperimentale pari a $528{,}1$ KB/s).

Un'altra conferma che viene da questi dati è il bassissimo overhead (sia sulla rete che sul processore) causato da una connessione TLS non cifrata, anche quando questa si trova sopra ad una connessione IPsec. Questo risultato è interessante perché mostra che dal punto di vista prestazionale non ci sono controindicazioni o "effetti collaterali" nell'uso congiunto dei due protocolli. Si può quindi ipotizzare uno scenario di questo tipo: è possibile utilizzare una connessione IPsec end–to–end e sopra di essa

un'autenticazione TLS. In questo modo si può sfruttare una connessione IPsec esistente (o si può instaurare appositamente, per sfruttare i vantaggi di IPsec), utilizzando al di sopra di essa l'autenticazione di TLS, tramite la quale è attualmente più semplice autenticare l'applicazione oppure l'utente anziché la macchina.

6.6 L'impatto della frammentazione sulle prestazioni di IPsec

Le differenze rilevate sperimentalmente nel carico sul processore di IPsec e TLS possono sicuramente dipendere almeno in parte dalle particolari implementazioni utilizzate, anche se qualitativamente si può ipotizzare un maggior peso di IPsec dovuto al fatto che in questo caso le operazioni di cifratura e autenticazione riguardano porzioni maggiori dei pacchetti IP (e quindi sono effettuate su una maggiore quantità di dati). Per quanto riguarda il carico sulla rete si può tuttavia affermare che IPsec è intrinsecamente meno efficiente di TLS. Quest'ultimo infatti, trovandosi al di sopra di TCP, opera su un flusso unico di dati, che per le operazioni crittografiche può essere suddiviso in blocchi abbastanza grandi (precisamente fino a 16 KB). IPsec opera invece necessariamente a livello dei singoli pacchetti IP, che ad esempio nel caso di una rete Ethernet sono al massimo di 1500 byte. Questo significa che l'overhead in termini di byte (dovuto in particolare al vettore di inizializzazione e ai dati di autenticazione) è maggiore nel caso di IPsec perché a parità di dati totali trasferiti il numero di blocchi è maggiore.

Per risolvere questo inconveniente, in uno scenario d'uso end–to–end (non gateway–to–gateway), si è pensato di costruire pacchetti IP più grandi, applicarvi IPsec e poi *frammentarli*. Questo provoca un certo overhead computazionale sui punti terminali della comunicazione dovuto alle operazioni di frammentazione e deframmentazione, ma questo non è un problema dato che ci troviamo in un caso end–to–end; non si impone invece nessun overhead sui sistemi intermedi. Il problema di questo approccio è che, per come funziona il protocollo IP, la perdita di un solo frammento comporta la necessità di scartare *tutti* i frammenti dello stesso pacchetto, e quindi la perdita (ed eventualmente la ritrasmissione, nel caso di TCP) dell'intero pacchetto.

6.6.1 Analisi teorica

La situazione può essere vista nel modo seguente. Siano L_{OF} l'overhead in byte presente in ogni frammento (ovvero quello dovuto all'header IP), L_{OP} l'overhead in byte presente in ogni pacchetto ma non ripetuto sui singoli frammenti (ovvero quello dovuto ad IPsec ed eventualmente, a seconda del livello a cui ci si pone, quello dovuto al protocollo di trasporto), $L_U(n)$ la lunghezza dei dati "utili" (payload) contenuti in un pacchetto costituito da n frammenti e M la *Maximum Transfer Unit* (MTU) della rete attraversata. In un pacchetto non frammentato e di grandezza massima si avrà allora $L_{OF} + L_{OP} + L_U(1) = M$. Se ipotizziamo che tutti i frammenti abbiano la grandezza massima, per un pacchetto suddiviso in n frammenti si avrà dunque $nL_{OF} + L_{OP} + L_U(n) = nM$. Se si considera come misura di efficienza (E) il rapporto tra dati utili e dati totali trasmessi si ha allora:

$$E(n) = \frac{L_U(n)}{nM} = \frac{nM - nL_{OF} - L_{OP}}{nM} = 1 - \frac{L_{OF}}{M} - \frac{1}{n}\frac{L_{OP}}{M}$$

In base alla formula precedente, l'efficienza cresce al crescere di n, tendendo al valore di $1 - \frac{L_{OF}}{M}$; l'overhead sul frammento non è eliminabile e dunque dà luogo al termine $-\frac{L_{OF}}{M}$, mentre l'overhead sul pacchetto rientra nel termine $-\frac{L_{OP}}{nM}$ e dunque la sua incidenza diminuisce all'aumentare di n. Questo discorso tuttavia non tiene conto della probabilità di perdita della rete. Sia p la probabilità di perdita del singolo frammento; allora, se il pacchetto è suddiviso in n frammenti, la probabilità che nessuno di

essi venga perso è $(1-p)^n$, e dunque la probabilità di perdita di un pacchetto (cioè la probabilità che si perda almeno un frammento) è $1-(1-p)^n$, pari a p per $n=1$ e tendente ad 1 per n tendente ad infinito.

Rivedendo allora in base a queste considerazioni la formula dell'efficienza, si può dire che i byte utili saranno mediamente $L_U\,(1-p)^n$ per ogni nM byte trasmessi, e che dunque l'efficienza diventa

$$E\,(n,p) = (1-p)^n \left(1 - \frac{L_{OF}}{M} - \frac{1}{n}\frac{L_{OP}}{M}\right)$$

Si considerino i seguenti valori per i parametri: $M = 1500$ byte (Ethernet), $L_{OF} = 20$ byte (lunghezza header IP) e $L_{OP} = 84$ byte (IPsec in modalità tunnel ESP con 3DES e SHA: 32 byte di overhead dovuti ad ESP[5], 20 byte per l'header IP interno e 32 per l'header TCP[6]). Con questi dati, si ottiene la funzione mostrata nelle figure 6.3 e 6.4[7].

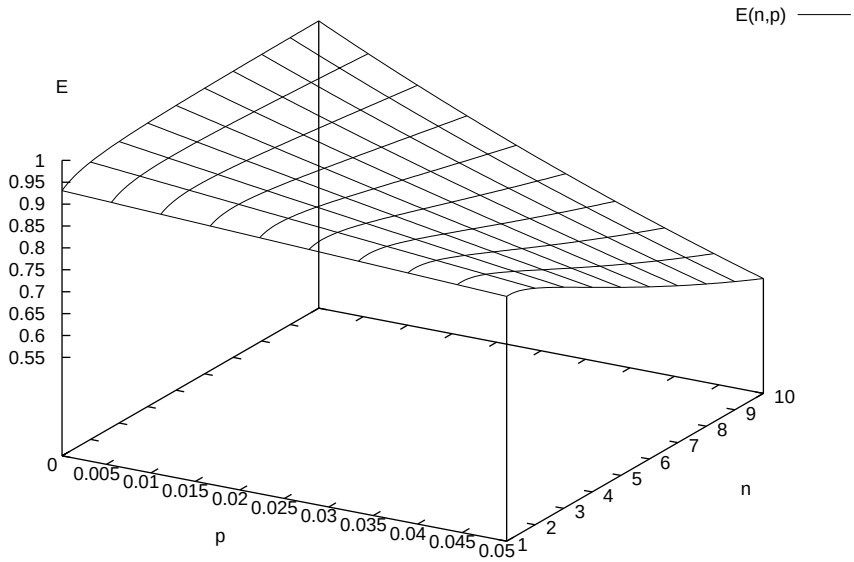

Figura 6.3: Rappresentazione tridimensionale della variazione dell'efficienza teorica E in presenza di frammentazione, in funzione della probabilità di errore sul frammento p e del numero di frammenti n.

Da questo semplice modello si possono trarre alcune considerazioni:

- Nel caso di una rete completamente affidabile (ovvero con probabilità di errore nulla) con i dati considerati l'efficienza passa da un valore di circa 0,931 per $n=1$ ad un valore limite di circa 0,987 per $n \to \infty$.
- Sempre con i dati considerati, non sembra opportuno utilizzare un valore di n maggiore di 5, in quanto con $p=0$ l'efficienza è comunque già abbastanza vicina al valore asintotico e cresce

[5] 8 per l'header, 8 per l'IV, 2 per il trailer, 12 per i dati di autenticazione, in più si è considerato un padding medio di 2 byte.
[6] Sono 32 e non 20 poiché si è utilizzata l'opzione di timestamp, che su Linux è abilitata per default.
[7] In entrambi i casi, per maggiore chiarezza della rappresentazione grafica, la funzione è mostrata come se le due variabili fossero continue, ma naturalmente ha senso solo per valori interi di n.

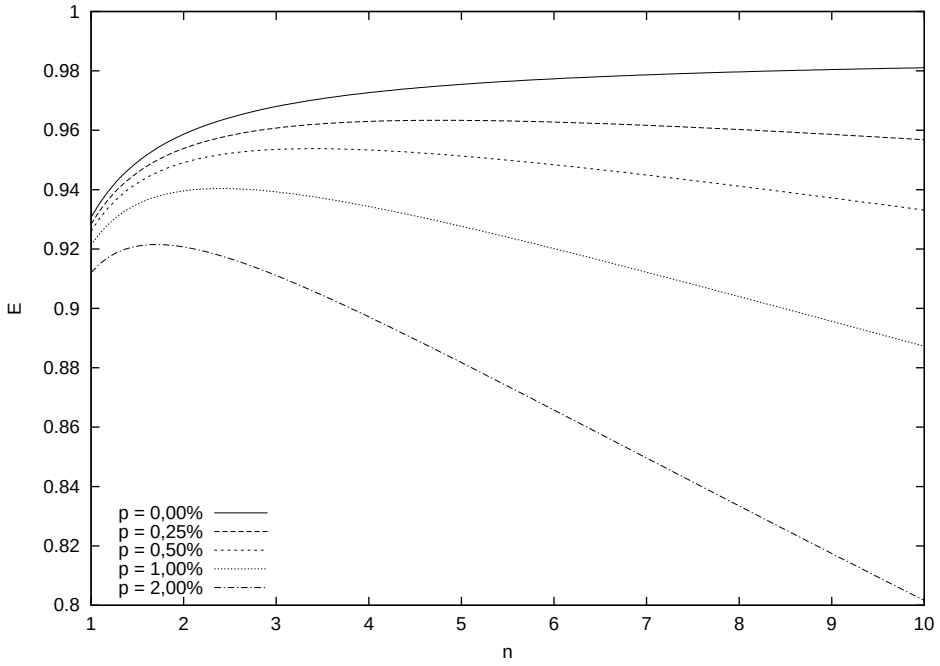

Figura 6.4: Variazione dell'efficienza teorica E in presenza di frammentazione, in funzione del numero di frammenti n, per alcuni valori della probabilità d'errore sul frammento p.

ormai molto lentamente (si ha infatti $E(4) \simeq 0{,}968$, $E(5) \simeq 0{,}973$ e $E(6) \simeq 0{,}975$), mentre al crescere di p trasmettere un pacchetto suddiviso in un gran numero di frammenti diventa sempre più rischioso (come si può vedere dalle figure).

