Alma Mater Studiorum
Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Triennale in Informatica

VoIP su Symbian: Sicurezza e Multihoming

Tesi di Laurea in Architettura degli Elaboratori

Autore:
Lorenzo Maiani

Relatore:
Dott. Vittorio Ghini

Anno Accademico 2009-2010
Sessione III

Indice

Elenco delle figure

Introduzione

Negli ultimi anni si è assistito ad un crescente sviluppo delle tecnologie legate alle comunicazioni *VoIP*, in parallelo ad una sempre maggiore disponibilità di reti *wireless*. L'introduzione e la conseguente diffusione sul mercato di terminali mobili *multi-homed*, come gli *smartphone*, rappresenta una prospettiva assai interessante riguardo la possibilità di utilizzare questi *device* per effettuare traffico telefonico tramite, per l'appunto, un sistema *VoIP*.

Nonostante l'implementazione di un sistema di telefonia *IP-based* quale *VoIP* su dispositivi *multi-homed*, dotati cioè di più interfacce di rete, risulti una soluzione vantaggiosa, vi sono alcune problematiche e limitazioni che hanno reso impossibile finora l'utilizzo di tali dispositivi per effettuare traffico *VoIP*, come la capacità di utilizzo di più di un indirizzo *IP* nella stessa sessione, e la possibile difficoltà nel mantenere adeguati requisiti di *QoS* (*Quality of Service*) durante l'*handover*, cioè la transazione da una rete all'altra.

L'architettura *ABPS* che verrà presentata in questa tesi, offre soluzioni in grado di superare questi ostacoli, permettendo ad un *device* mobile *multi-homed*, di effettuare traffico *VoIP* rispettando i requisiti di interattività richiesti dalle applicazioni multimediali. *Always Best Packet Switching* è un modello di infrastruttura distribuita, ed è di particolare rilevanza poiché, con la sola estensione dei protocolli *SIP* ed *RTP*, senza quindi dover modificare i protocolli di rete in uso, permette l'identificazione e l'autenticazione mediante un identificatore, univoco per ogni utente e indipendente dalla rete di provenienza.

Nel primi due capitoli verranno presentati alcuni tra i protocolli che sono di supporto alle tecnologie *VoIP*, saranno analizzati alcuni limiti e problematiche relative alle esigenze di *Quality of Service* legate al mondo della telefonia attraverso *Internet*, e verrà effettuata una breve descrizione delle

reti *wireless*, dello standard *IEEE 802.11*, e del concetto di *Multihoming*.

Nel terzo capitolo verranno presentate alcune delle soluzioni classiche ai problemi di sicurezza delle comunicazioni, con lo scopo di introdurre i protocolli più sofisticati utilizzati da questa architettura, tra cui il protocollo di *key-agreement ZRTP*, oggetto principale di questa tesi.

Nel quarto capitolo sarà presentato il sistema *ABPS*, verranno affrontate le modifiche apportate ai protocolli *SIP* ed *RTP* per supportare le caratteristiche di questo sistema, e saranno spiegate le tecniche adottate per fornire sicurezza ed autenticazione.

Nel quinto capitolo verrà descritto brevemente il sistema operativo *Symbian OS*, e nel sesto saranno presentate le librerie *PJSIP*, utilizzate per implementare *ABPS* su un sistema *VoIP*, e *LIBZRTP*, libreria che implementa le funzioni dell'omonimo protocollo.

Infine verrà spiegata la vera e propria fase di sviluppo, riguardante l'integrazione della libreria *LIBZRTP*, utilizzata per effettuare uno scambio di chiavi tra due partecipanti in modo che essi possano accordarsi su parametri di sicurezza per poi poter effettuare una comunicazione sicura *SRTP*.

Capitolo 1

Telefonia VoIP

1.1 Introduzione al sistema VoIP

Con il termine *VoIP* (*Voice over IP*) si fa riferimento ad un insieme di tecnologie e protocolli che permettono comunicazioni vocali, effettuate attraverso reti basate sul protocollo *IP*.

Vi sono svariate differenze tra questa tecnologia e le classiche reti telefoniche a commutazione di pacchetto *PSTN* (*Public Switched Telephone Network*).

Per prima cosa si evidenzia la sostanziale differenza nella gestione della codifica vocale: mentre le classiche infrastrutture telefoniche veicolano il traffico vocale in forma analogica, nel contesto *VoIP* questo deve essere necessariamente codificato in formato digitale. Anche le classiche reti *PSTN* stanno via via adottando sistemi digitali per lo *switching*, ma, soprattutto per quanto riguarda il cosiddetto *ultimo miglio*, la tratta di cavo che connette le centrali telefoniche agli utenti finali, persiste ancora la presenza della trasmissione analogica.

1.2 Storia e caratteristiche

I primi studi in questo ambito vengono fatti risalire al 1974 quando l'*IEEE* (*Institute of Electrical and Electronic Engineers*) pubblicò un articolo intitolato *"A Protocol for Packet Network Interconnection"* [26], anche se sarebbero

passati ancora 7 anni prima della standardizzazione del procollo *IPv4* [32]. L'articolo prendeva in considerazione la possibilità di offrire supporto per la condivisione di risorse tra reti commutate eterogenee.

I primi sviluppi significativi ci furono dopo l'avvento di *Internet*, a metà degli anni novanta. In particolare nel 1995, quando alcuni appassionati informatici israeliani sperimentarono la prima comunicazione voce tra due computer, e, nello stesso anno, la *VocalTec*, compagnia di telecomunicazioni israelita, rilasciò il primo software, noto come *"Internet Phone Software"*, [27]. Per effettuare una comunicazione tra due computer era sufficiente un modem, una scheda audio, altoparlanti ed un microfono.

Lo sviluppo delle tecnologie *VoIP* era tuttavia ancora agli albori, in quanto mancava un protocollo standard per la trasmissione e la localizzazione degli utenti attraverso la rete *Internet*. I primi sforzi in tal senso portarono allo sviluppo del protocollo *H.323* [16] e alla successiva standardizzazione, avvenuta nel 1999, di quello che è attualmente lo *standard de facto* per le comunicazioni *VoIP*, cioè *SIP* (*Session Initiation Protocol*).

Da allora l'utilizzo e le evoluzioni di queste tecnologie sono stati in continua crescita, particolarmente nell'anno 2003 con la prima release di *Skype* e nel 2004 col proliferare dei *provider VoIP* commerciali [6].

Il passaggio che la telefonia classica sta gradualmente effettuando verso le reti basate sul protocollo *IP*, non costituisce solamente un'evoluzione dei protocolli per la trasmissione vocale, ma introduce una serie di vantaggi in termini di prestazioni e servizi, descritti nella sezione successiva.

1.2.1 Vantaggi rispetto alle reti telefoniche

Il primo miglioramento significativo che caratterizza il *VoIP* rispetto alle classiche linee telefoniche, è un uso più efficiente del mezzo trasmissivo.

Le reti tradizionali usano un metodo di trasferimento tra un nodo e l'altro (*switching*) detto *"a circuito"*. Questa modalità prevede che al momento della chiamata venga instaurato un percorso tra i nodi nella rete e che questa connessione rimanga in piedi per tutta la durata della telefonata. Ogni connessione sfrutta un canale e questo viene riservato durante tutta la chiamata.

Inoltre viene usata tutta la banda richiesta dal canale, che è una risorsa fisica, non modificabile, salvo interventi di potenziamento sulle infrastrutture.

L'assegnazione delle risorse è quindi statica: una volta assegnata la linea infatti diventa "occupata". La natura statica di tale risorsa, impedisce sia la distribuzione del traffico (salvo in presenza di *switch* digitali), che l'allocazione dinamica delle risorse.

Le tecnologie *VoIP*, invece, introducono numerosi benefici in termini di flessibilità:

- **Possibilità di veicolare molteplici chiamate attraverso uno stesso canale fisico**, senza la necessità di dover modificare le infrastrutture per aggiungere ulteriori linee;
- **Indipendenza dalla posizione fisica dell'utente**: è sufficiente un canale di accesso a Internet, o a qualsiasi rete interna basata su *IP*, per dare la possibilità ad un utente di registrarsi presso un *provider*, effettuare chiamate e segnalare la propria presenza per essere raggiunto;
- **Estensione dei servizi disponibili**: possono essere implementate funzionalità aggiuntive come video chiamate, trasferimento di file, lavagne condivise, etc ...

Un'altra serie di vantaggi, sicuramente ben più apprezzata dagli utenti, è legata all'aspetto economico, sia per quanto riguarda i costi di mantenimento e gestione per gli operatori, che i costi del servizio per gli utenti.
Con la tecnologia *VoIP* infatti:

- **Il traffico voce è veicolato attraverso le linee dati**. È evidente il vantaggio economico che può trarne un'azienda, che, con tale approccio, non è piu costretta ad affrontare costi di gestione separati, ma può unificare la trasmissione dati e quella vocale, traendo anche ulteriori vantaggi dalla flessibilità che offrono le tecnologie *VoIP*.
- **I costi per gli utenti sono basati sul reale consumo**, piuttosto che sul tempo effettivo di durata della chiamata. Infatti, il costo di una telefonata considerando la quantità complessiva di informazione trasferita, piuttosto che la durata di questa, è generalmente molto inferiore.

- **Servizi aggiuntivi sono forniti gratuitamente o a basso prezzo**. La facilità con cui un *provider VoIP* può introdurre estensioni al classico servizio vocale (ad esempio avviso di chiamata, redirezione automatica delle chiamate, etc ...), senza dover pagare costi aggiuntivi per le infrastrutture, si traduce in una necessaria diminuzione dei prezzi.

- **L'allocazione delle risorse è dinamica**, a differenza delle linee *PSTN*, che richiedono collegamenti fisici dedicati per ogni linea telefonica. Con questa modalità veicolare il traffico attraverso un'unica linea dati permette di applicare politiche di distribuzione del traffico, che garantiscono un utilizzo più efficiente della banda a disposizione.

1.3 Protocolli per il VoIP

Vengono presentati di seguito alcuni protocolli di rete, particolarmente importanti nel contesto *VoIP*. La scelta di un determinato protocollo, a discapito di un altro, può variare a seconda dei contesti e in certi casi può essere offuscata dall'uso di tecnologie proprietarie.

In Figura 1.1 si può vedere come interagiscono i vari protocolli, e come il flusso audio viene trasformato per essere trasferito verso l'altro *end system*.

1.3.1 IP: Internet Protocol

VoIP, come si evince dal nome stesso, è una tecnologia basata sul protocollo *IP*, costituente le fondamenta di *Internet*. L'*Internet Protocol* [32], appartenente al terzo livello dello stack *ISO/OSI* [40], è un protocollo di rete a commutazione di pacchetto che prevede la trasmissione di blocchi (chiamati *datagrammi* o pacchetti) includendo anche, in caso essi siano di grosse dimensioni, una loro possibile frammentazione e riassemblaggio con una conseguente trasmissione di *"small packet"* sulla rete provenienti da fonti e destinazioni rappresentate da *host*.

Una sua caratteristica è quella di essere un protocollo non orientato alla connessione (*connectionless*): esso effettua infatti la cosiddetta consegna *Best Effort*, in base alla quale non viene garantita la trasmissione affidabile

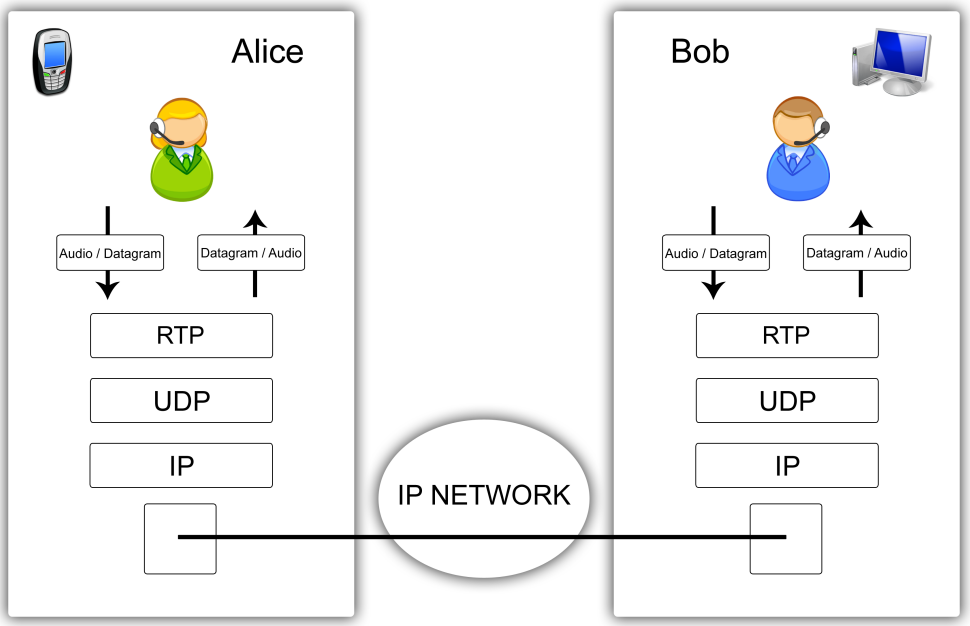

Figura 1.1: *Conversazione VoIP*

e ordinata dei datagrammmi, e non viene protetta dai duplicati, compito lasciato gestire da altri protocolli di livello superiore.

L'identificazione degli *host* avviene tramite l'assegnazione di un indirizzo a lunghezza fissa (indirizzo *IP*), nello specifico, è bene puntualizzare che l'indirizzo *IP* non è assegnato all'*host* fisico (computer, *server* o altri tipi di terminale) bensì ad ogni interfaccia di rete disponibile sull'*host* in grado di effettuare una comunicazione esterna verso *Internet*.

Tale assegnazione avviene tramite l'uso di diversi componenti e protocolli di seguito spiegati brevemente:

- ***DNS* (*Domain Name System*)**: realizzato tramite un *database* distribuito, rappresenta una delle caratteristiche più "visibili" di *Internet*, associando un indirizzo *IP* ad un nome simbolico e creando vantaggi per la memorizzazione di una fonte, sia a livello macchina (si pensi ad un sito *Word Wide Web*) sia a livello umano;

- ***Subnet Mask***: metodo utilizzato per definire il *range* di appartenenza

di un *host* all'interno di una rete *IP* al fine di ridurre il traffico di rete e facilitare la ricerca di un determinato indirizzo *IP* della stessa;

- ***Gateway* o *Router***: questi dispositivi possiedono più interfacce e collegano tra loro sottoreti diverse, inoltrando pacchetti *IP* da una all'altra. Per decidere su quale interfaccia inviare un pacchetto ricevuto, cercano l'indirizzo destinazione del pacchetto in una *tabella di routing*, che nei casi non banali viene gestita dinamicamente tramite uno o più *protocolli di routing*.

Attualmente esistono due versioni distinte del protocollo *IP*: *IPv4* ed *IPv6*. La prima rappresenta la versione corrente in uso, denominata così per distinguerla dalla nuova versione *IPv6* nata principalmente per soddisfare le esigenze dovute al crescente numero di terminali connessi ad *Internet*.

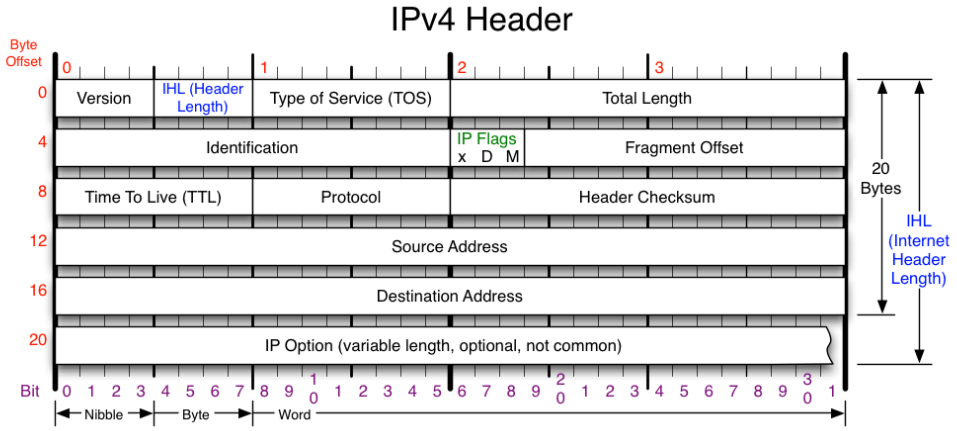

Figura 1.2: *Header IPv4*

1.3.2 UDP: User Datagram Protocol

Standardizzato nel 1980, è, insieme al sottostante protocollo *IP*, uno tra i protocolli fondamentali della rete *Internet*. Esso è un protocollo di trasporto, attraverso il quale è possibile scambiare messaggi, chiamati *datagram*.

Ogni canale di comunicazione tra *host* è determinato da una coppia indirizzo *IP* e porta. La mancanza di procedure di *handshake* o di sincronizzazione tra *host*, denota la natura semplice del protocollo *UDP*, il quale, come il

protocollo *IP*, non è orientato alla connessione, e, non garantendo quindi affidabilità della trasmissione, non si preoccupa di *datagram* duplicati, corrotti, persi o arrivati in modo non ordinato rispetto alla sequenza di invio. Questa affidabilità nella consegna può essere raggiunta a livelli superiori, mediante i protocolli a livello applicazione, quale ad esempio *RTP*.

L'integrità del singolo *datagram* è verificabile attraverso un valore di *checksum* a 16 bit, incluso nell'*header*, e generato con la stessa procedura descritta dal protocollo *TCP* (Transmission Control Protocol) [33].

Malgrado la sua scarsa affidabilità, viene spesso utilizzato nei sistemi *real-time* o di trasmissione multimediale, in quanto le applicazioni sensibili ai tempi di latenza spesso preferiscono la perdita di pacchetti piuttosto che sopportare l'*overhead* dovuto all'instaurazione di un canale affidabile.

Le differenze tra *UDP* ed *IP* sono minime: nel primo, infatti, vi è l'introduzione dei numeri di porta, grazie ai quali diviene possibile effettuare una multiplazione di più flussi di traffico diretti verso uno stesso indirizzo *IP*.

Un'altra sua caratteristica fondamentale è la possibilità di effettuare comunicazioni *broadcast* e *multicast*, particolarmente utile nel caso di conferenze multimediali tra più *host*.

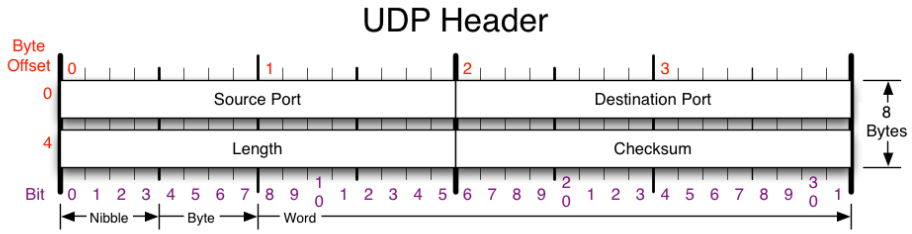

Figura 1.3: *Header UDP*

1.3.3 SIP: Session Initiation Protocol

È un protocollo di *signaling* a livello applicativo per la creazione, modifica e terminazione di sessioni multimediali che coinvolgono uno o più partecipanti. Queste sessioni comprendono telefonate attraverso *Internet*, distribuzione di traffico multimediale e conferenze multimediali [35].

È costruito per essere indipendente dal sottostante livello di trasporto, supportando quindi comunicazioni sia *UDP* che *TCP*, ed è a sua volta strutturato su diversi livelli logici.

Caratteristiche di SIP

I messaggi di invito *SIP* possono essere utilizzati per trasportare la descrizione della sessione, permettendo ai partecipanti di accordarsi su un insieme di protocolli multimediali compatibili.

Il protocollo fa uso di elementi chiamati *proxy server* per aiutare l'indirizzamento delle richieste verso l'attuale posizione di un utente.

Esso fornisce servizi di autenticazione degli utenti e di autorizzazione per l'utilizzo dei servizi. Implementa *policy* per l'instradamento delle chiamate e offre funzionalità agli utenti. Fornisce inoltre servizi di registrazione che permettono agli utenti di aggiornare il sistema sulla propria posizione corrente all'interno della rete.

Ci sono 5 aspetti di cui *SIP* si occupa per stabilire e terminare le comunicazioni multimediali:

- ***User location***: determinare l'*end system* che deve essere utilizzato per la comunicazione;
- ***User availability***: determinare la volontà del chiamato di prendere parte nella comunicazione;
- ***User capabilities***: determinare il tipo di *media* e i parametri per stabilire la comunicazione;
- ***Session setup*** (o ***Ringing***): stabilire i parametri di sessione per entrambi gli *endpoint*;
- ***Session management***: trasferimento e terminazione delle sessione, modifica dei parametri e invocazione dei servizi.

SIP non è un sistema di comunicazione integrato verticalmente, ma piuttosto un componente da utilizzare insieme ad altri protocolli *IETF* (*Internet Engineering Task Force*), per costruire un'architettura multimediale completa.

Non fornisce servizi, ma soltanto le primitive che possono essere usate per implementarli. La natura dei servizi offerti fa della sicurezza una problematica di particolare importanza. A tale scopo, *SIP* fornisce una *suite* di servizi di sicurezza che includono la prevenzione degli attacchi *DoS* (*Denial of Service*), supporto per l'autenticazione (sia dell'*user agent* verso il *proxy* che viceversa), integrità e cifratura. Il protocollo funziona sia con *IPv4* che *IPv6*.

Definizioni

Di seguito vengono fornite alcune definizioni che descrivono le entità coinvolte nel protocollo:

- ***Address of Record* (*AoR*)**: un *indirizzo di record* è composto da uno o più *SIP-URI* che puntano a un dominio con un servizio di locazione, attraverso il quale si è in grado di mappare un *URI* verso un altro dove l'utente potrebbe essere raggiungibile. Tipicamente le tabelle di questo servizio di locazione vengono popolate attraverso i servizi di registrazione. Un *AoR* può essere considerato l'indirizzo pubblico di un utente.

- ***Call***: il termine chiamata è usato in maniera informale per riferirsi ad un qualche genere di comunicazione tra *peer*, generalmente instaurata per avviare una conversazione multimediale;

- ***Client***: è un qualsiasi componente della rete che invia *SIP Request* e riceve *SIP Response*. Non è detto che un *client* debba per forza interagire direttamente con un utente umano. *User Agent* (*UA*) e *proxy* sono *client*.

- ***Conference***: con conferenza si intende una sessione multimediale che coinvolge più di due partecipanti;

- ***Call Stateful***: un *proxy* è *"call-stateful"* se mantiene le informazioni di stato per un *dialog* a partire dal messaggio iniziale di *INVITE*, fino alla richiesta di terminazione *BYE*. Un *call-stateful proxy* è sempre *transaction stateful*, ma non è detto che valga il contrario.

- ***Dialog***: un dialogo è una relazione *peer-to-peer* tra due *UA* che dura per un certo periodo di tempo. Viene stabilito attraverso messaggi *SIP*, come una risposta *2XX* a un *INVITE Request*. È identificato da un identificativo di chiamata, un *tag* locale e uno remoto.

- ***Home Domain***: il dominio che fornisce servizi *SIP* ad un determinato utente. Tipicamente è il dominio presente nell'*URI*, all'interno dell'*AoR* di una registrazione.

- ***Location Service***: è un servizio di localizzazione usato da un *proxy SIP* per ottenere informazioni sulla possibile posizione di un utente. Contiene una lista di *binding* tra un *AoR* con zero o più indirizzi. Il *binding* può essere creato e rimosso in vari modi, tipicamente attraverso un messaggio *REGISTER* che aggiorna lo stato del *binding*.

- ***Message***: un messaggio sono dati scambiati tra entità *SIP* come parte del protocollo. Si dividono in *Request* e *Response*.

- ***Method***: un metodo è la funzione primaria che viene invocata su un *server* attraverso una *Request*;

- ***Proxy Server***: è un'entità intermediaria che agisce sia da *client* che da *server*, con lo scopo di effettuare richieste da parte di altri *client*. Il ruolo primario di un *proxy* è quello di occuparsi del *routing*. Sono anche utili per forzare l'utilizzo di *policy*. Un *proxy* interpreta e, ove necessario, riscrive specifiche parti di un messaggio *Request*.

- ***Registrar***: è un *server* che accetta richieste di tipo *REGISTER*, e salva le informazioni che riceve nel *location service* del dominio che gestisce;

- ***Request***: messaggio *SIP* inviato da un *client* verso un *server*, con l'intento di invocare una particolare operazione;

- ***Response***: messaggio *SIP* inviato da un *server* verso un *client*, come risposta indicante lo stato di una richiesta precedentemente ricevuta;

- ***Server***: è un elemento della rete che riceve *SIP Request*, esegue operazioni e invia *SIP Response*. Esempi di server sono i *Proxy*, *User Agent Server*, *Redirect Server* e *Registrar*.

- ***Session***: dalle specifiche di *SDP*: "Una sessione multimediale è definita da un insieme di chiamanti, riceventi, e dagli *stream* di dati che viaggiano tra loro" [25]. Se viene usato il protocollo *SDP*, una sessione è definita dalla concatenazione del *SDP user name*, *session id*, *network type*, *address type* e gli *address element* nel campo sorgente.

- ***SIP Transaction***: una transazione si verica tra un *client* e un *server*, e comprende tutti i messaggi a partire dalla prima *SIP Request* del *client* fino all'ultimo *SIP Response* del *server*;

- ***Stateful Proxy***: è un'entità logica che conserva lo stato delle transazioni tra un *client* e un *server*;

- ***Stateless Proxy***: è un'entità logica che non mantiene lo stato della transazione tra *client* e *server*. Si limita ad inoltrare ogni *SIP Request* verso il *server* ed ogni *SIP Response* verso il *client*.