- La frammentazione sembra vantaggiosa per valori di p non superiori a circa il 2%, in quanto per probabilità di perdita più elevate la curva $E(n)$ assume subito una pendenza negativa.

Ciò che si vede è dunque che, per quanto riguarda IPsec, in condizioni di bassa probabilità di errore la frammentazione end–to–end dei pacchetti dovrebbe portare ad un aumento delle prestazioni. TLS, operando sul flusso continuo di dati al di sopra di TCP, non ha invece nulla a che vedere con la frammentazione dei pacchetti IP ed è dunque estraneo a questo discorso[8].

La realtà è comunque naturalmente più complessa del semplice modello proposto, in quanto esso non tiene conto di almeno due fattori:

- Nel modello si è ignorata la presenza dei riscontri (acknowldgement, o brevemente ack). Gli ack sono puro overhead, in quanto non trasportano payload, ma contribuiscono ad occupare la banda. La frammentazione ha un effetto positivo relativamente alla loro presenza, perché dato che si trasmette un ack per ogni pacchetto ricevuto (e non per ogni frammento), il numero di ack trasmessi in presenza di pacchetti suddivisi in n frammenti è pari ad un ennesimo del numero di quelli trasmessi nel caso di pacchetti non frammentati. Questo riduce l'overhead e, nel caso di una Ethernet, riduce anche la probabilità di collisione poiché riduce il numero di pacchetti che

[8] In realtà si potrebbe notare che il modello si applica comunque a TCP, e in questo senso la frammentazione potrebbe portare ad un aumento di prestazioni di TLS come conseguenza indiretta dell'aumento di prestazioni di TCP. In questo caso si ha però che $L_{OP} = 32$ byte (contro gli 84 byte di IPsec) per cui il guadagno di efficienza è nettamente minore (per $p = 0$ si trova $E(1) \simeq 0{,}965$, mentre il livello asintotico è lo stesso di IPsec) e appare trascurabile.

fluiscono in senso opposto. Tali effetti hanno un impatto positivo sulle prestazioni, ed essendo dipendenti da $1/n$ ci si può attendere che siano significativi al crescere di n soprattutto per n non molto grande, andando poi incontro ad una sostanziale "saturazione". Per quanto riguarda la probabilità di perdita, gli ack possono essere persi come tutti gli altri pacchetti (la loro probabilità di perdita è pari a p poiché non sono frammentati), tuttavia, dato che gli ack di TCP sono cumulativi, in genere la perdita di un ack non ha effetti e non comporta una ritrasmissione, poiché i dati da esso riscontrati saranno considerati riscontrati dall'ack successivo.

- Il modello considera la trasmissione sostanzialmente come un flusso unico di dati dal mittente al destinatario, in cui alcuni dati possono essere ritrasmessi per via della probabilità di perdita della rete. Le cose non stanno così poiché nella realtà entrano in gioco i complessi meccanismi di controllo di flusso e di congestione di TCP, per cui quando avviene una perdita il flusso si arresta ed è dunque tutt'altro che continuo. Nella realtà quindi, al crescere della probabilità di perdita sul pacchetto (che vale $1-(1-p)^n$) cresce la presenza di alcuni "tempi morti" nella connessione; questo effetto riduce le prestazioni, e dovrebbe essere signficativo per n e p abbastanza grandi (il parametro che conta è la probabilità di perdita sul pacchetto che, come appena ricordato, dipende da n e p).

Da queste considerazioni qualitative ci si può attendere che per p nulla o molto piccola e n non molto grande l'incremento di prestazioni rispetto all'assenza di frammentazione (caso $n = 1$) sia maggiore rispetto a quello indicato nel modello, mentre per p significativa e n sufficientemente grande il decadimento prestazionale dovrebbe essere maggiore di quello indicato.

6.6.2 Risultati sperimentali

Per verificare sperimentalmente la situazione descritta nel paragrafo precedente, si sono condotte alcune prove di laboratorio sulla rete mostrata in figura 6.1. Per i test si è utilizzato ancora una volta netperf con il test "TCP stream", ponendo il client su H1 e il server su H2 (per cui il flusso di dati va da H1 a H2); su G2 è stato attivato NIST Net in modo da simulare la presenza di una rete intermedia con una data probabilità di perdita (in sostanza NIST Net agisce scartando i pacchetti in transito secondo la probabilità fornitagli). La frammentazione è stata ottenuta modificando la MTU dell'interfaccia virtuale di IPsec su H1, scegliendo valori tali da ottenere il numero di frammenti desiderato e frammenti della dimensione massima. Nei test non si è utilizzato il selective acknowledgement (SACK) di TCP[9] (tale opzione è implementata in Linux ma non in tutti i sistemi) per non alterare il comportamento standard di TCP in presenza di perdite casuali come quelle qui introdotte.

La figura 6.5 mostra l'andamento del throughput di IPsec in funzione del numero di frammenti, per varie probabilità di perdita comprese tra 0 e 1%. Per $p = 0$ si nota che la curva cresce rapidamente passando da un valore $X = 839$ KB/s per $n = 1$ ad un valore $X = 940$ KB/s già per $n = 4$, assestandosi poi intorno a questo valore. Tra il caso $n = 1$ e il caso $n = 4$ si ha quindi un miglioramento di circa il 12%, che scende a circa il 10% per $n = 3$, mentre non sale significativamente per $n > 4$. Per $p = 0{,}25\%$ si vede che la curva cresce fino a $n = 3$ (a cui corrisponde un guadagno di circa il 10% sul rispettivo caso $n = 1$), poi rimane circa costante dopo di che decresce leggermente per $n > 6$ (per $n = 8$ si ha comunque ancora un guadagno di circa il 7%). La curva corrispondente a $p = 0{,}5\%$ ha anch'essa un andamento crescente fino a $n = 3$ (con un guadagno di poco più del 5%) dopo di che decresce lentamente. La curva relativa a $p = 0{,}75\%$ si mantiene sostanzialmente costante fino a $n = 3$

[9] L'opzione SACK permette a TCP di inviare riscontri selettivi oltre ai normali riscontri cumulativi. Con il TCP standard infatti il riscontro sul byte x comporta implicitamente il riscontro su tutti i byte precedenti x.

per poi decrescere, infine la curva relativa a $p = 1\%$ assume subito una pendenza negativa, presentando nel caso $n = 3$ una perdita di poco più del 4% rispetto al caso $n = 1$. Le curve decrescono sempre più rapidamente al crescere di p (come ci si attendeva), per cui si allontanano tra loro al crescere di n. Da questi risultati si può concludere che, su una rete affidabile (ovvero con probabilità di perdita media inferiore all'1%), sfruttando la frammentazione dei pacchetti IP si può avere un miglioramento degno di nota delle prestazioni di IPsec. Non appare però conveniente andare oltre i 3 (o al massimo 4) frammenti per ogni pacchetto IP, poiché il maggior guadagno a probabilità di perdita nulla è pressoché insignificante, mentre per probabilità di perdita superiori si può avere un peggioramento.

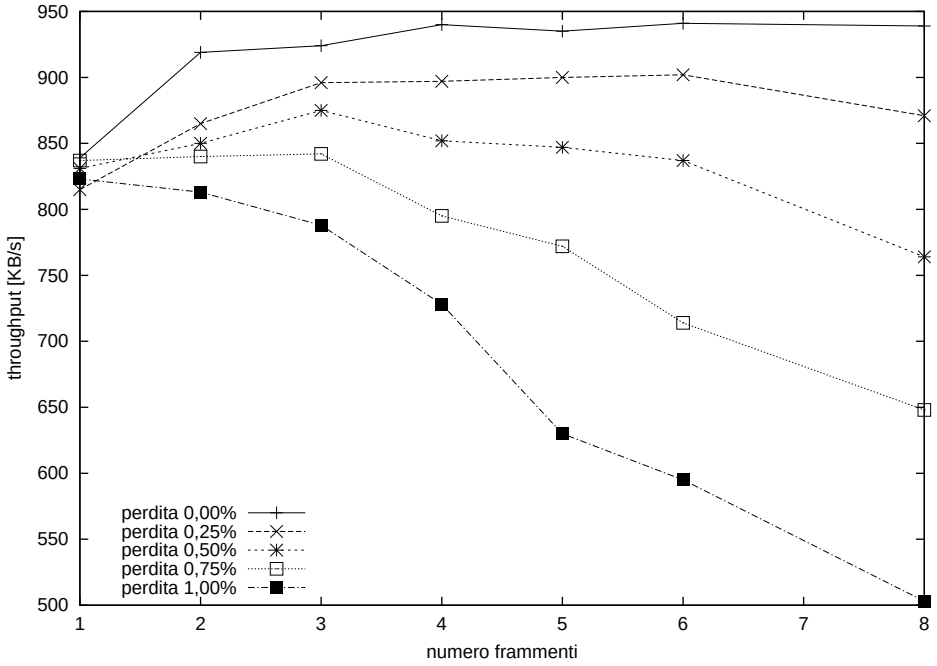

Figura 6.5: Comportamento di IPsec in presenza di frammentazione. Variazione del throughput in funzione del numero di frammenti.

Rispetto al modello teorico analizzato in precedenza si può notare che l'andamento delle curve è quello previsto. Si nota però che il miglioramento nel caso $p = 0$ riscontrato sperimentalmente (12% per $n = 4$) è molto maggiore rispetto a quello teorico (6% per $n \to \infty$). In figura 6.6 sono mostrati gli andamenti (per $p = 0$) dell'efficienza teorica e del throughput misurato, quest'ultimo riportato sulla scala dell'efficienza eguagliandone il valore per $n = 1$ alla corrispondente efficienza teorica, e utilizzando poi tale rapporto per tradurre gli altri punti. Come si vede, le due curve si discostano notevolmente: un effetto del genere era stato previsto (attribuendolo essenzialmente agli ack), ma lo scostamento è tanto grande da suggerire l'opportunità di un approfondimento sulla questione.

Essendo in assenza di perdite, il secondo degli "effetti collaterali" ignorati dal modello (ovvero i meccanismi di ritrasmissione di TCP) non sussiste, per cui ci si può concentrare sul primo, vale a dire la presenza degli ack. Gli ack, come detto, fluiscono in senso opposto ma contribuiscono ad occupare la banda disponibile[10]: poiché viene inviato un ack per ogni pacchetto (e *non* per ogni frammento) si può pensare di includerli nell'overhead di pacchetto, ovvero in L_{OP}. Nelle prove sperimentali, un ack è composto da 104 byte: 20 byte per l'header IP esterno, 8 byte per l'header ESP, 8 byte per l'IV,

[10] Il canale è infatti half-duplex.

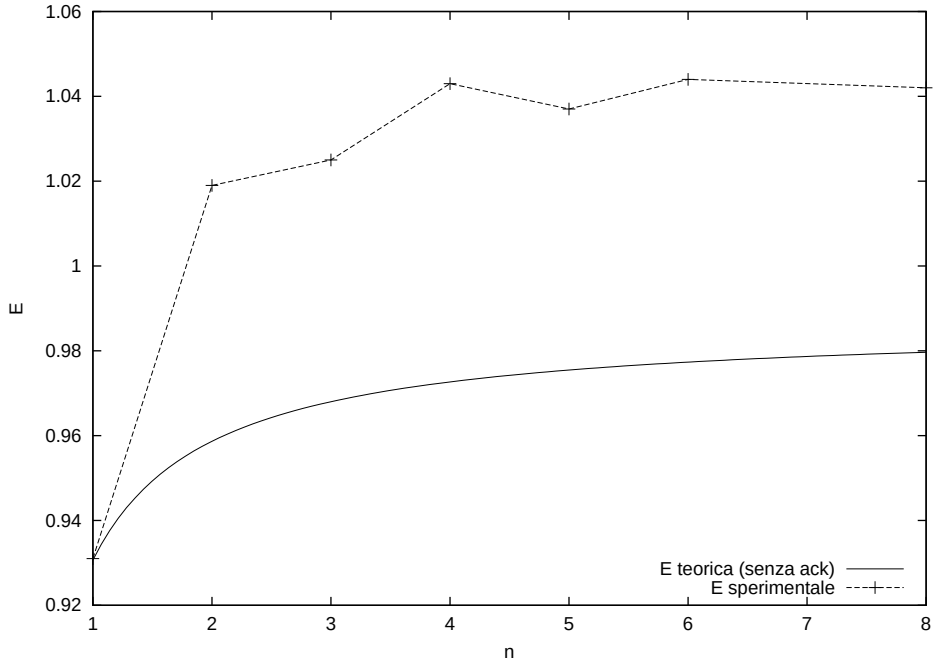

Figura 6.6: Confronto (nel caso $p = 0$) tra l'efficienza teorica e l'efficienza misurata sperimentalmente.

20 byte per l'header IP interno, 32 byte per l'header TCP, 2 byte di padding, 2 byte per il trailer ESP e infine 12 byte per i dati di autenticazione. Si può allora inserire nella formula dell'efficienza un valore $L_{OP} = 188$ byte, anziché 84 byte come considerato in precedenza: in questo caso il valore asintotico rimane pari a circa $0{,}987$, ma il valore per $n = 1$ è pari a circa $0{,}861$ (anziché $0{,}931$), il che comporta un guadagno asintotico di circa il 14,6%, decisamente in linea con il valore ottenuto sperimentalmente.

Si è allora di confrontato il throughput sperimentale con l'andamento dell'efficienza teorica, questa volta calcolata includendo gli ack in L_{OP}, anche qui parificando il throughput per $n = 1$ alla corrispondente efficienza teorica e traducendo poi in base a tale rapporto i valori di throughput misurati in valori sulla scala dell'efficienza. Il risultato è mostrato in figura 6.7: come si vede con questa ipotesi l'andamento osservato sperimentalmente è molto vicino a quello teorico.

Per eliminare completamente l'effetto degli ack, e valutare l'adeguatezza del modello, si sono infine condotti dei test su UDP anziché su TCP[11]. L'overhead in questo caso è più basso, infatti, poiché l'header UDP misura solamente 8 byte contro i 32 byte di TCP, in questo caso si ha $L_{OP} = 60$ byte. Questo significa che, sempre per $p = 0$, si ha $E(1) \simeq 0{,}947$, e che quindi il guadagno per $n \to \infty$ è di circa il 4,2% (il valore asintotico dell'efficienza è sempre il medesimo). Come si vede in figura 6.8, la coincidenza tra valori teorici e sperimentali è in questo caso pressoché completa.

6.7 Tempi di handshake

Dopo aver valutato le prestazioni di IPsec e TLS nella fase di trasferimento dei dati, si è passati a considerarne i tempi di handshake. Per quanto riguarda IKE le misurazioni sono state effettuate intercettando

[11] Per effettuare questi test si è abbassata a 10 Mbit/s la velocità di trasmissione sul collegamento tra H1 e G1 per evitare che, in mancanza dei meccanismi di controllo di flusso di TCP, netperf inviasse dati a 100 Mbit/s (con conseguenti enormi e incontrollate perdite su G1).

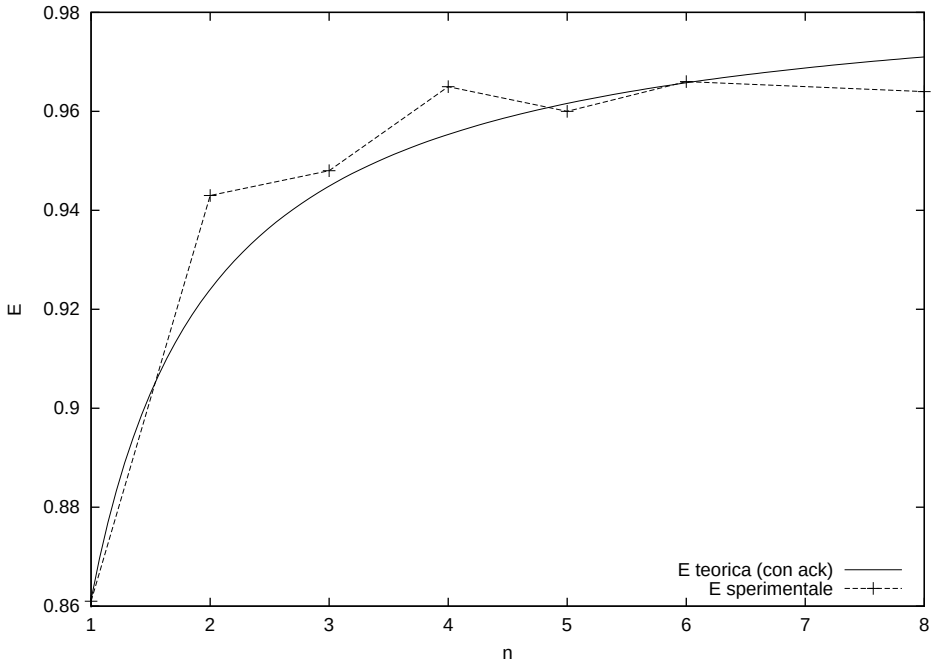

Figura 6.7: Confronto (nel caso $p = 0$) tra l'efficienza teorica includendo gli ack nell'overhead di pacchetto e l'efficienza misurata sperimentalmente.