- ***User Agent Client (UAC)***: è l'entità logica che crea una nuova *SIP Request*, e usa la *macchina a stati* del *client* per inviarla. Il ruolo dell'*UAC* dura solamente per il tempo necessario a completare la transazione. In altre parole, se un'entità invia una *SIP Request*, agisce come *UAC*. Se successivamente riceve una *SIP Request*, allora assumer il ruolo di *UAS*.

- ***User Agent Server (UAS)***: è l'entità logica che genera un *SIP Response* in seguito ad una *SIP Request*. Il *Response* accetta, rifiuta o redireziona la *Request*. Come per l'*UAC*, il ruolo di *server* dura solo fino al completamento della transazione.

- ***User Agent (UA)***: è l'entità logica che può agire sia da *UAC* che da *UAS*.

Messaggi SIP

SIP è un protocollo testuale, derivato da *HTTP* (*HyperText Transfer Protocol*) [21], di cui riflette la struttura dei messaggi, composti da un *header* e un *body* opzionale.

La similitudine con *HTTP* risiede anche nella tipica logica di *Request-Response*. Ad ogni messaggio di risposta, come in *HTTP*, è associato un opportuno codice numerico di tre cifre che ne identifica la tipologia. La prima cifra indica la classe a cui appartiene tale risposta, mentre le restanti due identificano una specifica risposta all'interno di tale classe.

Le risposte appartenenti alla classe *1XX* sono provvisorie ed informative; tutte le altre sono invece definitive.

Lo standard prevede che per ogni richiesta invocata si possano ottenere una o più risposte provvisorie (classe *1XX*) ma al più una risposta definitiva.

Le classi definite sono le seguenti:

- **100-199**: indica una risposta informativa e provvisoria (es. quando si invita un utente ad una conversazione, una risposta di tipo *180* indica che questo è stato avvisato ma non ha ancora accettato la conversazione);
- **200-299**: indica che la richiesta è stata eseguita con successo;
- **300-399**: indica che la richiesta è stata reindirizzata;
- **400-499**: indica un fallimento della richiesta, imputabile al *client*;
- **500-599**: indica un errore da parte del *server*;
- **600-699**: indica un errore globale.

Ogni *SIP Request* contiene un campo chiamato *metodo*, che indica la funzione che il *client* vuole invocare sul *server*.

Di seguito alcuni metodi tipici del protocollo:

- ***INVITE***: viene usato per stabilire una sessione tra due o più *UA*. Il *body* di questo messaggio contiene informazioni relative alla sessione che si vuole stabilire. Tali informazioni vengono descritte e gestite tramite il protocollo *SDP*.

- ***ACK***: conferma la ricezione di una risposta definitiva relativa ad una precedente richiesta di *INVITE*. Quando un *UA* genera un *INVITE* può capitare che trascorra un discreto intervallo di tempo, che può durare anche svariati secondi, prima che questo riceva risposta. Quando e se l'*UA* destinatario riceve l'*INVITE* viene generata subito una risposta provvisoria (tipicamente *180 Ringing*), mentre l'eventuale risposta definitiva (*200 OK*) viene generata dal destinatario solo quando l'utente relativo ha accettato effettivamente la chiamata. Possono passare diversi secondi, di conseguenza, per accertarsi che lo *UA* chiamante sia ancora in attesa di instaurare la sessione, quest'ultimo ha il dovere di generare subito un *ACK* per confermare la sua presenza appena riceve il *200 OK*.

- ***CANCEL***: interrompe l'instaurazione di una transazione. Tipicamente viene usato quando lo *UA* che ha generato un *INVITE*, e non ha ancora ricevuto una risposta definitiva, decide di interrompere tale transazione. La richiesta di interruzione non ha effetto se la transazione era già stata portata a termine, ovvero non permette di interrompere una sessione già in corso.

- ***BYE***: abbandona una sessione attiva. Quando la sessione coinvolge due partecipanti l'abbandono da parte di uno di questi comporta la terminazione della sessione. Quando invece sono coinvolti più partecipanti (es. conferenza) l'abbandono di uno di questi non ha conseguenze sulla sessione.

- ***REGISTER***: informa un relativo *Registrar* che l'utente identificato da un determinato *SIP-URI* è rintracciabile presso una determinata lista di contatti, ovvero di *UA*. Tale lista di contatti contiene un *URL* per ogni *UA* sul quale l'utente desidera ricevere eventuali richieste. Specifica inoltre un intervallo di tempo (tipicamente un'ora), scaduto il quale, se l'utente non rinnova la propria registrazione, viene considerato non più rintracciabile. Di conseguenza un *UA* genera tale richiesta anche per rinnovare una precedente registrazione; in tal caso può anche aggiornare la lista dei contatti.

Sessione SIP

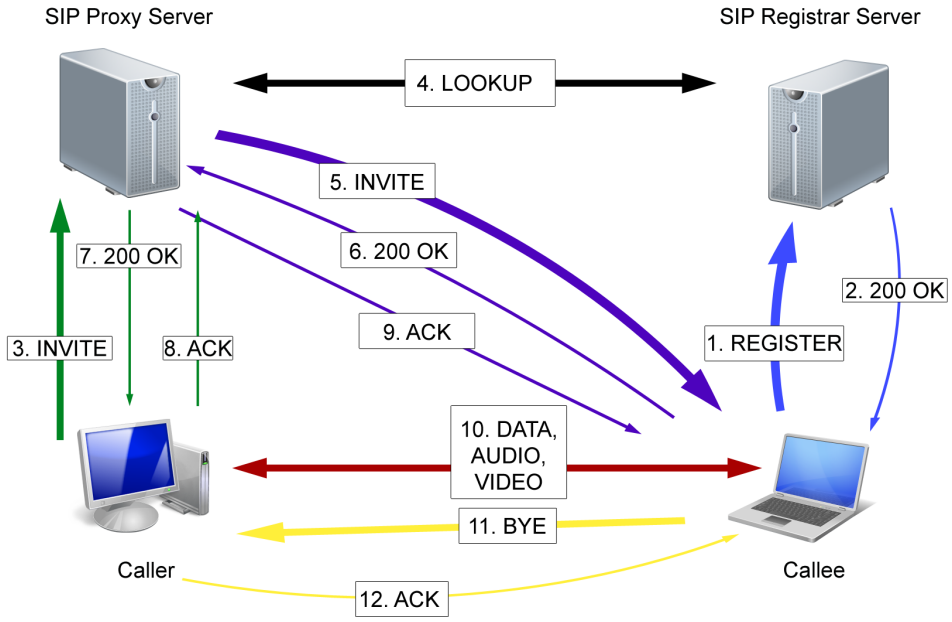

Figura 1.4: *Transazione SIP*

Per mettere in pratica tutto ciò che è stato spiegato precedentemente, in Figura 1.4 sono raffigurate le fasi fondamentali per l'instaurazione di un canale *SIP*, partendo da una registrazione, fino ad arrivare alla chiamata da parte di un chiamante (*Caller*) ad un chiamato (*Callee*).

1. Come prima operazione (fase 1 e 2) l'*UA* per poter essere contattato dovrà registrarsi al servizio *SIP* disponibile, effettuando una *REGISTER* al *SIP Registrar Server* e ricevendo una conferma (messaggio *200 OK*) da quest'ultimo. Una volta avvenuta la registrazione il contatto risulterà disponibile a ricevere ed effettuare chiamate.

2. Il chiamante inoltra una chiamata tramite l'invio di un messaggio di *INVITE* (fase 3) e, il *SIP Proxy Server*, che funge da intermediario momentaneo, verifica la presenza dell'indirizzo di destinazione (fase 4): se il controllo è andato a buon fine (l'utente richiesto esiste ed

è regolarmente registrato) redirige l'*INVITE* al destinatario il quale decide se accettare la chiamata o no (fase 5 e 6). In caso positivo inizia una fase di scambio, sempre con il *Proxy Server* da intermediario, di *ACK* per confermare il collegamento diretto (fase 6, 7 e 8).

3. Entrambi gli *User Agent* (sia chiamato sia chiamante) entrano in una comunicazione diretta, il *Server Proxy* non svolge più da intermediario e *SIP* lascia la gestione della comunicazione con il trasporto di dati *real-time* audio e/o video al protocollo *RTP*, che verrà descritto in seguito.

4. L'ultima fase è la fase di fine chiamata (fase 11 e 12): il controllo ripassa al protocollo *SIP*, uno dei partecipanti decide di terminare la comunicazione mandando un messaggio *SIP* di *BYE* ed il destinatario risponde con un *200 OK* di avvenuta recezione del messaggio.

1.3.4 SDP: Session Description Protocol

È un protocollo per la descrizione di sessioni multimediali. *SDP* gestisce l'annuncio, l'invito e altri metodi di inizializzazione di una sessione multimediale [25].

Non si occupa del trasporto dei dati, ma permette agli *endpoint* di negoziare parametri di sessione, come il tipo di trasmissione, formati, *codec* e tutta una serie di proprietà a cui spesso ci si riferisce col termine *"profilo di sessione"*.

Nato come componente di *SAP* (*Session Announcement Protocol*), ha trovato nel tempo svariati usi, sia in congiunzione a *RTP*, *RTSP* (*Real-time Streaming Protocol*) e *SIP*, sia come formato a sé stante per la descrizione di sessioni multicast.

1.3.5 RTP: Real-time Transport Protocol

Questo protocollo fornisce funzioni di trasporto *end-to-end*, adatte in particolare alla trasmissione *real-time* di dati come audio, video o dati di simulazione, attraverso servizi di rete *unicast* o *multicast*.

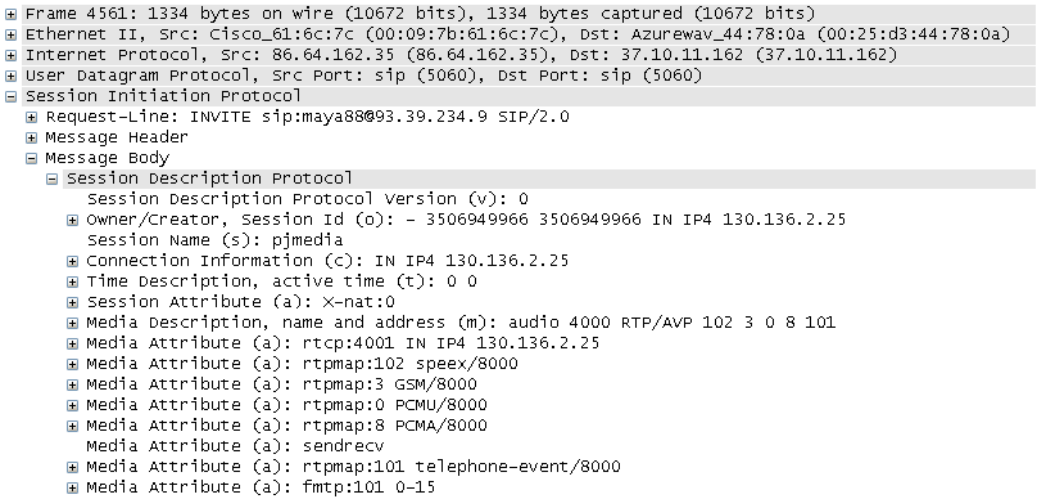

Figura 1.5: *Protocollo SDP inserito in un messaggio di INVITE*

È un protocollo di livello applicativo, indipendente dai livelli sottostanti di rete e di trasporto, anche se tipicamente viene usato attraverso lo scambio di datagram *UDP* [36].

Si noti che *RTP* non fornisce meccanismi per garantire *Quality of Service*, ma fa affidamento ai protocolli di livello inferiore, per la gestione di tale problematica. Non vengono garantite né la consegna affidabile dei *datagram* né l'ordine di arrivo sequenziale, tuttavia il *numero di sequenza* presente nell'*header RTP* permette al ricevente di ricostruirne l'ordine corretto. Insieme a *RTP* viene usato il protocollo *RTCP* (*Real-time Transport Control Protocol*), per monitorare la *QoS* e trasmettere informazioni che riguardano i partecipanti a una sessione in corso.

La sicurezza è parte fondamentale e sicuramente da non trascurare, in una sessione multimediale. Uno scenario costituito da due o più partecipanti ad una comunicazione multimediale in cui si presenta una terza parte, all'insaputa dei partecipanti, in grado di intercettare i flussi multimediali, registrarli o addirittura ascoltarli direttamente nel momento in cui avviene la discussione, non si deve assolutamente presentare.

RTP per quanto effettui un ottima pacchettizzazione di flussi audio e/o video non garantisce nessun tipo di integrità e sicurezza dei dati.

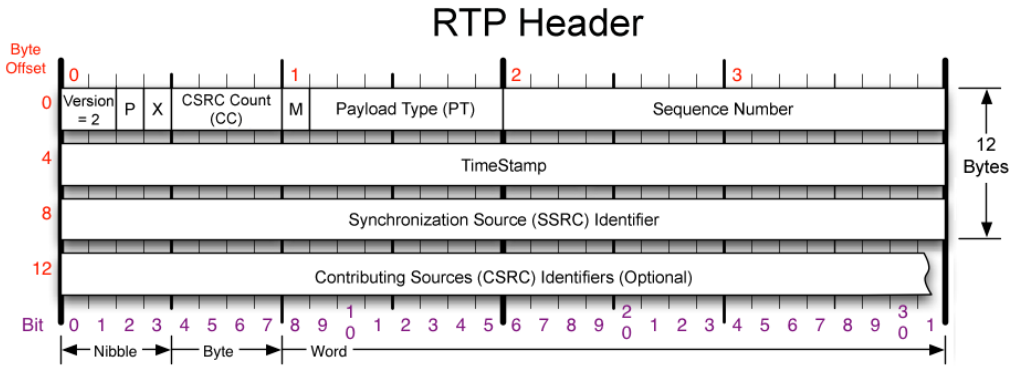

Figura 1.6: *Header RTP*

Nasce quindi l'esigenza di introdurre un sistema che garantisca piena sicurezza tramite l'utilizzo di tecniche quale la crittografia dei dati multimediali; tale sistema verrà descritto nella sezione successiva.

1.3.6 SRTP: Secure Real-time Transport Protocol

SRTP è un profilo per *RTP*, che garantisce le proprietà di confidenzialità, autenticazione dei messaggi e protezione da *replay attack*, ai datagram *RTP* e *RTCP*.

Il protocollo mette a disposizione un *framework* per la cifratura e l'autenticazione dei messaggi, definendo un *set* di trasformazioni crittografiche e permettendo di introdurne altre in futuro. Se supportato da un appropriato sistema di distribuzione delle chiavi, rende sicure applicazioni *RTP* sia *unicast* che *multicast* [11].

SRTP può garantire un livello elevato di *throughput* e una ridotta espansione dei pacchetti, fornendo una protezione adeguata attraverso tipi di rete eterogenei. Queste caratteristiche sono ottenute descrivendo trasformazioni di *default*, basate su tecniche di *additive stream cipher* per quanto riguarda la cifratura, e funzioni di *hash crittografiche* per l'autenticazione. La sequenzialità/sincronizzazione è garantita implicitamente facendo affidamento al campo *seqnum* del protocollo *RTP*.

Gli obiettivi si sicurezza che si pone *SRTP* sono di assicurare:

- **Confidenzialità** del *payload* del traffico *RTP* e *RTCP*;
- **Integrità** dell'intero pacchetto.

Questi servizi di sicurezza sono opzionali e indipendenti uno dall'altro, eccetto per quanto riguarda la protezione di integrità dei *datagram SRTCP* che è obbligatoria: alterazioni erronee o maliziose dei messaggi *RTCP* possono distruggere la possibilità di processare lo stream *RTP*.

Altri obbiettivi funzionali del protocollo sono:

- **Supporto per nuovi algoritmi crittografici**: il *framework* deve garantire la possibilità di poter utilizzare in futuro nuovi algoritmi, qualora ne vengano scoperti di piu robusti o efficienti;
- **Ridotto consumo di banda**: il *framework* deve preservare l'efficienza della compressione degli *header RTP*.

Le funzioni crittografiche predefinite garantiscono inoltre:

- **Costo computazionale ridotto**;
- **Footprint ridotto**;
- **Ridotta espansione dei pacchetti**, per garantire un basso consumo di banda;
- **Indipendenza dai livelli sottostanti di trasporto, rete e fisici**, garantendo un'alta tolleranza ai pacchetti persi e disordinati.

Queste proprietà assicurano che l'uso di *SRTP* è adatto alla protezione di *RTP/RTCP* sia in scenari *wired* che *wireless*.

Oltre a quanto citato finora, il protocollo ha altre caratteristiche aggiuntive che sono state introdotte per regolamentare la gestione delle chiavi e aumentare il grado di sicurezza. In particolare:

- Una singola *master-key* può fornire materiale crittografico per garantire confidenzialità e integrità per entrambi i canali *SRTP* e *SRTCP*. Questo obbiettivo è raggiunto utilizzando *session-key* diverse per i rispettivi canali, derivate in maniera sicura dalla *master-key*.

- Il *refresh* periodico della *session-key*, limita la quantita di traffico cifrato con la stessa chiave, rendendo piu difficile il processo di crittoanalisi.

- *Salting-keys* sono usate per proteggere il protocollo da attacchi di tipo *pre-computation* [31] e *time-memory tradeoff* [12].

Il problema che presenta *SRTP* è la mancata presenza di un proprio algoritmo di *key-agreement* necessario ad effettuare uno scambio di chiavi segrete (le sopra citate *master-key*) costituenti le chiavi utilizzate per la crittografia e l'autenticazione. È per questo che si rende necessario l'utilizzo e l'integrazione di un protocollo aggiuntivo, come *ZRTP* o *MIKEY* (*Multimedia Internet KEYing*) [10], utilizzati per la negoziazione dei parametri di sicurezza in un contesto crittografico e in grado di fornire i terminali di flusso di una coppia di segreti condivisi *"master-key"* e *"master-salt"*.

1.4 Quality of Service nei sistemi VoIP

Sono stati espressi in precedenza tutti i principali vantaggi e novità, che le tecnologie *VoIP* hanno introdotto nel mondo della telefonia. Rimangono tuttavia da analizzare alcune problematiche, soprattutto per quanto riguarda l'aspetto prestazionale.

Una conversazione telefonica, per essere udibile, impone vincoli stringenti di interattività. Il passaggio dalle classiche linee dedicate *PTSN*, alle reti dati con tecnologia *IP*, introduce degli *overhead* nella comunicazione.

Le metriche usate per misurare la qualità di una conversazione *VoIP* sono:

- ***One-way delay*** **(latenza)**: misura il tempo che impiega un pacchetto a percorrere la strada tra due *endpoint*. Al crescere di questo valore, si ha un degrado della qualità della conversazione, in termini di interattività. Le linee guida fornite da *ITU-T* [4] indicano una latenza di al piu di 150 ms per ottenere una qualità audio soddisfacente.

- ***Packet delay variation*** **(*Jitter*)**: misura la variazione nel tempo del *one-way delay*, ignorando i pacchetti persi lungo il percorso [18]. Il termine *Jitter*, preso in prestito a sproposito dal vocabolario elettronico

(dove indica la deviazione o il malposizionamento di alcune caratteristiche di un impulso, in un segnale digitale ad alta frequenza), spesso viene usato in riferimento all'*istantaneous packet delay-variation*, che misura la differenza tra la latenza di due pacchetti successivi. A valori negativi di questa metrica ci si riferisce con il termine *dispersion*, a quelli positivi con *clumping*. Anche la varianza sui tempi di latenza dei pacchetti rappresenta un'ulteriore metrica, utile a valutare la qualità della trasmissione.

- ***Packet Loss rate***: misura la percentuale di pacchetti persi. Alti valori di questo indice, provocano un degrado della conversazione in termini di qualità della voce. Per garantire una certa qualità del traffico vocale, non andrebbe superata la soglia del 10%. Una perdita di pacchetti, uniformemente distribuita nel tempo, ha un minore impatto rispetto alla perdita di diversi pacchetti consecutivi. I *codec* piu avanzati utilizzano algoritmi *PLC* (*Packet Loss Concealment*) per compensare questo fenomeno.

- ***Throughput***: misura il consumo di banda. I *codec* usati dalle applicazioni *VoIP* sono progettati per ridurre il consumo di banda del canale di trasmissione, arrivando a consumare al piu 64 Kbps in trasmissione.

Di seguito si elencano alcuni approcci classici a questi problemi prestazionali. Si noti che il sistema *ABPS* che verrà descritto in seguito, offre soluzioni originali e alternative ad alcune di queste problematiche.

Per quanto riguarda i tempi di latenza, non c'è molto che si possa fare, in quanto sono fortemente dipendenti dalla linea usata per la trasmissione. Ridurre il consumo di banda, può aiutare a non sovraccaricare il canale. La prima scelta opportuna è utilizzare protocolli adatti, come ad esempio *UDP*: per evitare di introdurre *overhead* non necessario (che aumenta utilizzando procedure di *handshake* tipiche di un protocollo affidabile come *TCP*), si preferisce in genere sopportare una perdita di pacchetti, seppur contenuta.

L'architettura *ABPS* in qualche modo aggiunge qualità al servizio, introducendo una notifica a livello applicativo dei pacchetti persi durante il primo *hop*, permettendo una tempestiva ritrasmissione.

Per quanto riguarda il *packet delay variation*, tipicamente il destinatario usa un *buffer* di adeguate dimensioni, per cercare di compensare adattivamente le variazioni.

Per ridurre la percentuale di pacchetti persi si usano tecniche di *PLC* (*Packet Loss Concealment*).
Le principali si possono riassumere come:

- ***Zero insertion***: i *frame* mancanti vengono rimpiazzati con 0;
- ***Waveform substitution***: i *frame* mancanti vengono rimpiazzati da porzioni di traffico vocale disponibili, che vengono replicate. Questa tecnica è molto popolare poichè di facile implementazione.
- ***Model-based Method***: è una delle tecniche più sofisticate. Sfrutta modelli del linguaggio per elaborare tecniche di interpolazione e di estrapolazione dei *gap* nella comunicazione.

Anche il sistema *ABPS* aiuta a mitigare questo problema, rendendo possibile una tempestiva ritrasmissione dei *datagram* perduti, almeno per quanto riguarda quelli persi attraverso il primo *link* fisico e rilevabili dal sistema *TED*, (*Transmission Error Detector*).

Capitolo 2

Wireless e Multihoming

2.1 Wireless

In seguito all'enorme diffusione delle reti *Wi-Fi* (*Wireless Fidelity*), dovuta sia all'abbattimento dei costi di produzione e di commercializzazione, sia alla facilità d'uso e d'installazione, si è giunti in breve tempo ad un'inevitabile e stretta associazione tra sistemi *VoIP* e reti *Wi-Fi* [7], grazie alle quali vi è la possibilità di usufruire di connettività mobile.

Con il termine *Wi-Fi* si indicano dispositivi in grado di collegarsi a reti locali senza fili (*WLAN*) basate sulle specifiche *IEEE 802.11*.

Le reti *Wi-Fi* sono infrastrutture relativamente economiche e di veloce attivazione in grado di realizzare sistemi flessibili per la trasmissione di dati usando frequenze radio, estendendo o collegando reti esistenti, ovvero, creandone di nuove.

L'architettura *Internet* è del tutto simile ai tradizionali *ISP* (*Internet Service Providers*), i quali forniscono un punto di accesso agli utenti che si collegano da remoto.

La fonte di connettività a banda larga può essere via cavo (*ADSL* o *HDSL*), oppure via satellite. Oggi esistono connessioni ad *Internet* satellitari bidirezionali, che consentono alte velocità di trasferimento dei dati sia in *download* che in *upload*. La trasmissione satellitare ha, tuttavia, tempi di latenza elevati: il tempo di attesa prima che inizi l'invio dei pacchetti è dell'ordine di 1-2 secondi, un tempo molto grande se confrontato ai pochi centesimi di secondo

necessari ad una connessione *DSL*.

La copertura *Wi-Fi*, permessa da dispositivi quali antenne, è generalmente di due tipi: *omnidirezionali* e *direttive*. Le antenne *omnidirezionali* vengono utilizzate di norma per distribuire la connettività all'interno di uffici, o comunque in zone private e relativamente piccole. Oppure, con raggi d'azione più grandi, si possono coprire aree pubbliche (come aeroporti, centri commerciali, ecc ...). Con le antenne *direttive* è invece possibile coprire grandi distanze, definibili in termini di chilometri, e sono utili proprio per portare la banda larga nei territori scoperti dalla rete cablata. In questo caso, è possibile aggregare più reti in un'unica grande rete, portando la banda in zone altrimenti scollegate.

Al giorno d'oggi molti dispositivi mobili offrono la possibilità di connessione tra numerose interfacce disponibili: *GPRS*, *UMTS* ed altro ancora. Ma sicuramente, dati i vantaggi, specialmente legati ai costi del servizio, la *Wi-Fi* rimane in assoluto la tecnologia più utilizzata per i sistemi *VoIP* su dispositivi mobili. Tale primato probabilmente in futuro verrà lasciato ad un nuovo sistema, una sorta di evoluzione di tale tecnologia: la *WiMax* (*Worldwide Interoperability for Microwave Access*), una tecnologia basata sulle specifiche *IEEE 802.16* (non ancora molto diffuse) che consente l'accesso a reti di telecomunicazioni a banda larga e senza fili, offrendo una maggior e decisamente più vasta copertura sul territorio, oltre ad apportare migliorie legate alla velocità di trasmissione dei dati.