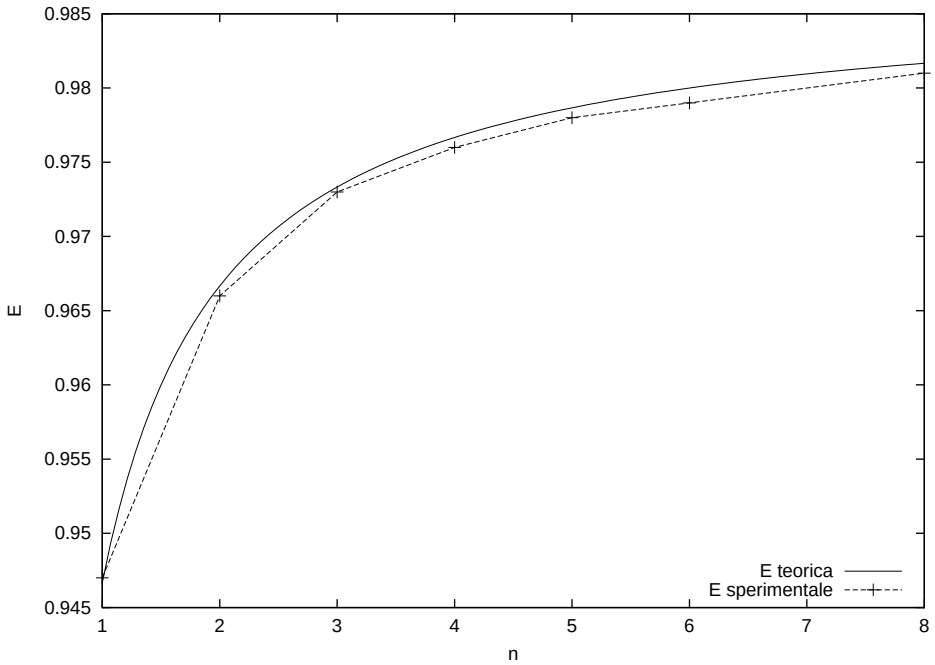

Figura 6.8: Confronto (nel caso $p = 0$) tra l'efficienza teorica e l'efficienza misurata sperimentalmente per UDP.

i pacchetti con uno sniffer[12] posto su una quinta macchina (che sarà indicata con S) collegata all'hub di figura 6.1, mentre nel caso di TLS si sono utilizzati dei timestamp inseriti nel codice di netperf, prima e dopo la chiamata alla funzione che effettua l'handshake.

È importante notare che nei tempi che sono stati misurati la componente largamente dominante è data dai tempi di elaborazione e non dalla latenza di rete, che nella rete di laboratorio è tanto piccola da essere trascurabile. Si sono infatti misurati i *round trip time* (RTT) tra le varie macchine tramite flood ping, ottenendo i seguenti risultati:

- da H1 a H2: 480 μs;
- da H1 a S: 340 μs;
- da H1 a G1: 100 μs;
- da H1 a G2: 380 μs.

Come si vedrà, il tempo di handhsake misurato per IKE è di tre ordini di grandezza superiore al RTT tra H1 e H2, che si può dunque considerare trascurabile nell'analisi dei risultati sperimentali. Per lo stesso motivo, a parte per alcuni intertempi dell'ordine del millisecondo, è trascurabile il fatto che i timestamp forniti dallo sniffer siano presi su S e non su H1 o H2.

6.7.1 IKE

Si è misurato l'handshake IKE secondo la modalità delle firme digitali, mostrato in figura 2.5 per la fase 1 (main mode) e in figura 2.7 per la fase 2 (quick mode)[13]. La sequenza dei pacchetti, con i timestamp (presi su S) e gli intertempi tra pacchetti successivi, è mostrata in tabella 6.8.

I primi due pacchetti del main mode non necessitano di elaborazioni significative (il primo può anche essere un pacchetto precalcolato), e infatti i relativi tempi sono molto piccoli e comparabili con il RTT[14]. Il terzo e il quarto contengono i dati per effettuare lo scambio Diffie–Hellman, il cui valore segreto deve essere calcolato dall'iniziatore tra la ricezione del quarto pacchetto e l'invio del quinto (mentre il ricevitore può farlo tra l'invio del quarto e la ricezione del quinto, ed ha dunque più tempo). Questo fatto, unito al fatto che il quinto pacchetto deve essere firmato (altra operazione di crittografia asimmetrica), determina per esso un tempo di elaborazione maggiore (circa 100 ms); anche il sesto pacchetto viene firmato, e dunque anch'esso comporta un tempo significativo. Per quanto riguarda il quick mode, i primi due pacchetti presentano tempi poco maggiori rispetto al terzo e al quarto del main mode: presentano in effetti funzionalità simili (scambio dei valori per il Diffie–Hellman e dei nonce) anche se hanno alcuni payload in più e soprattutto sono cifrati, la cifratura è però effettuata con un algoritmo *simmetrico*, dunque molto più leggero dal punto di vista computazionale. Si nota infine un tempo significativo sull'ultimo messaggio, probabilmente dovuto al calcolo del valore Diffie–Hellman.

Come detto, i tempi misurati sono sostanzialmente i tempi di elaborazione, dato che la durata totale dell'handshake è risultata di circa 470 ms a fronte di un RTT tra H1 e H2 di circa 480 μs (ovvero inferiore di tre ordini di grandezza). È però importante notare che, al di fuori di una rete locale, il RTT può essere molto maggiore e diventare il fattore dominante, considerato che al tempo di elaborazione

[12] Si è utilizzato Ethereal. I timestamp ottenuti hanno una precisione del microsecondo, ma nei dati che verranno analizzati ci si atterrà al decimo di millisecondo.

[13] Si è utilizzata la perfect forward secrecy, per cui sono presenti i payload KE.

[14] Dato che la misurazione inizia da quando il primo pacchetto è intercettato dallo sniffer, in realtà il suo tempo di elaborazione non viene misurato. Questo è comunque molto piccolo, infatti il pacchetto ha una struttura analoga al secondo (che ha un intertempo di 0,9 ms sul primo) e in più, a differenza di questo, può essere precalcolato.

misurato bisogna aggiungere $\frac{9}{2}RTT$ (essendo l'handshake composto in totale da 9 pacchetti). Con un RTT di 100 ms, ad esempio, si può stimare un handshake di circa $450 + 470 = 920$ ms (per cui la laenza di rete ha più o meno lo stesso peso dei tempi di elaborazione), mentre con un RTT di 200 ms si avrebbe una durata totale di circa $900 + 470 = 1370$ ms (per cui la latenza di rete pesa per due terzi del totale, e i tempi di elaborazione per un terzo).

Timestamp [ms]	Pacchetto	Direzione	Intertempo [ms]
0	IKE main mode 1	H1 → H2	-
0,9	IKE main mode 2	H2 → H1	0,9
35,6	IKE main mode 3	H1 → H2	34,7
81,5	IKE main mode 4	H2 → H1	45,9
181,6	IKE main mode 5	H1 → H2	100,1
271,5	IKE main mode 6	H2 → H1	89,9
312,4	IKE quick mode 1	H1 → H2	40,9
363,4	IKE quick mode 2	H2 → H1	51,0
469,9	IKE quick mode 3	H1 → H2	106,5

Tabella 6.8: Timestamp dell'handshake di IKE.

6.7.2 IKE "opportunistic"

Si è misurato un handshake IKE con la opportunistic encryption: i dati relativi sono presentati in tabella 6.9.

Questo è sostanzialmente un normale handshake IKE, con in più alcune interrogazioni al DNS. Gli intertempi tra i pacchetti IKE sono gli stessi del caso precedente, come era lecito attendersi, e le query al DNS non comportano tempi di elaborazione significativi. In un caso reale però le tre[15] interrogazioni al DNS comprese nell'handshake hanno sicuramente un peso non trascurabile: sia per i tempi di risposta del DNS e per i tempi di trasmissione, sia perché l'implementazione attuale *non* utilizza DNSSEC, e dunque non verifica l'autenticità dei dati (che è un'operazione onerosa in quanto comporta l'utilizzo di crittografia asimmetrica).

6.7.3 TLS

L'handshake di TLS (figura 3.2) risulta opaco all'analizzatore di pacchetti utilizzato (Ethereal). È stato allora misurato inserendo dei timestamp nel codice (già modificato) di netperf, prima e dopo le chiamate di funzione che effettuano l'handshake. Si è effettuato un handshake con l'autenticazione del solo server[16] e si è trovato che tale handshake, nel caso di laboratorio, richiede circa 52 ms, cioè un tempo inferiore di un ordine di grandezza rispetto all'handshake IKE. Questo dipende dal fatto che l'handshake di IKE è in realtà un doppio handshake con una doppia negoziazione: nella prima fase si negozia la ISAKMP SA, nella seconda la IPsec SA. A ciò bisogna aggiungere che l'handshake di TLS

[15] Potrebbero anche essere due, perché si potrebbe includere la chiave nel record TXT e dunque evitare la seconda query. Vale la pena di notare che qui si è considerato il caso end–to–end, perché nel caso gateway–to–gateway il ricevitore (security gateway della destinazione) dovrebbe fare anche una query TXT per accertarsi che l'iniziatore sia autorizzato a rappresentare la sorgente.

[16] Questo è il caso pratico più comune attualmente, ma è importante notare che con IKE si sono invece autenticati entrambi gli interlocutori.

Timestamp [ms]	Pacchetto	Direzione	Intertempo [ms]
0	DNS query TXT	H1 → S	-
0,5	DNS reply TXT	S → H1	0,5
1,3	DNS query KEY	H1 → S	0,8
1,8	DNS reply KEY	S → H1	0,5
7,2	IKE main mode 1	H1 → H2	5,4
8,3	IKE main mode 2	H2 → H1	1,1
42,5	IKE main mode 3	H1 → H2	34,2
88,4	IKE main mode 4	H2 → H1	45,9
188,2	IKE main mode 5	H1 → H2	99,8
188,9	DNS query KEY	H2 → S	0,7
189,4	DNS reply KEY	S → H2	0,5
279,8	IKE main mode 6	H2 → H1	90,4
320,8	IKE quick mode 1	H1 → H2	41,0
371,5	IKE quick mode 2	H2 → H1	50,7
478,0	IKE quick mode 3	H1 → H2	106,5

Tabella 6.9: Timestamp dell'handshake di IKE "opportunistic".

è completamente in chiaro, mentre in quello di IKE cinque pacchetti su nove sono cifrati, e che avendo autenticato solo una delle due parti l'impatto delle operazioni di crittografia asimmetrica è minore.

L'handshake TLS risulta decisamente più breve anche in un caso reale, in cui conta la latenza di rete: questa infatti è stimabile in $2RTT$, contro i $\frac{9}{2}RTT$ di IKE.

7 Conclusioni

IPsec, per la sua versatilità e per il fatto di essere parte integrante di IPv6 (per cui in futuro sarà integrato in tutti i sistemi operativi), si può senza dubbio considerare la soluzione più generale per la protezione delle informazioni in transito su Internet. Come si è visto nel capitolo 4, IPsec può proteggere sostanzialmente qualunque tipo di traffico, è molto flessibile, e per certi aspetti si può considerare più sicuro di TLS.