2.1.1 Standard IEEE 802.11

Lo standard si occupa di definire specifiche, per un livello *MAC* (*Medium Access Control*) e diversi livelli fisici, per la connettività di stazioni fisse, portatili e mobili, all'interno della rete locale (*LAN - Local Area Network*).

Descrive inoltre delle prassi regolamentative per standardizzare l'accesso a una o più bande di frequenza in un contesto di comunicazione locale [2].

2.2 Multihoming

Con il concetto di *Multihoming* si intende un dispositivo che fa uso di più indirizzi *IP* associati a varie reti collegate. All'interno di questo scenario, l'*host multihomed* è fisicamente legato ad una varietà di connessioni dati associate a differenti interfacce di rete.

Quando si verifica una problematica di connessione riguardante un interfaccia (es. guasto nella rete, fallimento nell'instaurazione del collegamento, scollegamento causato da un basso segnale di ricezione, ecc...) un dispositivo *multihomed* è in grado di passare ad un'altra interfaccia di rete disponibile e funzionante. Inoltre, un dispositivo *multihomed* che opera tramite un protocollo *VoIP* su rete *wireless*, è in grado di usufruire di uno specifico adattamento e bilanciamento del traffico (per il singolo pacchetto) in relazione alla fluttuazione della disponibilità di banda e della latenza della rete.

2.2.1 Scenario Multihoming

Precedentemente si è spiegato il concetto di *Multihoming*, come esso può essere esteso a dispositivi mobili: *personal computer* o cellulari di nuova generazione, quali gli *smartphone*, dotati di più interfacce di rete ed i vantaggi e svantaggi che esso porta.

In Figura 2.1 è rappresentato un tipico scenario *multihoming handover*: si hanno due terminali *A* e *B*, rispettivamente uno *smartphone* dotato di due interfacce di rete (*Wi-Fi standard 802.11 b/g/n* e *3G*) ed un terminale dotato di un unica interfaccia di rete (anch'essa *Wi-Fi standard 802.11 b/g/n*), collegati alla rete per una comunicazione *VoIP*. Il cambio di *Access Point* e di interfaccia di rete è un fatto comune, basti pensare ad un ambiente urbano in cui un utente effettua spostamenti quotidiani, rendendo il cambio di copertura di rete obbligatorio. Risulta quindi necessario l'utilizzo del concetto di *Multihoming*, grazie al quale il problema della mancanza di un segnale specifico ad un interfaccia di rete, dovuta a varie problematiche, viene risolto tramite l'estensione dell'interfaccia *Wi-Fi* con una copertura *GPRS/UMTS* permettendo al dispositivo una continua connettività, il tutto gestito da un metodo che garantisce la modalità migliore di *switching* dell'interfaccia di rete, chiamato *ABC* (*Always Best Connected*).

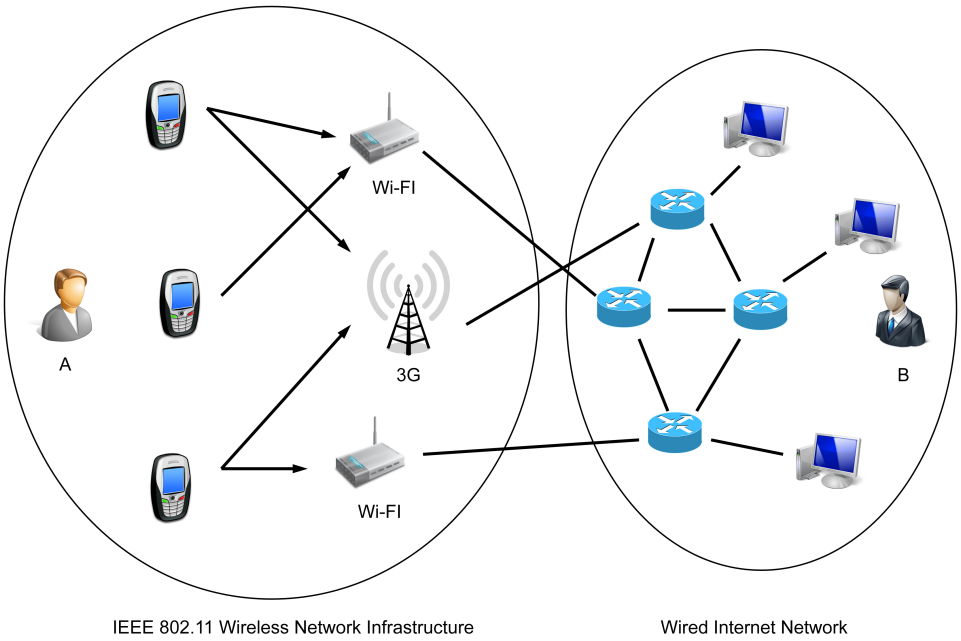

Figura 2.1: *Scenario Multihoming*

ABC è uno speciale *framework* il quale gestisce al miglior modo la scelta del miglior *NIC* (*Network Interface Controller*) disponibile al momento, da selezionare in caso di degradamento dell'attuale in uso.

L'architettura *Always Best Packet Switching* si basa su questo modello, ponendosi come scopo quello di offrire all'utente mobile il massimo della qualità di comunicazione sfruttando al meglio le capacità aggregate di tutte le interfacce di rete disponibili.

Capitolo 3

Sicurezza

3.1 Il concetto di sicurezza

La sicurezza è quel ramo dell'informatica che si occupa della salvaguardia dei sistemi informatici da potenziali rischi e/o violazione dei dati. I principali aspetti di protezione del dato sono: *confidenzialità*, *integrità* e *disponibilità*.

Di seguito, nella prima parte, verranno presentati i principali algoritmi di crittografia necessari a garantire confidenzialità ed integrità, i meccanismi di autenticazione ed alcuni metodi per eseguire *key-agreement* (scambio di chiavi).

Successivamente verrà presentato un particolare protocollo di sicurezza per eseguire *key-agreement*: *ZRTP*, in quanto protocollo utilizzato in appoggio a *SRTP*, per adempiere ad un sistema di sicurezza in *ABPS* (scambio di chiavi e crittografia dei pacchetti).

3.2 Confidenzialità

Nell'ambito della sicurezza informatica, con il termine *confidenzialità* si fa riferimento alla protezione dei dati e delle informazioni scambiate tra un mittente e uno o più destinatari nei confronti di terze parti.

In supporto della confidenzialità possono essere usati vari meccanismi, tra i quali il *controllo d'accesso* o la *crittografia*.

La diffusione di massa dei *personal computer* ha portato anche i normali cittadini ad avere l'esigenza di mantenere alcuni dati confidenziali, in particolare per quanto riguarda gli aspetti legati al commercio *on-line* (bancomat, carte di credito, etc ...)

Il contesto *wireless* pone ulteriori problematiche dovute all'utilizzo di un mezzo di trasmissione condiviso che, se non adeguatamente protetto, permette di rubare informazioni anche attraverso una semplice azione di *sniffing*, cioè l'intercettazione passiva dei dati che transitano in una rete.

I protocolli crittografici sono una pietra miliare per mettere in sicurezza le comunicazioni.

La parola *crittografia* deriva dal greco e significa *"scrittura segreta"*. Rappresenta l'arte e la scienza di occultare il significato.

Il termine *crittoanalisi* invece, indica lo studio di come forzare sistemi crittografici, che sono definiti come:

Crittosistema. *Un sistema crittografico è una quintupla (*$\mathbf{M}$*,* $\mathbf{K}$*,* $\mathbf{C}$*,* $\mathbf{E}$*,* $\mathbf{D}$*), dove* $\mathbf{M}$ *è un insieme di messaggi in chiaro,* $\mathbf{K}$ *è l'insieme delle chiavi,* $\mathbf{C}$ *è l'insieme dei messaggi cifrati,* $\mathbf{E} : \mathbf{M} \times \mathbf{K} \to \mathbf{C}$ *è l'insieme delle funzioni di cifratura e* $\mathbf{D} : \mathbf{C} \times \mathbf{K} \to \mathbf{M}$ *è l'insieme delle funzioni di decifratura.*

Lo scopo della crittografia è quello di riuscire a mantenere le informazioni segrete, assumendo che esista un avversario che vuole forzare il cifrario. La prassi comune assume che l'algoritmo usato per la cifratura sia noto, ma non la chiave (si conoscono soltanto $\mathbf{D}$ ed $\mathbf{E}$).

Gli attacchi che un avversario può effettuare sono i seguenti:

- ***Ciphertext-only attack***: l'avversario ha a sua disposizione solo il testo cifrato $\mathbf{E}$. L'obiettivo è scoprire il testo in chiaro $\mathbf{M}$ e possibilmente la chiave $\mathbf{K}$;

- ***Known Plaintext attack***: l'avversario conosce il testo cifrato $\mathbf{C}$ e quello in chiaro $\mathbf{M}$. L'obbiettivo è recuperare la chiave $\mathbf{K}$;

- ***Chosen Plaintext attack***: l'avversario può provare a cifare testi in chiaro $\mathbf{M}$, ricevendo il corrispondente testo cifrato $\mathbf{C}$. L'obbiettivo è recuperare la chiave $\mathbf{K}$.

Un buon sistema di cifratura protegge da tutti questi attacchi descritti, che solitamente usano approcci matematici e statistici. Si noti che, a causa della natura finita dei dati, non esiste un cifrario perfetto nel vero senso del termine, in quanto è sempre possibile indovinare la chiave.

Con il termine *cifrario perfetto* si vuole sottolineare che, per scoprire una chiave, non esiste un metodo più efficiente che provare tutte le combinazioni possibili. Tutte le attuali tecnologie crittografiche, sfruttano uno spazio ampio delle chiavi possibili, in maniera tale che sia impraticabile, o comunque molto dispendioso, per un calcolatore farne una visita esaustiva.

3.2.1 Algoritmi di Cifratura Simmetrici

Uno schema di crittografia *simmetrica* è caratterizzato dalla proprietà che, data la chiave di cifratura, sia facilmente calcolabile la chiave di decifratura. In molti dei casi questi sistemi utilizzano addirittura la stessa chiave per cifrare e per decifrare.

Più formalmente:

Algoritmo a chiave simmetrica. *Un cifrario è a chiave simmetrica se:*

$$\forall E_k \in C \quad e \quad \forall k \in K,\ \exists D_k \in D \mid D_k = E_k^{-1}.$$

Le famiglie principali sono i cifrari *a trasposizione*, matematicamente fondati su funzioni di permutazione, e i cifrari *a sostituzione*.

Gli algoritmi di cifratura a chiave simmetrica sono generalmente più efficienti di quelli a chiave asimmetrica, in quanto negli ultimi non è garantita la proprietà di *non ripudiabilità* di un messeggio.

Uno degli algoritmi più conosciuti, il *DES* (*Data Encryption Standard*), scelto come standard dal *Federal Information Processing Standard* (*FIPS*) per il governo degli Stati Uniti d'America nel 1976 e in seguito diventato di utilizzo internazionale, dal 1999 è ufficialmente considerato non più sicuro. Al suo posto, il *National Institute of Standards and Technology* ha proposto, nel 2001, l'algoritmo noto come *AES* (*Advanced Encryption Standard*) [1].

3.2.2 Algoritmi di Cifratura Asimmetrici

Nel 1972 Diffie ed Hellman [3] proposero un nuovo modello di crittografia, che utilizzasse due chiavi: una per cifrare e una per decifrare il messaggio.

La chiave per cifrare è pubblica: per trasmettere un messaggio in maniera sicura, lo si cifra con la chiave pubblica del destinatario. La chiave per decifrare è privata, e deve essere tenuta segreta, in quanto serve per decriptare i messaggi associati alla rispettiva chiave pubblica.

Poiché una delle due chiavi è pubblica, il sistema di cifratura deve rispettare le seguenti tre condizioni:

- Deve essere **computazionalmente semplice** cifrare e decifrare un messaggio, avendo a disposizione la chiave;
- Deve essere **computazionalmente impossibile** derivare la chiave privata, conoscendo quella pubblica;
- Deve essere **computazionalmente impossibile** scoprire la chiave privata effettuando un *Chosen Plaintext attack*.

L'algoritmo *RSA* (*Rivest, Shamir e Adleman*) [37], che utilizza particolari proprietà formali dei numeri primi con alcune centinaia di cifre, soddisfa queste richieste garantendo segretezza ed autenticazione. Tale algoritmo, infatti, non risulta sicuro da un punto di vista matematico teorico, in quanto esiste la possibilità che tramite la conoscenza della chiave pubblica si possa decrittare un messaggio, ma l'enorme mole di calcoli e l'enorme dispendio in termini di tempo necessario per trovare la soluzione (basata sulla scompozione in fattori primi di un numero), fa di questo algoritmo un sistema di affidabilità pressoché assoluta, tantoché è uno dei più utilizzati per la cifratura di firme digitali.

3.3 Autenticazione

Con il termine *autenticazione* si intende il processo di associazione tra un'identità e un soggetto. In altre parole è il processo attraverso il quale si è in grado di identificare un utente, e accertare che i messaggi scambiati con esso siano effettivamente stati generati da lui.

Esistono svariate tecniche in grado di garantire tale proprietà. Tutte si basano su un qualche genere di informazione condivisa tra i due sistemi coinvolti nel processo.

Di seguito se ne presentano due, che il sistema *ABPS*, estendendole opportunamente, utilizza per autenticare i *client*. La prima è l'*HMAC*, una tecnica usata per autenticare i messaggi trasmessi tra due *end-point*, mentre la seconda è il cosiddetto *challenge-response*, grazie al quale è possibile l'autenticazione tra un utente e un sistema remoto.

3.3.1 HMAC

Avere un metodo per verificare l'integrità di un'informazione trasmessa, o memorizzata su un dispositivo non affidabile, è una necessità nel mondo dei computer e delle comunicazioni. I meccanismi che forniscono tali controlli di integrità, basati sull'uso di una chiave segreta, vengono chiamati *MAC* (*Message Authentication Codes*). Vengono tipicamente utilizzati da due parti che, condividendo una chiave, sono quindi in grado di validare l'informazione trasmessa.

HMAC (*keyed-Hash Message Authentication Code*) [29] è un meccanismo per l'autenticazione di messaggi che fa uso di funzioni di *hash* crittografiche. Può essere utilizzato insieme ad una qualsiasi funzione iterativa come *MD5* o *SHA1*. La sicurezza di *HMAC* dipende dalle proprietà delle sottostanti funzioni di *hash*. In riferimento all'articolo [14], si può trovare una lunga argomentazione ed inoltre un'analisi crittografica.

Gli obbiettivi principali di questo protocollo sono:

- Utilizzare senza alcuna modifica le funzioni di *hash* disponibili. In particolare quelle che hanno buone prestazioni software, il cui codice è libero e largamente disponibile;
- Preservare le prestazioni originali delle funzioni di *hash*;
- Usare e gestire le chiavi in una maniera semplice;
- Avere un'analisi accurata della sicurezza crittografica del meccanismo di autenticazione, basandosi su ragionevoli assunzioni sulle funzioni di *hash* sottostanti;

- Garantire che le funzioni di *hash* siano facilmente intercambiabili, qualora se ne ne scoprano di più veloci e sicure.

Funzionamento di HMAC

Sia **M** il messaggio al quale la funzione *MAC* deve essere applicata, **H** la funzione *hash* utilizzata da *HMAC* (*MD5* o *SHA-1*), **K** la chiave segreta e condivisa per l'autenticazione del messaggio delle due parti, ed **L** la lunghezza in byte dell'*hash* di output (**L** = 16 per *MD5*, **L** = 20 per *SHA-1*). Si assume inolte che **H** calcoli l'*hash* dei dati iterando una funzione di compressione base su blocchi di dati di **B** byte.

- Se la chiave ha lunghezza maggiore di **B** byte ($|\mathbf{K}| > \mathbf{B}$) si calcola il valore *hash* di **K** cioè $\mathbf{H}(\mathbf{K})$ per ottenere una nuova chiave $\mathbf{K} = \mathbf{H}(\mathbf{K})$ di **L** byte;
- Se **K** ha lunghezza minore di **B** byte ($|\mathbf{K}| < \mathbf{B}$) si esegue la procedura di *padding* (come in *MD5*), in cui vengono aggiunti degli zeri per rendere la lunghezza di esattamente **B** byte.

Si fissano inoltre due differenti stringhe di **B** byte "*ipad*" e "*opad*" come segue:

$$ipad = \text{il byte 0x36 ripetuto } \mathbf{B} \text{ volte;}$$

$$opad = \text{il byte 0x5c ripetuto } \mathbf{B} \text{ volte;}$$

La funzione *HMAC* prende come *input* la chiave **K** ed il messaggio **M**, e produce come output il valore MAC di **M** con chiave **K** calcolato come segue:

$$HMAC_K(M) = H(K \oplus opad \,||\, H((K \oplus ipad) \,||\, M)) \quad [1]$$

[1] Il simbolo $\oplus$ corrisponde all'operatore logico *XOR* (o *OR esclusivo*), il quale restituisce il risultato *VERO* se, e solo se, uno solo dei suoi operandi è *VERO*. Il simbolo || indica la *concatenazione tra due stringhe*.

I passi svolti dalla funzione *HMAC* sono i seguenti:

1. $hash_1 = B \,||\, (\text{0x00} \times (B - \text{len}(K)))$;
2. $hash_2 = hash_1 \oplus ipad$;
3. $hash_3 = hash_2 \,||\, M$;
4. $hash_4 = H(hash_3)$;
5. $hash_5 = hash_1 \oplus opad$;
6. $hash_6 = hash_5 \,||\, hash_4$;
7. $hash_{result} = H(hash_6)$;

Scelta della chiave

La chiave da usare con *HMAC* può essere di qualsiasi lunghezza. In ogni caso, è sconsigliato l'uso di chiavi lunghe meno di **L** byte, in quanto diminuscono la sicurezza dell'algoritmo. Si preferiscono chiavi di lunghezza superiore a **L** byte, anche se aumentare ulteriormente la dimensione della chiave non porta miglioramenti significativi alla robustezza del sistema.

La chiave scelta deve essere casuale, e creata usando un generatore corretto di sequenze *pseudo-casuali* inizializzato con un *seme casuale*, e deve essere cambiata periodicamente.

Funzioni crittografiche di supporto ad HMAC

Come detto in precedenza, la peculiarità di *HMAC* è quella di non essere legato a nessuna funzione di *hash* particolare, questo per rendere possibile una sostituzione della funzione nel caso fosse scoperta debole. Nonostante ciò le funzioni più utilizzate sono *MD5* e *SHA-1*, che, in questo paragrafo, verranno descritte brevemente.

MD5 L'*MD5* (*Message Digest algorithm 5*) [34] indica un algoritmo crittografico di *hashing*, il quale prevede un *input* costituito da una stringa di dimensione arbitraria seguito da un *output* avente un'altra stringa di lunghezza

128 bit (chiamato *MD5 checksum* o *MD5 hash*) che può essere facilmente utilizzata per calcolare la firma digitale. Inoltre la codifica dell'*output* restituito avviene molto velocemente ed è tale da rendere improbabile il presentarsi di *output* simili.

Nonostante siano presenti algoritmi di *hasing* più veloci ed efficienti, quali *Whirlpool SHA-1* o *RIPEMD-160*, e nonostante sia stata provata in più occasioni la vulnerabilità a numerosi attachi, quali, per esempio, il *collision search attack*, basato, come si evince dal nome, sulle collisioni della funzione di compressione, *MD5* rimane un algoritmo ancora ampliamente usato, infatti, gran parte dei controlli di integrità su file vengono eseguiti tramite tale algoritmo.

Le operazioni di base su cui lavora *MD5* sono:

1. Fase di *Padding*: al messaggio in *input* da codificare (di lunghezza arbitraria) vengono aggiunti un insieme di bit fino a raggiungere la dimensione pari a 448 (mod 512). In particolare il primo bit aggiunto è 1 ed i restanti tutti 0;

2. Aggiunta della lunghezza: viene aggiunta una rappresentazione a 64 bit della lunghezza del messaggio prima che avvenisse il *padding*. Nel caso in cui la lunghezza superi i 264 bit vengono utilizzati solamente i 64 bit meno significativi del messaggio. Questi 64 bit rappresentano due parole da 32 bit ciascuna, che vengono poi appese al seguito del messaggio in input (dando la precedenza alla parola meno significativa), ottenendo quindi un messaggio di lunghezza multipla di 512 bit (16 parole da 32 bit ciascuna);

3. Nella terza fase avviene l'inizializzazione del *buffer* costituito da 4 *words* aventi ciascuna dei valori esadecimali di inizializzazione;

4. Nell'ultima fase vengono eseguite quattro funzioni logiche sull'insieme delle sedici parole ottenute nelle prime due fasi iniziali e sui quattro registri del *buffer*. In quanto rappresenti una fase estremamente lunga da spiegare, verranno semplicemente elencate le 4 funzioni tralasciando

il procedimento logico-matematico:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z);\ ^{2}$$

$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg A);$$

$$H(X,Y,Z) = X \oplus Y \oplus Z;$$

$$I(X,Y,Z) = Y \oplus (X \vee \neg Z).$$

Le funzioni non operano sulle parole intere, ma direttamente bit per bit. Ogni singola parola viene passata su ogni differente funzione.

Come *output* si ottengono 4 parole da 32 bit che, unite, dando sempre la precedenza alla parola meno significativa, compongono l'*hash* di 128 bit.

SHA-1 Più efficiente ma di minor impiego risulta l'*SHA-1* (*Secure Hash Algorithm*) [19], il più diffuso di una famiglia composta da cinque possibili varianti dell'algoritmo: *SHA-1*, *SHA-224*, *SHA-256*, *SHA-384* e *SHA-512*. È basato su un sistema simile all'*MD5*, se non per il fatto che produce un *digest* di 160 bit ricavato da un messagio in input di massimo $264 - 1$ bit.

Il principio di funzionamente dell'*SHA-1* è simile a quello dell'*MD5*.

3.3.2 Challenge - Response

Le tecniche di *challenge-response* sono un efficace sistema di autenticazione, che si occpuano di colmare alcune lacune del classico meccanismo basato su *nome utente* e *password*.

Un sistema di questo tipo infatti soffre di un problema fondamentale: la *password* può essere utilizzata per successive autenticazioni; se un attaccante sta in qualche modo facendo *sniffing* del traffico, può carpire il segreto ed utilizzarlo per autenticarsi in seguito. Con il *challenge-response*, invece, la

[2] Il simbolo $\wedge$ corrisponde all'operatore logico *AND* (o *congiunzione*), il quale restituisce il risultato *VERO* (1) se, e solo se, entrambi i suoi operandi sono a loro volta di valore *VERO* (1). Il simbolo $\vee$ corrisponde all'operatore logico *OR* (o *disgiunzione*), che restituisce il risultato *VERO* se almeno uno degli operandi è *VERO*. Il simbolo $\neg$ corrisponde all'operatore logico *NOT* (o *negazione*).

chiave trasmessa, basata su un qualche segreto condiviso, cambia ad ogni successiva autenticazione, garantendo che, anche se essa viene scoperta, non può essere usata per sessioni successive a quella in corso.

Il meccanismo è definito come segue:

Challenge-Response. *Sia* **U** *un utente che desidera autenticarsi presso un sistema* **S**. *Si supponga che* **U** *ed* **S** *si siano precendentemente accordati su una funzione segreta* $\boldsymbol{f}$. *Un meccanismo di autenticazione è* challenge-response *se il sistema* **S** *invia un messaggio casuale* $\boldsymbol{m}$ (challenge) *al sistema* **U**, *il quale risponde con il risultato della funzione* $\boldsymbol{r} = \boldsymbol{f}\,(\boldsymbol{m})$ (response). **S** *è in grado di validare* $\boldsymbol{r}$ *calcolando autonomamente il risultato della funzione.*

3.3.3 HTTP Digest Authentication

L'*HTTP Digest Authentication* [22] è basato su una verifica crittografica di una *password* in chiaro, condivisa tra *client* e *server*.

Esso è un protocollo *challenge-response*, dove il *client* che richiede il servizio viene sfidato a dimostrare di essere in possesso delle giuste credenziali. Utilizza la funzione *hash MD5* per permettere al *client* di trasmettere la chiave attraverso canali non sicuri. Il *server* può verificare l'*hash* della *password*, mentre un soggetto che tenti un attacco *MiTM* (*Man in The Middle*), una volta intercettato l'*hash*, non potrà risalire alla *password* in chiaro.

Il client *SIP* calcola un valore *hash* a 128-bit utilizzando la *password* condivisa con il *server*, attraverso l'algoritmo *MD5*.