Ovviamente IPsec non è comunque la soluzione a tutti i problemi di sicurezza esistenti, e potrebbe anzi aprirne di nuovi, come si è visto, ad esempio per quanto riguarda gli attacchi di denial of service.

In particolare IPsec non elimina la necessità di funzionalità di sicurezza a livello applicazione, sia perché protegge i dati solo durante il transito in rete (e non dopo che sono giunti a destinazione) sia perché non fornisce alcune funzionalità come il non ripudio. Quest'ultima potrebbe essere una caratteristica desiderabile ad esempio in una transazione di commercio elettronico; attualmente in questo ambito si usa in genere SSL/TLS, ma come si è detto nemmeno questo fornisce il non ripudio. In presenza di IPsec, le applicazioni possono comunque sfruttarne i servizi senza dover ogni volta "reinventare la ruota" implementandoli in proprio, e aggiungendo al loro interno solo le funzionalità che IPsec non può offrire.

SSL/TLS è nato prima di IPsec, ed è andato ad occupare un settore allora scoperto; per questo motivo e per il fatto che non presenta particolari difficoltà di "infrastruttura" (che in IPsec sono invece indubbiamente maggiori) ha avuto una buona diffusione. Per quanto sia in sé un protocollo di uso abbastanza generale, il suo campo di applicabilità è più ristretto rispetto ad IPsec, ed è lecito domandarsi se in un mondo in cui tutte le macchine fossero in grado di instaurare automaticamente connessioni IPsec tra di loro ci sarebbe bisogno di qualcosa come TLS oppure no. Questa domanda fa in un certo senso da filo conduttore all'analisi svolta.

Dal punto di vista delle prestazioni, nel capitolo 6 si è visto che quelle di IPsec sono comparabili con quelle di TLS o lievemente inferiori, e che comunque in alcuni casi sono migliorabili. Si è infatti visto che, su una rete molto affidabile, è possibile ottenere un buon incremento delle prestazioni di IPsec agendo sulla frammentazione dei pacchetti IP.

IPsec, come ricordato più volte, presenta una grande complessità, il che comporta varie difficoltà nella sua diffusione e nel suo utilizzo. La risposta alla domanda posta poco fa è probabilmente che se qualunque macchina connessa ad Internet fosse in grado di comunicare tramite IPsec con qualunque altra non ci sarebbe più bisogno di TLS. Il fatto è che uno scenario del genere è al momento ancora piuttosto lontano, soprattutto per le problematiche connesse all'instaurazione di una connessione che non sia preconfigurata.

Alcune proposte per risolvere questo problema sono state analizzate nel capitolo 5: hanno caratteristice diverse, soprattutto per il diverso scenario in cui sono applicabili, ma non comportano grandi rivoluzioni. Il caso "centralizzato", che permette di appoggiarsi a Kerberos, è ancora allo studio ma non presenta grosse difficoltà; la opportunistic encryption non è ancora completamente definita ma è già allo stato di prototipo implementato, anche se per un suo completo utilizzo è necessario tenere presente il ruolo di DNSSEC. Quest'ultimo è infatti un protocollo ormai standardizzato ed implementato, e a detta

di molti necessario perché l'attuale totale insicurezza del DNS non è più sostenibile, ma al momento incontra anch'esso grossi problemi di diffusione e non è chiaro se le zone radice (ovvero quelle che fanno capo ai root server) saranno mai firmate. Su DNSSEC si appoggia anche il protocollo HIP, che è stato descritto nel capitolo 5; HIP presenta molti vantaggi ma costituisce un grande cambiamento, anche a livello concettuale, e ha un grosso impatto sull'intero stack dei protocolli di rete per come sono utilizzati oggi, e dunque appare per ora una soluzione abbastanza futuribile.

In sostanza IPsec, insieme con protocolli quali IPv6 e DNSSEC, e con proposte quali HIP, fa parte del grande disegno di rinnovamento dell'infrastruttura di Internet, che comporta l'aggiornamento o la sostituzione dei protocolli (e in alcuni casi anche dei concetti) che ancora oggi ne regolano il funzionamento, ma che furono definiti venti o più anni or sono in un contesto radicalmente diverso da quello attuale. La transizione non è semplice, come dimostra attualmente la situazione di IPv6, perché intacca le fondamenta stesse del funzionamento della rete, ma è necessaria perché Internet possa rispondere adeguatamente alle nuove esigenze manifestatesi nel corso degli ultimi anni, e sostenere le nuove sfide che si presenteranno in quelli a venire.

Appendice

A Note di crittografia[1]

A.1 Crittografia simmetrica e asimmetrica

La *crittografia* è la scienza che si occupa di proteggere delle informazioni rendendole incomprensibili a chi le dovesse intercettare, in modo che possano essere lette e capite solo dal destinatario[2].

Il messaggio da proteggere viene detto *testo in chiaro*, mentre quello "trasformato" in modo da essere incomprensibile viene detto *testo cifrato*; la trasformazione da testo in chiaro a testo cifrato si dice *cifratura*, mentre la trasformazione inversa si dice *decifratura*. La trasformazione crittografica è detta *algoritmo di cifratura*, e specifica la procedura che trasforma il testo in chiaro in quello cifrato. Questa trasformazione è parametrica, e il parametro è detto *chiave*: questo significa che la trasformazione in sé è soltanto un procedimento, che però per essere attuato ha bisogno di un'informazione ulteriore, da cui dipende il risultato. Per decifrare il messaggio non basta conoscere l'algoritmo di cifratura utilizzato, ma è necessario conoscere anche la chiave. In genere nello studio degli algoritmi crittografici e della loro sicurezza si ipotizza che l'algoritmo sia noto a tutti e che ciò che non è noto sia esclusivamente la chiave: questo perché oggi si considera inaffidabile un sistema crittografico la cui sicurezza si basi sulla segretezza dell'algoritmo.

La crittografia tradizionale si basa sul meccanismo descritto in figura A.1, per cui cifratura e decifratura sono effettuate utilizzando la stessa chiave: per questo si parla di *crittografia simmetrica*, o anche, dato che tale chiave deve essere nota solo ai due interlocutori, di *crittografia a chiave segreta*. Un noto algoritmo di questo tipo è il *DES* (Data Encryption Standard).

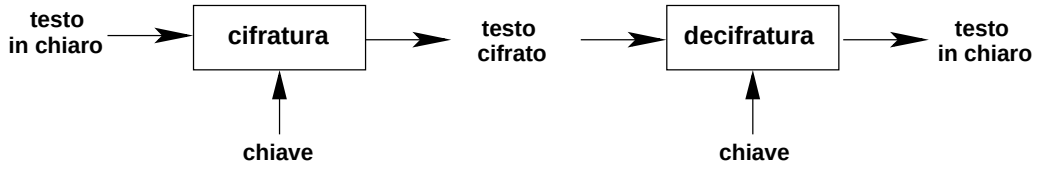

Figura A.1: Crittografia simmetrica.

Il grosso problema di questo approccio è però la *distribuzione delle chiavi*: se due interlocutori vogliono usare un algoritmo di questo tipo per comunicare in modo sicuro devono prima accordarsi in qualche modo sulla chiave, per esempio vedendosi di persona. Dato che il canale che usano per la trasmissione dei messaggi non è sicuro (altrimenti non avrebbero bisogno di cifrarli), non possono infatti utilizzarlo per trasmettere la chiave.

Il problema è stato risolto in tempi relativamente recenti (anni Settanta) con l'invenzione della *crittografia a chiave pubblica*. Con algoritmi di questo tipo ognuno ha due chiavi: una pubblica da

[1] Per una trattazione completa degli argomenti relativi alla crittografia si veda [34].

[2] Non bisogna confondere i sistemi crittografici, in cui il messaggio è identificabile ma incomprensibile, con quelli atti a *nascondere* l'informazione in modo che non ci si accorga della sua presenza (ad esempio scrivere un messaggio con inchiostro simpatico). La scienza che si occupa di queste ultime tecniche è detta *steganografia*.

distribuire a tutti quelli con cui vuole comunicare, e una privata da tenere segreta. Ciò che viene cifrato con la chiave pubblica (operazione che può essere fatta da chiunque) può essere decifrato solo con la chiave privata corrispondente (operazione che può essere fatta solo dal proprietario della chiave): in questo modo non c'è più il problema di comunicare segretamente la chiave, perché questa è nota a tutti; per comunicare in modo sicuro con una persona basta cifrare il messaggio con la sua chiave pubblica, come illustrato in figura A.2. Gli algoritmi di questo tipo sono detti *a chiave asimmetrica*, e il più noto tra essi è probabilmente RSA (si veda l'appendice A.4).

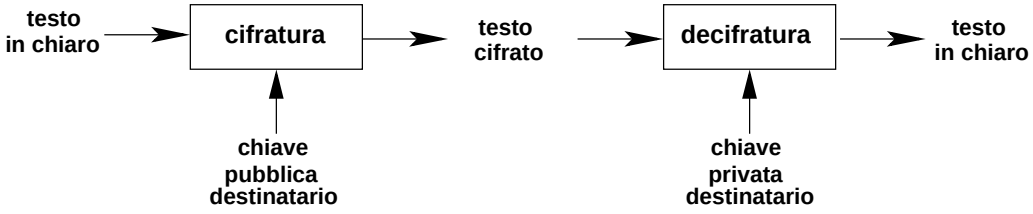

Figura A.2: Crittografia asimmetrica.

Una delle grosse innovazioni permesse dalla crittografia asimmetrica è la *firma digitale*: il mittente di un messaggio può infatti firmarlo grazie alla sua chiave privata (che solo lui possiede), ma tutti sono in grado di verificare l'autenticità della firma grazie alla chiave pubblica (che è globalmente nota). La firma può poi essere abbinata alla normale cifratura, ottenendo messaggi firmati e cifrati, nel modo seguente (figura A.3). Si ipotizzi che Bob voglia mandare un messaggio ad Alice: per prima cosa lo cifrerà con la propria chiave privata (firma) e poi con quella pubblica di Alice (cifratura), infine spedirà il messaggio. Quando Alice lo riceverà, prima lo decifrerà con la propria chiave privata (operazione che solo lei può fare, per cui è garantita la confidenzialità) e poi con quella pubblica di Bob: se l'operazione va a buon fine Alice ha la certezza che il mittente è davvero Bob, perché solo lui può aver cifrato il messaggio con la propria chiave privata. In realtà il modo in cui si realizza la firma digitale non è proprio questo, ma è leggermente più complicato. Ciò che il mittente cifra con la propria chiave privata per garantire l'autenticità non è l'intero messaggio, ma una sua "impronta" (detta *digest*) ottenuta mediante una particolare funzione (*funzione di hash*, si veda l'appendice A.5); il digest cifrato viene poi allegato al messaggio e ne costituisce la firma. Il destinatario calcola il digest del messaggio ricevuto (la funzione di hash è pubblica) e lo confronta con quello che ottiene decifrando con la chiave pubblica del mittente la firma allegata: se coincidono la firma è autentica.

Si potrebbe pensare che dopo l'invenzione della crittografia a chiave pubblica quella a chiave segreta abbia perso interesse, ma le cose non stanno così. Gli algoritmi asimmetrici sono infatti computazionalmente molto più onerosi di quelli simmetrici, per cui il loro uso risulta molto più pesante. Una soluzione comunemente adottata è l'uso della crittografia asimmetrica per concordare una chiave segreta da utilizzare poi per scambiarsi i messaggi tramite un algoritmo a chiave simmetrica: in questo modo il "pesante" algoritmo a chiave pubblica viene usato solo per trasmettere una piccola quantità di dati (la chiave segreta), mentre per il resto si usa il più leggero algoritmo a chiave segreta.

A.2 Cifrari a blocco e loro modalità operative

Un *cifrario a blocco* (block cipher) opera su blocchi di dati in ingresso producendo blocchi di dati in uscita (esitono anche i *cifrari a flusso* (stream ciphers), che operano invece su un flusso di dati in maniera continua). Il più noto cifrario a blocco è probabilmente il *DES* (con la variante del triplo DES), altri sono ad esempio IDEA, Blowfish, CAST-128 e RC5.

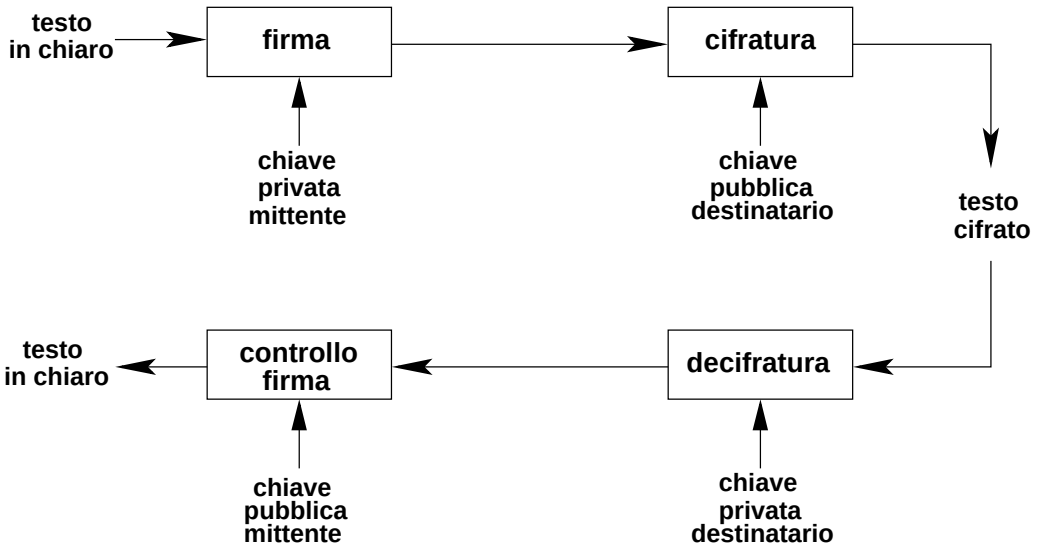

Figura A.3: Cifratura e firma digitale.

Il modo più semplice di utilizzare un cifrario a blocco è quello di cifrare singolarmente (e indipendentemente) ogni blocco in ingresso: in questo modo (noto come modalità *Electronic CodeBook* — ECB) blocchi uguali in ingresso (cioè nel testo in chiaro) producono blocchi uguali in uscita (cioè nel testo cifrato). Questa proprietà, in presenza di ridondanze note, può essere d'aiuto per la crittoanalisi del testo cifrato, inoltre facilita operazioni quali la sostituzione di uno o più blocchi. Si utilizzano allora delle modalità operative diverse. Una di esse è detta *Cipher Block Chaining* (CBC), e funziona nel modo seguente. Siano E_K la funzione di cifratura del singolo blocco (parametrizzata dalla chiave K), P_i l'i-esimo blocco di testo in chiaro e C_i l'i-esimo blocco di testo cifrato; allora C_i viene calcolato come

$$C_i = E_K \left[C_{i-1} \oplus P_i \right]$$

dove il simbolo $\oplus$ indica la disgiunzione esclusiva (XOR) bit a bit. In questo modo si crea una catena, in quanto la cifratura di un blocco è effettuata tenendo in considerazione anche il blocco precedente. Se la funzione di decifratura sul singolo blocco è D_K, la decifraura è allora effettuata nel modo seguente:

$$P_i = C_{i-1} \oplus D_K \left[C_i \right]$$

Per ottenere il primo blocco si deve utilizzare il cosiddetto *vettore di inizializzazione* (IV), che deve essere segreto e noto ad entrambe le parti, per cui:

$$C_1 = E_K \left[IV \oplus P_1 \right]$$

$$P_1 = IV \oplus D_K \left[C_1 \right]$$

Esistono altre modalità operative per i cifrari a blocco, quali ad esempio la modalità *Cipher Feed-Back* (CFB).

A.3 Diffie–Hellman

Il protocollo di Diffie–Hellman permette a due interlocutori che non hanno mai avuto nessun precedente contatto di accordarsi su una chiave segreta utilizzando un canale pubblico.

Alcune definizioni necessarie per poter illustrare il protocollo:

- Sia $\mathbb{Z}$ l'insieme dei numeri interi.
- Un numero $p \in \mathbb{Z}$ è detto primo se è divisibile solo per 1 e per sé stesso.
- Un numero $p \in \mathbb{Z}$ si dice primo rispetto a $q \in \mathbb{Z}$ se il massimo comune divisore tra p e q è 1. Si dice anche che p e q sono primi tra loro.
- Sia $\mathbb{Z}_p$ l'insieme dei numeri interi tra 0 e $p-1$ compresi, ovvero l'insieme $\{0, ..., p-1\}$.
- Sia $\mathbb{Z}_p^*$ il sottoinsieme di $\mathbb{Z}_p$ comprendente solo i numeri primi rispetto a p. Se p è un numero primo allora $\mathbb{Z}_p^* = \{1, ..., p-1\}$.
- Sia a un numero intero e m un intero positivo, allora indichiamo con $a \bmod m$ il resto della divisione di a per m, ovvero $a \bmod m = a - \left\lfloor \frac{a}{m} \right\rfloor m$ (dove il simbolo $\lfloor x \rfloor$ indica l'intero inferiore di x).
- Siano a e b due numeri interi e m un intero positivo, allora diciamo che $a \equiv b \pmod{m}$ se $a \bmod m = b \bmod m$, ovvero se m divide $b - a$.
- Un numero $\alpha \in \mathbb{Z}_p^*$ è detto elemento generatore di $\mathbb{Z}_p^*$ se il più piccolo numero q tale che $\alpha^q \equiv 1 \pmod{p}$ è $p-1$.

Detti Bob e Alice i due interlocutori, il protocollo procede come segue:

1. Bob e Alice si accordano pubblicamente su un numero primo p e su un numero α che sia elemento generatore di $\mathbb{Z}_p^*$.
2. Alice sceglie un numero a_1 tale che $1 \leq a_1 \leq p-2$, calcola $b_1 = \alpha^{a_1} \bmod p$ e lo manda a Bob.
3. Bob sceglie un numero a_2 tale che $1 \leq a_2 \leq p-2$, calcola $b_2 = \alpha^{a_2} \bmod p$ e lo manda ad Alice.
4. La chiave segreta è $K = \alpha^{a_1 a_2} \bmod p$: Alice può calcolarla come $K = b_2^{a_1} \bmod p$, mentre Bob può calcolarla come $K = b_1^{a_2} \bmod p$.

La sicurezza del protocollo di Diffie-Hellman, ovvero il fatto che un intruso non possa calcolare la chiave K avendo a disposizione solo i dati pubblici p, α, b_1, b_2, si basa sulla congettura dell'intrattabilità del problema dei logaritmi discreti in $\mathbb{Z}_p$, per cui conoscendo solo i dati pubblici è molto difficile ricavare a_1 e a_2. Così com'è, ovvero senza alcuna forma di autenticazione (come ad esempio dei certificati), è però soggetto ad un attacco di tipo man–in–the–middle. Se infatti un intruso, che chiamiamo Oscar, si pone nel mezzo della comunicazione può intercettare b_1 e b_2 e mandare, rispettivamente, ad Alice un valore $b_3 = \alpha^{a_3} \bmod p$ e a Bob un valore $b_4 = \alpha^{a_4} \bmod p$ (calcolati a partire da dei valori a_3 e a_4 scelti da lui). In questo modo Oscar riuscirà a stabilire una chiave $K_1 = \alpha^{a_1 a_3} \bmod p$ con Alice e una chiave $K_2 = \alpha^{a_2 a_4} \bmod p$ con Bob, mentre Alice e Bob penseranno di essersi accordati su una chiave tra di loro: così Oscar avrà la successiva comunicazione tra Bob e Alice in chiaro, e potrà leggere i messaggi, modificarli, inserirne altri.