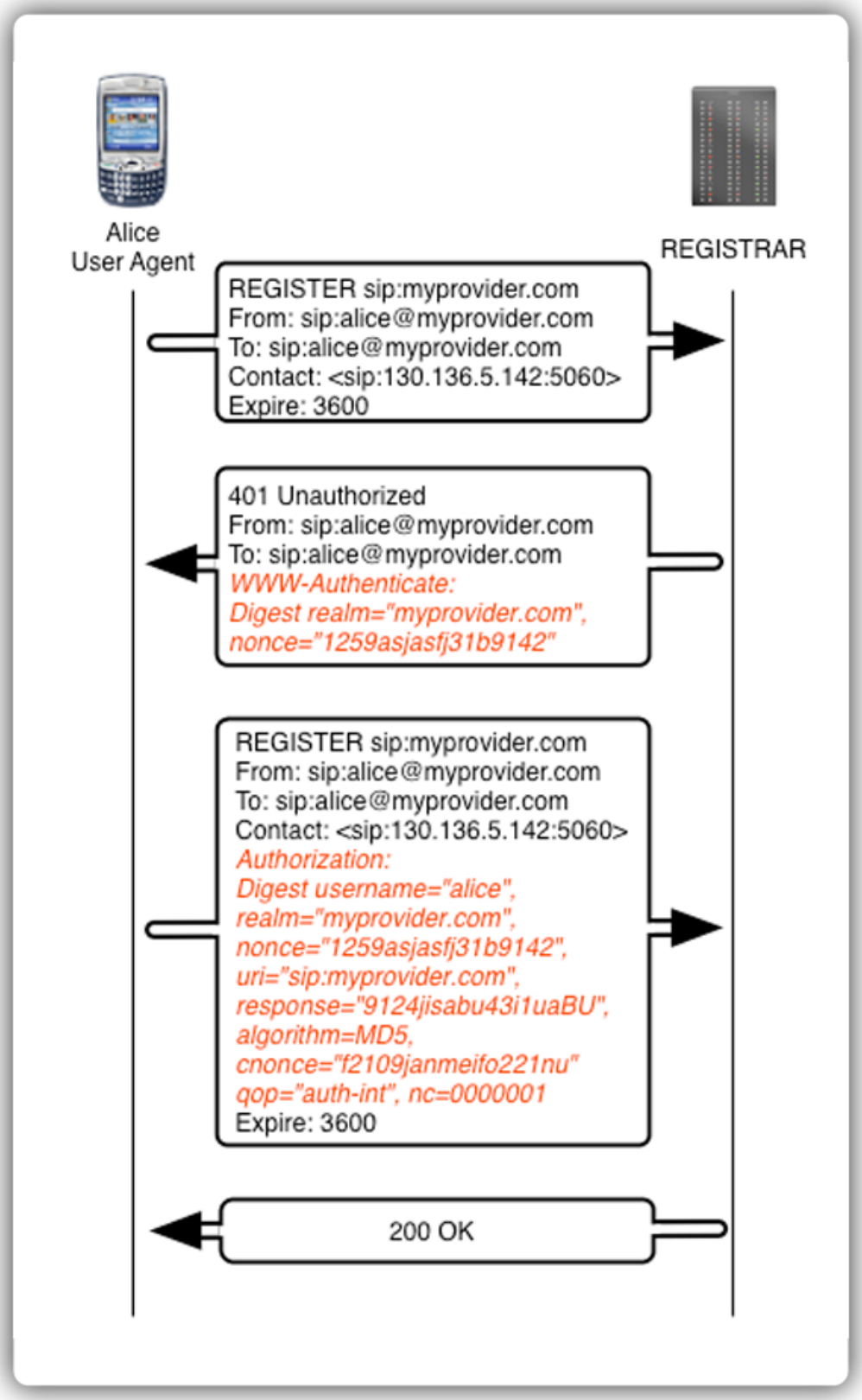

Figura 3.1: *HTTP Digest Authentication*

Come mostrato in Figura 3.1, l'autenticazione *digest* di una registrazione *SIP* è una procedura formata da 4 fasi, chiamata *four-way handshake*:

1. **Richiesta iniziale**: il *SIP User Agent Client* (*UAC*) invia una richiesta (*REGISTER*), per la quale è necessaria l'autenticazione al *SIP Registrar Server*).

2. **Challenge**: il *server* sfida il *client* rispondendo con un *errore 401* (*Unauthorized*), o *407* nel caso si tratti di un *server proxy*. Il messaggio contiene un *header WWW-Authenticate* che include i parametri:

   - ***Realm***: è il *real name*, e contiene il nome dell'*host* o del dominio del *server* che richiede l'autenticazione e viene utilizzato per far capire al *client* quale coppia *username/password* debba usare.
   - ***Quality of Protection* (*QoP*)**: può essere *"auth"* o *"auth-int"*, o può non comparire. Influenza il modo in cui il *response* deve essere calcolato, come sarà spiegato in seguito.
   - ***Nonce***: rappresenta il *challenge* generato dal *server* ed è utilizzato dal *client* per il calcolo del *response*. Sarà denotato come *server nonce*, per distinguerlo dal *client nonce*.
     Il *server nonce* consiste in un *timestamp* concatenato con un *hash* calcolato sul *timestamp* e su una chiave segreta del *server*. Questo schema offre una protezione contro i *replay-attack*, permettendo al *server* di verificare l'attualità del *nonce*, che sarà inserito negli *header Authorization* delle richieste successive.
   - ***Opaque***: è un campo utilizzato solo dal *server*, in cui può inserire ciò che vuole per scopi sconosciuti al *client*, il quale dovrà semplicemente ricopiarlo nei messaggi di richiesta successivi. Spesso viene utilizzato per memorizzare informazioni di stato relative alla sessione di autenticazione.
   - ***Algorithm***: può essere *"MD5"* o *"MD5-sess"*, e determina la struttura di un termine dell'equazione per il calcolo del *response*. Se viene utilizzato l'algoritmo *"MD5-sess"*, successivamente ad un'autenticazione con esito positivo, viene generata e condivisa una

session-key (chiave di sessione), il cui calcolo includerà i valori *nonce* iniziali del *server* e del *client*. La *session-key* permette al *server* di accettare dei *response* contenenti richieste consecutive del *client*, anche senza rigenerare un *nonce* diverso ad ogni richiesta. In questo modo si riduce il numero di messaggi scambiati, permettendo al *client* di calcolare e fornire i *response* autonomamente, senza dover essere sfidato ogni volta da un *challenge*. Una *session-key* è valida per tutta la durata di una sessione di autenticazione, che termina quando il *server* invia al *client* un nuovo *header "WWW-Authenticate"* o *"Authentication-Info"*. Se il parametro *Algorithm* è assente, si assume sia *"MD5"*.

- ***Stale***: se è *true*, indica al *client* che la sua precedente richiesta è stata rifiutata perchè il *nonce* era scaduto, ma il *response* era stato calcolato con la giusta *password*. In questo caso il *client* userà la stessa *password*, ma con il nuovo *server nonce*, per ricalcolare il *response*. Se *false*, allora deve essere richiesto all'utente del *client* di fornire nuovamente la *password*.

3. **Response**: l'*UAC* calcola il *response* e inoltra una nuova richiesta, questa volta includendo l'*header Authorization*, che conterrà i seguenti parametri:

   - ***Username***: il nome utente del *realm* specificato;
   - ***Realm***: lo stesso dell'*header WWW-Authenticate*;
   - ***Nonce***: lo stesso dell'*header WWW-Authenticate*;
   - ***Digest-URI* (*uri*)**: il valore del *Request-URI*;
   - ***Quality of Protection* (*QoP*)**: indica la qualità della protezione scelta dall'*UAC*;
   - ***Nonce count* (*nc*)**: indica il numero di *Request* distinte inviate dall'*UAC* usando lo stesso valore di *nonce* del messaggio attuale. Il *nonce count* è utilizzato come input nel calcolo del *response*, permettendo al *server* di rilevare dei *replay-attack*.
   - ***Client nonce* (*cnonce*)**: questo parametro deve essere presente se è stato specificato un parametro *QoP*. È il *nonce* generato

dall'*UAC* e utilizzato nel calcolo e nella validazione del *response*. Il suo scopo è di proteggere la *password* contro attacchi di tipo *Chosen Plaintext*. Un attacco di questo tipo potrebbe essere lanciato da un'entità maliziosa che si finge il *server SIP* e sceglie i valori di *nonce*. Il *client nonce* permette all'*UAC* di introdurre una maggiore entropia nell'output, controllata solo dal *client*.

- ***Response***: il response dell'*UAC*;
- ***Opaque***: lo stesso dell'*header WWW-Authenticate*;
- ***Algorithm***: lo stesso dell'*header WWW-Authenticate*.

4. **Autenticazione mutuale** assumendo che l'utente si sia autenticato con successo, il *server* può includere un *header Authentication-Info*, il cui scopo principale, nel contesto *SIP*, è quello di fornire un meccanismo di autenticazione mutuale.

   L'*header Authentication-Info* contiene i seguenti parametri:

   - ***Nextnonce***: fornisce il valore *nonce* che l'*UAC* deve usare per autenticare la richiesta successiva;
   - ***Quality of Protection* (*QoP*)**: indica la qualità della protezione fornita dal *server* al messaggio di *response*;
   - ***Client nonce* (*cnonce*)**: uguale al campo *cnonce* dell'*header Authorization* inviato dal *client*;
   - ***Nonce count* (*nc*)**: la copia del *server* del conto dei *nonce*;
   - ***Response authentication* (*rspauth*)**: un response calcolato dal *server* per provare che effettivamente conosce la *password* dell'utente. Vengono utilizzate le stesse equazioni del *client*, eccetto per il termine *A2*, che ha una struttura differente. Quando la qualità di protezione (*QoP*) è impostata ad *"auth"* o non è presente, il termine *A2* del *server response authentication* viene calcolato come:

     $$A2 = \text{“ :” } || \; URI$$

     Se invece è selezionato *"auth-int"*, la formula diventa:

     $$A2 = \text{“ :” } || \; URI \; || \text{ “ :” } || \; H(body)$$

Viene descritta in seguito la procedura utilizzata dal *client* per calcolare il *response*. Siano:

$$KD(secret, data) = H(secret \,||\, \text{“ : ”} \,||\, data)$$

dove H è la funzione di *hash*. Si noti che il valore dei parametri è da considerarsi senza le virgolette.

Quando la qualità della protezione (QoP) è impostata ad *"auth"* o *"auth-int"*, il *response* è calcolato come:

$$response =$$

$$KD(H(A1), nonce \,||\, \text{“ : ”} \,||\, nc \,||\, \text{“ : ”} \,||\, cnonce \,||\, \text{“ : ”} \,||\, QoP \,||\, \text{“ : ”} \,||\, H(A2))$$

Quando invece il parametro QoP è assente:

$$response = KD(H(A1), nonce) \,||\, \text{“ : ”} \,||\, H(A2))$$

Se il parametro del campo *Algorithm* è *"MD5"* o non è specificato nessun algoritmo, il termine *A1* è calcolato come:

$$A1 = username \,||\, \text{“ : ”} \,||\, realm \,||\, \text{“ : ”} \,||\, password$$

Quando viene usato l'algoritmo *"MD5-sess"*, *A1* diventa:

$$A1 =$$

$$username \,||\, \text{“ : ”} \,||\, realm \,||\, \text{“ : ”} \,||\, password \,||\, \text{“ : ”} \,||\, nonce \,||\, \text{“ : ”} \,||\, cnonce$$

Si noti come il *response* contenga un valore *hash* calcolato sulla *password* dell'utente: la *password* vera e propria non viene mai trasmessa.

Quando la QoP è *"auth"* o non è specificata, il termine *A2* si calcola come:

$$A2 = method \,||\, \text{“ : ”} \,||\, URI$$

dove *method* è il nome del metodo *SIP* invocato.
Se è selezionato *"auth-int"* invece :

$$A2 = method \,||\, \text{“ : ”} \,||URI \,||\, \text{“ : ”} \,||H(body)$$

Considerazioni sulla sicurezza dell'autenticazione HTTP Digest

L'autenticazione *HTTP digest* soffrre di alcune debolezze in termini di sicurezza, nonostante rimanga la tecnica di autenticazione più utilizzata dal protocollo *SIP*. Come già accennato in precedenza, la debolezza crittografica dell'algoritmo *MD5* è stata ampiamente studiata e documentata nel corso degli anni. La vulnerabilità agli attacchi basati su collisione dell'*hash*, ovvero quelli volti alla ricerca di due valori che producano lo stesso *hash MD5*, sono stati oggetto di una ricerca intensiva da parte dei crittoanalisti (si vedano [13] [17] [30] [39]). In particolare Vlastimil Klima nel 2006, in una versione rivisitata di un suo saggio, ha pubblicato l'algoritmo e il codice sorgente per trovare collisioni *MD5* in meno di 31 secondi, usando un normale *PC COLLISIONI*.

Allo stato attuale, quindi, non vi è più alcuna buona ragione per continuare ad usare *MD5*, considerando che esistono algoritmi più robusti come i già trattati *SHA-1* e *SHA-2*.

Gli unici campi del messaggio *SIP* di cui viene garantita l'integrità sono il *Digest-URI* e opzionalmente il corpo del messaggio *SIP*, selezionando il meccanismo di protezione *"auth-int"*. Un attacco *MiTM* potrebbe alterare facilmente il contenuto del campi *Contact* nell'*header* del messaggio *REGISTER*.

Saranno in seguito analizzate le modifiche introdotte dall'architettura *ABPS*, per rafforzare il meccanismo di autenticazione.

3.4 Key Management

Con il termine *Key Management* ci si riferisce a quei meccanismi di gestione delle chiavi crittografiche, in particolare:

- **Distribuzione** delle chiavi crittografiche;
- Meccanismi di **binding** tra un'identità ed una chiave;
- **Generazione** delle chiavi;
- **Mantenimento** delle chiavi;

- **Revoca** delle chiavi.

Il problema della distribuzione delle chiavi rappresenta uno dei punti più critici per garantire una reale confidenzialità delle comunicazioni.