A.4 RSA

RSA (dai nomi dei tre ricercatori che lo proposero, Rivest, Shamir e Adleman) è probabilmente il più noto algoritmo crittografico asimmetrico, e può essere utilizzato sia per la cifratura che per la firma. I passi preliminari sono i seguenti (per alcune definizioni si veda il paragrafo precedente):

1. Bob sceglie due numeri primi molto grandi, p e q.
2. Bob calcola $n = pq$ e $\varphi(n) = (p-1)(q-1)$ (la funzione $\varphi(n)$, detta *funzione fi di Eulero*, indica il numero di elementi di $\mathbb{Z}_n$ primi rispetto a n).
3. Bob sceglie un numero b, tale che $0 \leq b \leq \varphi(n)$, primo rispetto a $\varphi(n)$.
4. Bob calcola $a = b^{-1} \bmod \varphi(n)$ (ovvero trova quel numero a tale che $ab \equiv 1 \pmod{\varphi(n)}$, per come è stato scelto b tale numero esiste ed è unico, e si può calcolare con un algoritmo detto di Euclide).
5. Bob rende disponibili i valori n e b, che costituiscono la sua chiave pubblica, e tiene segreti i valori p, q e a, che costituiscono la sua chiave privata.

A questo punto l'uso di RSA per la cifratura funziona come segue:

1. Sia x il testo in chiaro del messaggio che Alice vuole mandare a Bob cifrandolo. Alice calcola $y = x^b \bmod n$ e lo manda a Bob.
2. Bob riceve y da Alice, e può decifrare il messaggio calcolando $x = y^a \bmod n$.

RSA può essere utilizzato per la firma nel modo seguente:

1. Sia x il testo in chiaro del messaggio che Bob vuole mandare ad Alice firmandolo. Bob calcola la firma come $y = x^a \bmod n$ e la manda ad Alice.
2. Alice riceve x e y da Bob, e verifica che sia $x = y^b \bmod n$. Se è così la firma è autentica, altrimenti è falsa.

La sicurezza di RSA si basa sulla congettura dell'intrattabilità del problema della scomposizione in fattori primi, per cui un intruso, noti n e b, non può calcolare a perché non conosce p e q (e quindi $\varphi(n)$).

A.5 Funzioni di hash e MAC

Una *funzione di hash* è una funzione che, dato un qualunque messaggio di lunghezza arbitraria, ne produce un'*impronta* (detta *digest*) di lunghezza prefissata (di solito dell'ordine di 100-200 bit). L'utilità di una funziona di hash sta nel poter utilizzare l'impronta come rappresentazione compatta del messaggio stesso, tipicamente firmando l'impronta anziché l'intero messaggio; perchè questo possa avvenire è desiderabile che la funzione presenti due proprietà particolari, sia cioè *senza collisioni* e *unidirezionale*.

- Una funzione di hash $h(x)$ è detta *debolmente senza collisioni* se, dato un messaggio x, è computazionalmente impraticabile trovare un messaggio $x' \neq x$ tale che $h(x') = h(x)$.

- Una funzione di hash $h(x)$ è detta *fortemente senza collisioni* se è computazionalmente impraticabile trovare due messaggi x e x' tali che $x' \neq x$ e $h(x') = h(x)$.
- Una funzione di hash $h(x)$ è detta *unidirezionale* se, data un'impronta z, è computazionalmente impraticabile trovare un messaggio x tale che $h(x) = z$.

Le funzioni di hash possono essere utilizzate insieme con delle chiavi segrete per ottenere autenticazione e integrità dei messaggi, realizzando così dei *MAC* (Message Authentication Code). Più precisamente, possiamo definire algoritmo di MAC una funzione $h_k(x)$, parametrizzata da una chiave k, con le seguenti proprietà:

- Dati una chiave k e un input x, è facile calcolare $h_k(x)$. Tale valore è detto brevemente MAC.
- Dato un input x di lunghezza qualsiasi la funzione genera un output $h_k(x)$ di lunghezza prefissata.
- Noto un certo numero di coppie $(x_i, h_k(x_i))$ è computazionalmente impraticabile calcolare qualunque altra coppia $(x, h_k(x))$ per un qualsiasi nuovo input $x \neq x_i$ senza conoscere k.

B Software utilizzato

Linux

Sistema operativo.
Si è utilizzata la distribuzione Red Hat, nelle versioni 7.0 e 7.1. I kernel utilizzati sono il 2.2.16, 2.4.2 e 2.4.6.
`http://www.linux.org`
`http://www.redhat.com`
`http://www.kernel.org`

FreeS/WAN

Implementazione IPsec.
Si è sono utilizzate le versioni 1.9 e 1.91.
`http://www.freeswan.org`

OpenSSL

Implementazione TLS/SSL.
Si è utilizzata la versione 0.9.6a.
`http://www.openssl.org`

Netperf

Programma per misurare le prestazioni di rete.
Si è utilizzata la versione 2.2alpha.
`http://www.netperf.org`

NIST Net

Emulatore di rete.
Si è utilizzata la versione 2.0.10.
`http://www.antd.nist.gov/nistnet`

Ethereal

Sniffer e analizzatore di protocolli di rete.
Si è utilizzata la versione 0.8.15.
`http://www.ethereal.com`

TCPdump

Sniffer.
Si è utilizzata la versione 3.4
`http://www.tcpdump.org`

GNUplot

Programma per tracciare grafici.
Si è utilizzata la versione 3.7.1
`http://www.gnuplot.org`

LaTeX

Sistema di formattazione documenti.
Si è utilizzato LaTeX 2_ε, distribuzione teTeX versione 1.0.7.
`http://www.tug.org/teTeX`

Elenco degli acronimi

AH	Authentication Header
CBC	Cipher Block Chaining
CPU	Central Processing Unit
DES	Data Encryption Standard
DH	Diffie–Hellman
DNS	Domain Name System
DNSSEC	Domain Name System Security
DoS	Denial of Service
ESP	Encapsulating Security Payload
FTP	File Transfer Protocol
GPL	General Public License
HI	Host Identity
HIP	Host Identity Payload
HIT	Host Identity Tag
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
ICV	Integrity Check Value
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IPComp	IP Payload Compression Protocol
IPsec	Internet Protocol Security

IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IRC	Internet Relay Chat
ISAKMP	Internet Security Association and Key Management Protocol
IV	Initialization Vector
LDAP	Lightweight Directory Access Protocol
LSI	Local Scope Identity
KINK	Kerberized Internet Negotiation of Keys
MAC	Message Authentication Code
MITM	Man In The Middle
MTU	Maximum Transfer Unit
NAPT	Network Address Port Translation
NAT	Network Address Translation
NNTP	Network News Transfer Protocol
OSI	Open System Interconnection
PFS	Perfect Forward Secrecy
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
POP	Post Office Protocol
RFC	Request For Comment
RSA	Rivest–Shamir–Adleman
RTT	Round Trip Time
SA	Security Association
SAD	Security Association Database
SHA	Secure Hash Algorithm
SET	Secure Electronic Transaction
SMTP	Simple Mail Transfer Protocol
SPD	Security Policy Database

SPI Security Parameters Index

SSL Secure Socket Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

VPN Virtual Private Network

Elenco delle figure

1.1 Diversi posizionamenti delle funzioni di sicurezza nello stack TCP/IP. 6

2.1 Posizionamento di AH all'interno del pacchetto IP. 8
2.2 Formato dell'header AH. 9
2.3 Posizionamento di ESP all'interno del pacchetto IP. 10
2.4 Formato del pacchetto ESP. 11
2.5 IKE main mode. 12
2.6 IKE aggressive mode. 12
2.7 IKE quick mode. 13
2.8 Elaborazione dei pacchetti IPsec. 14
2.9 Rete privata virtuale. 15
2.10 Road warrior. 16

3.1 Sequenza delle trasformazioni di TLS. 20
3.2 Handshake completo e abbreviato di TLS. 21

5.1 Posizione di HIP nello stack TCP/IP. 35
5.2 Formato del payload HIP. 36
5.3 Formato dello HIP key payload. 37

6.1 Rete di laboratorio utilizzata per i test sulla frammentazione. 39
6.2 Modello a code per i test con UDP. 42
6.3 Rappresentazione tridimensionale della variazione dell'efficienza teorica E in presenza di frammentazione, in funzione della probabilità di errore sul frammento p e del numero di frammenti n. 45
6.4 Variazione dell'efficienza teorica E in presenza di frammentazione, in funzione del numero di frammenti n, per alcuni valori della probabilità d'errore sul frammento p. . . 46
6.5 Comportamento di IPsec in presenza di frammentazione. Variazione del throughput in funzione del numero di frammenti. 48
6.6 Confronto (nel caso $p = 0$) tra l'efficienza teorica e l'efficienza misurata sperimentalmente. 49
6.7 Confronto (nel caso $p = 0$) tra l'efficienza teorica includendo gli ack nell'overhead di pacchetto e l'efficienza misurata sperimentalmente. 50
6.8 Confronto (nel caso $p = 0$) tra l'efficienza teorica e l'efficienza misurata sperimentalmente per UDP. 50

A.1 Crittografia simmetrica. 58
A.2 Crittografia asimmetrica. 59

A.3 Cifratura e firma digitale. 60

Elenco delle tabelle

4.1 Principali differenze tra IPsec e TLS. 28

6.1 Prestazioni TCP su IPsec con 3DES e SHA1 a 10 Mbit/s 41
6.2 Prestazioni UDP su IPsec con 3DES e SHA1 a 10 Mbit/s 41
6.3 Prestazioni UDP su IPsec con 3DES e SHA1 a 100 Mbit/s 42
6.4 Prestazioni TCP su IPsec con 3DES e SHA1 a 100 Mbit/s 42
6.5 Prestazioni TLS con 3DES e SHA1 a 10 Mbit/s 43
6.6 Prestazioni TLS con 3DES e SHA1 a 100 Mbit/s 43
6.7 Prestazioni TLS con 3DES e SHA1 sopra IPsec (tunnel ESP con 3DES e SHA1) a 10 Mbit/s 43
6.8 Timestamp dell'handshake di IKE. 52
6.9 Timestamp dell'handshake di IKE "opportunistic". 53

Bibliografia

[1] William Stallings. *Network Security Essentials — Applications and Standards*. Prentice–Hall, first edition, 2000. Trad. it. *Sicurezza delle reti — Applicazioni e standard*, Addison–Wesley 2001.

[2] Stephen Kent, Randall Atkinson. *Security Architecture for the Internet Protocol.* Standards Track RFC 2401, IETF, Novembre 1998.

[3] Stephen Kent, Randall Atkinson. *IP Authentication Header.* Standards Track RFC 2402, IETF, Novembre 1998.

[4] Stephen Kent, Randall Atkinson. *IP Encapsulating Security Payload (ESP).* Standards Track RFC 2406, IETF, Novembre 1998.

[5] Douglas Maughan, Mark Schneider, Mark Schertler. *Internet Security Association and Key Management Protcol (ISAKMP).* Standards Track RFC 2408, IETF, Novembre 1998.

[6] Derrel Piper. *The Internet IP Security Domain of Interpretation for ISAKMP.* Standards Track RFC 2407, IETF, Novembre 1998.

[7] Hilarie K. Orman. *The OAKLEY Key Determination Protocol.* Informational RFC 2412, IETF, Novembre 1998.

[8] Dan Harkins, Dave Carrell. *The Internet Key Exchange (IKE).* Standards Track RFC 2409, IETF, Novembre 1998.

[9] Abraham Shacham, Robert Monsour, Roy Pereira. *IP Payload Compression Protocol (IPComp).* Standards Track RFC 2393, IETF, Dicembre 1998.

[10] Tim Dierks, Cristopher Allen. *The TLS Protocol — Version 1.0.* Standards Track RFC 2246, IETF, Gennaio 1999.

[11] Niels Ferguson, Bruce Schneier. *A Cryptographic Evaluation of IPsec.* `http://www.counterpane.com`, Aprile 1999.

[12] Radia Perlman, Charlie Kaufman. *Key Exchange in IPsec: Analysis of IKE. IEEE Internet Computing*, 4(6):50–56, Novembre 2000.

[13] Kurt Seifried. *The End of SSL and SSH?* Securityportal.com, 18 Dicembre 2000. `http://www.securityportal.com/cover/coverstory20001218.html`.

[14] Donald Eastlake. *Domain Name System Security Extensions.* Standards Track RFC 2535, IETF, Marzo 1999.

[15] Mike Thomas, Mike Froh, Matthew Hur, David McGrew, Jan Vilhuber, Sasha Medvinsky. *Kerberized Internet Negotiation of Keys (KINK).* Internet Draft draft-ietf-kink-kink-01, IETF, Luglio 2001. Work in progress.

[16] Ari Medvinsky, Matthew Hur. *Addition of Kerberos Cipher Suites to Transport Layer Security (TLS).* Standards Track RFC 2712, IETF, Ottobre 1999.

[17] Kurt Seifried. *The End of SSL and SSH? Follow-up.* Securityportal.com, 22 Dicembre 2000. `http://www.securityportal.com/seifried/sslssh-followup20001222.html`.

[18] Kurt Seifried. *IPSec — We've Got a Ways to Go (Part I).* Securityportal.com, 19 Luglio 2000. `http://www.securityportal.com/closet/closet20000719.html`.

[19] Yongguang Zhang, Bikramjit Singh. *A Multi-Layer IPsec Protocol.* Agosto 2000.

[20] Shiuh-Pyng Shieh, Fu-Shien Ho, Yu-Lun Huang. *Network Address Translators: Effects on Security Protocols and Applications in the TCP/IP Stack. IEEE Internet Computing*, 4(6):42–49, Novembre 2000.

[21] Donald Eastlake, Olafur Gudmundsson. *Storing Certificates in the Domain Name System (DNS).* Standards Track RFC 2538, IETF, Marzo 1999.

[22] R. Gieben. *Chain of Trust — The parent–child and keyholder–keysigner relations and their communication in DNSSEC.* Master's thesis, University of Nijmegen, Dicembre 2000.

[23] Michael Thomas. *Requirements for Kerberized Internet Negotiation of Keys.* Informational RFC 3129, IETF, Giugno 2001.

[24] John Kohl, B. Clifford Neuman. *The Kerberos Network Authentication Service (V5).* Standards Track RFC 1510, IETF, Settembre 1993.

[25] Michael C. Richardson, D. Hugh Redelmeier, Henry Spencer. *A method for doing opportunistic encryption with IKE.* Internet Draft draft-richardson-ipsec-opportunistic-02, IETF, Settembre 2001. Work in progress.

[26] Henry Spencer, D. Hugh Redelmeier. *Opportunistic Encryption.* Draft 4, FreeS/WAN Project, Maggio 2001. Work in progress.

[27] Robert Moskowitz. *Host Identity Payload Architecture.* Internet Draft draft-moskowitz-hip-arch-02, IETF, Febbraio 2001. Work in progress.

[28] Robert Moskowitz. *Host Identity Payload And Protocol.* Internet Draft draft-moskowitz-hip-04, IETF, Luglio 2001. Work in progress.

[29] Robert Moskowitz. *Host Identity Payload Implementation.* Internet Draft draft-moskowitz-hip-impl-01, IETF, Febbraio 2001. Work in progress.

[30] Brian Carpenter. *Internet Transparency.* Informational RFC 2775, IETF, Febbraio 2000.

[31] Robert Moskowitz. *Protocols and Puzzles.* Marzo 2000.

[32] Paul Mockapetris. *Domain Names — Implementation and Specification.* Standard RFC 1035, IETF, Novembre 1987.

[33] Paul Vixie. *Extension Mechanisms for DNS (EDNS0).* Standards Track RFC 2671, IETF, Agosto 1999.

[34] Bruce Schneier. *Applied Cryptography*. John Wiley, second edition, 1996.

Indice analitico

AH, *vedi* IPsec, AH
autenticazione, 5, 11
autorizzazione, 5

CBC, modalità, 60
cifrario a blocco, 59
crittografia, 58–63
    a chiave pubblica, 58
    asimmetrica, 59
    simmetrica, 58

Denial of Service, 5, 25, 32, 33, 35
DES, 16, 58, 59
Diffie–Hellman, 11–13, 22, 30, 36, 60
digest, 62
disponibilità, 5
DNSSEC, 26, 29–36

ESP, *vedi* IPsec, ESP

firma digitale, 59
frammentazione, 14
FreeS/WAN, *vedi* IPsec, implementazioni, FreeS/WAN

hash - funzione di, 62
HI, *vedi* HIP, Host Identity
HIP, 27, 29, 32–36
    HIT, 33, 35
    Host Identity, 33, 35
    LSI, 33, 36
HIT, *vedi* HIP, HIT

IKE, 7, 10–13, 16, 29–35, 51–52
    aggressive mode, 11, 12
    main mode, 11, 25
    quick mode, 11, 12
integrità, 5
integrity check value, 9, 10
IPComp, 16, 24
IPsec, 7–17, 24–27, 29–33, 35, 38, 40–49
    AH, 7–9
    architettura, 7–8
    elaborazione, 13–14
    ESP, 7, 9–10, 33, 35
    IKE, *vedi* IKE
    implementazioni, 16–17
        bump–in–the–stack, 16
        bump–in–the–wire, 16
        FreeS/WAN, 16, 30, 32, 38, 64
        PGPnet, 17
        Windows 2000, 17
    modalità trasporto, 7–10, 35
    modalità tunnel, 7–10, 27, 35
    SAD, 8, 13
    security association, 7, 9–11, 13, 35
    SPD, 8, 13
    SPI, 9, 33, 36
    utilizzi, 14–15
IPv6, 7, 27
ISAKMP, 11, 30, 35

Kerberos, 26, 30
KINK, 30

Linux, 16, 38, 39, 64
LSI, *vedi* HIP, LSI

MAC, 18, 19, 62
man–in–the–middle, 5, 26, 61
masquerading, 27, 34
MTU, 44
Multi-Layer IPsec, 27

NAPT, 27
NAT, 27, 32–34
netperf, 39–42, 47, 64
NIST Net, 39, 47, 64
non ripudio, 5

Oakley, 11
OpenSSL, *vedi* TLS, implementazioni, OpenSSL
opportunistic encryption, 30–32, 34, 52

perfect forward secrecy, 13, 51
PGP, 6, 17, 33
prestazioni, 38–53
    handshake, 49–53
    trasferimento dati, 39–49

replay attack, 8, 9
rete privata virtuale, 14, 15
riservatezza, 5
road warrior, 15
round trip time, 51
RSA, 62

SAD, *vedi* IPsec, SAD
security association, *vedi* IPsec, security association
Security Association Database, *vedi* IPsec, SAD
security gateway, 7, 13, 14, 25
Security Policy Database, *vedi* IPsec, SPD
SET, 6
SPD, *vedi* IPsec, SPD
SPI, *vedi* IPsec, SPI
SSL, 18, 22

throughput, 39
TLS, 18–27, 34, 38, 42–44, 46, 52
    Handshake Protocol, 18–22
    implementazioni, 23
        OpenSSL, 23, 38, 64
    Record Protocol, 18–19
    utilizzi, 23