Ogni algoritmo di cifratura, si basa implicitamente sul fatto che le due parti coinvolte nella comunicazione, condividano un qualche genere di informazione tra di loro: in un sistema di cifratura simmetrico è la chiave stessa che deve essere condivisa. In un sistema asimmetrico, invece, ogni parte coinvolta deve avere a disposizione una copia autentica della chiave pubblica del destinatario.

Le soluzioni a queste problematiche devono rispettare le seguenti condizioni:

- La chiave condivisa non può essere trasmessa in chiaro. O questa viene trasmessa in maniera cifrata, oppure le due parti coinvolte nella comunicazione devono essere in grado di derivarla. Le due entità coinvolte possono scambiare dati tra loro, ma una terza entità non deve essere in grado di derivare la chiave analizzando i dati scambiati.
- Le due entità possono decidere di dare fiducia ad una terza parte coinvolta;
- Il sistema di cifratura e i protocolli usati devono essere noti. L'unica informazione che deve rimanere segreta è la chiave crittografica.

Non esistono soluzioni standard per questo problema. La scelta di un protocollo rispetto ad un altro è fortemente dipendente dal contesto di utilizzo e dal grado di sicurezza che si vuole ottenere. Verranno presentati in seguito due protocolli, entrambi rilevanti per analizzare l'architettura *ABPS* che verrà descritta in seguito.

3.4.1 Protocollo Diffie-Hellman

È un protocollo crittografico per lo scambio di chiavi. Nato nel 1976 da un'idea di Whitfield Diffie e Martin Hellman [3], sulla base dei precedenti studi condotti sulla crittografia asimmetrica da parte di Ralph Merkle.

Il protocollo consente a due entità di accordarsi su una chiave di cifratura simmetrica, attraverso un canale non sicuro.

Descrizione dell'algoritmo

L'algoritmo si basa sull'apparente complessità computazionale, costituita dal calcolo di un logaritmo su di un campo discreto *GF(q)*, con un numero q di elementi. Sia:

$$Y = \alpha^X \bmod q, \qquad 1 \leq X \leq q-1$$

Fissato un α, elemento primitivo appartenente a *GF(q)*, si predispone X in maniera tale che:

$$X = \log_\alpha Y \bmod q, \qquad 1 \leq Y \leq q-1$$

Il calcolo di Y a partire da X è computazionalmente semplice, in quanto richiede al massimo $2 \times \log_2 q$ moltiplicazioni. Scegliendo q come un numero primo sufficientemente grande e x non primo ma grande anch'esso, allora il calcolo di X dato Y è computazionalmente infattibile.

Un utente, per generare una chiave sceglie un numero X_i in maniera casuale dall'insieme di interi $\{1, 2, \ldots, q\}$. X_i viene mantenuto segreto. Viene tuttavia reso pubblico il valore Y_i calcolato come segue:

$$Y_i = \alpha^{X_i} \bmod q$$

Quando l'utente i vuole comunicare con l'utente j, sceglie come chiave:

$$K_{ij} = X_i X_j \bmod q$$

Per calcolare il valore di K_{ij}, l'utente prende il valore pubblico Y_j e calcola:

$$K_{ij} = Y_j^{X_i} \bmod q = (\alpha^{X_j})^{X_i} = \alpha^{X_j X_i} = \alpha^{X_j X_i} \bmod q$$

L'utente j può calcolare la chiave procedendo in maniera analoga:

$$K_{ij} = Y_i^{X_j} \bmod q$$

Qualsiasi altra entità che volesse indovinare la chiave, deve calcolare:

$$K_{ij} = Y_i^{(\log_\alpha Y_j)} \bmod q$$

Come visto in precedenza, questo calcolo non è computazionalmente affrontabile se i parametri sono scelti sufficientemente grandi. Si noti come, scartando il numero segreto X_i alla fine di una sessione, venga rispettata la proprietà di *perfect forward secrecy*.

Perfect forward secrecy (PFS). *è una proprietà associata ad un sistema di scambio di chiavi crittografiche, che utilizza una chiave segreta persistente (long-term key) per derivare, di volta in volta, la chiave di sessione (short-term key) che verrà utilizzata per cifrare una singola conversazione.*

Un tale sistema gode della proprietà di PFS se garantisce che un attacante, qualora entrasse in possesso della chiave segreta, non possa essere in grado di:

- *decifrare i messaggi cifrati in precedenza;*
- *decifrare i messaggi che verranno cifrati in futuro, senza portare a termine con successo ulteriori tipi di attacco (es. MiTM).*

L'algoritmo *Diffie-Hellman* non è tutta via esente dal problema del *MiTM*. Non essendo infatti autenticata la comunicazione tra i due *end-point*, un attacante che si trova nel mezzo può avviare due sessioni *Diffie-Hellman* distinte con le altre due entità, e intercettare in questa maniera il traffico.

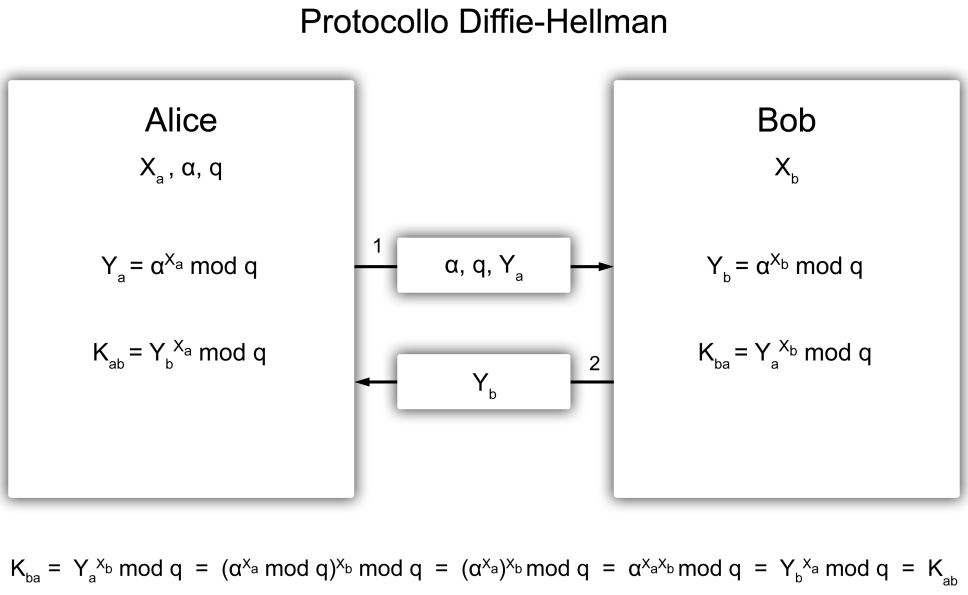

Figura 3.2: *Protocollo Diffie-Hellman*

In Figura 3.2 è raffigurato un tipico scenario in cui sono presenti due interlocutori, *Alice* e *Bob*, che vogliono effettuare uno scambio di chiavi segrete tramite il protocollo *Diffie-Hellman*.

1. viene scelto un numero casuale α (chiamato anche *generatore* o *radice primitiva*) modulo q, dove q rappresenta un numero primo;
2. *Alice* sceglie un numero casuale X_a e calcola il valore $Y_a = \alpha^{X_a} \bmod q$, inviandolo, insieme ai valori α e q, a *Bob*;
3. anche *Bob* sceglie un numero casuale X_b e calcola $Y_b = \alpha^{X_b} \bmod q$ e lo invia ad *Alice*;
4. a questo punto *Alice* calcola $K_{ab} = Y_b^{X_a} \bmod q$, mentre *Bob* calcola $K_{ba} = Y_a^{X_b} \bmod q$;
5. finite le fasi di calcolo entrambi l'interlocutori sono in possesso della chiave segreta K_{ab}.

3.4.2 Certificati e PKI (Public Key Infrastructure)

I sistemi di cifratura simmetrici soffrono di una debolezza dovuta all'utilizzo di una chiave condivisa. In questi sistemi infatti, non essendo possibile effettuare il *binding* tra una chiave e un'identità, non si possono applicare meccanismi di firma digitale.

Il sistemi di crittografia asimmetrica invece, riescono a garantire proprietà più robuste come la *firma digitale*. Un utente può apporre la firma a un messaggio cifrandolo con la propria chiave privata. Il destinatario, avendo a disposizione la chiave pubblica può calcolare la funzione inversa verificando che il documento non sia stato alterato. Nella pratica comune non è il documento ad essere firmato, ma un *hash* di questo generato opportunamente. Un documento firmato possiede la proprietà aggiuntiva di *non ripudiabilità*, che garantisce l'autenticità del messaggio per il motivo che esso può essere stato generato solo conoscendo la chiave privata.

Si consideri la seguente notazione:

$$X \to Y : \{Z\}_k$$

dove l'entità X manda all'entità Y un messaggio Z cifrato con la chiave k. Siano poi *Alice* e *Bob* due utenti che desiderano comunicare tra loro, e *Cathy* una terza entità esterna, considerata fidata da entrambi.

Si può ora introdurre la nozione di certificato come:

Certificato. *Un certificato* **C** *è un token che permette di effettuare il binding tra un'identità (es. Alice) ed una chiave crittografica* K_{Alice}*. Sia* **T** *un time-stamp relativo alla data di emissione,* **e** *la chiave pubblica e* **d** *la chiave privata associata; il certificato di un utente viene calcolato come:*

$$C_{Alice} = \{e_{Alice} \,||\, Alice \,||\, T\}d_{Cathy}$$

Le due tipologie più diffuse di certificati sono:

- ***Certificati X.509v3***: per validare un certificato, l'utente deve essere in possesso della chiave pubblica dell'entità che ha emesso il certificato chiamata *CA* (*Certification Authority*), che usa per decifrare il campo *signature*. Usa poi le informazioni ricavate da quel campo, per ricalcolare l'*hash* degli altri campi del certificato. Se questi corrispondono, la firma è valida così come la chiave pubblica. L'utente controlla infine il periodo di validità del certificato per assicurarsi che sia ancora valido.

  Se tutti certificati fossero rilasciati dalla stessa *CA*, allora la chiave pubblica di questa potrebbe essere distribuita attraverso un canale *out-of-band*. Allo stato pratico questo non è possibile. Abbiamo quindi a che fare con una molteplicità di *CA*, che complicano il processo di validazione. Due *CA* possono tuttavia certificarsi a vicenda (*CA cross-certified*), per permettere la propagazione del *trust*. Il protocollo *X.509v3* permette la creazione di *catene di firma* (*signature chains*), con l'unica richiesta che un certificato debba poter essere validato da quello che lo precede lungo la catena. Il protocollo suggerisce un'organizzazione gerarchica dei *CA* per minimizzare la crescita delle catene.

- ***Certificati PGP***: I certificati *PGP* (*Pretty Good Privacy*) sono basati sull'omonimo programma di cifratura asimmetrica, sviluppato da Philip Zimmermann nel 1991. *PGP* è ampiamente usato per garantire confidenzialità ai messaggi di posta elettronica, e per effettuare operazioni di firma digitale.

I certificati *PGP* differiscono da quelli *X.509v3* in differenti aspetti. Sono formati da unità chiamate *packets*. Una singola chiave pubblica, può essere firmata da più soggetti, tra cui il proprietario del certificato stesso (*self-signing*). Introduce diversi livelli di fiducia. A differenza dei certificati *X.509* che includono un elemento di fiducia, ma questo non è indicato nel certificato, quelli *PGP* indicano all'interno del certificato stesso il livello di fiducia, ma questo può avere valori diversi a seconda del firmatario. Inoltre è l'utente che attribuisce in qualche modo la fiducia a un certificato, a seconda della fiducia che ha nei rispettivi firmatari.

L'analisi approfondita dei certificati esula dallo scopo di questa tesi. Si fa notare tuttavia che, malgrado le proprietà crittografiche robuste di tali sistemi, l'utilizzo pratico deve essere effettuato con le dovute precauzioni e pone più di qualche problema.

Oltre alle difficoltà legate all'esistenza di differenti protocolli e differenti *CA* incompatibili tra loro, Bruce Schneier nell'articolo [20] pone i seguenti problemi:

- **Di chi ci fidiamo e perchè?**
  C'è un rischio correlato ad un uso impreciso del termine *trust*. Un *CA* è spesso definito *"trust"*. Nella letteratura crittografica, l'unico significato di questa parola è che si occupa in maniera sicura della gestione delle chiavi private. Questo non significa che si possa necessariamente riporre fiducia in un certificato di una *CA* per qualche ragione particolare, che siano micropagamenti o la firma su una transazione di milioni di dollari. Chi ha dato a quel *CA* l'autorità di garantire questo? Chi l'ha resa fidata?

- **Chi sta usando la mia chiave?**
  Su che supporto è conservata la chiave privata? Che garanzie ci sono che il sistema non sia stato compromesso e la chiave venga rubata? Questa tematica risulta particolarmente importante soprattutto in relazione alla proprietà di *non repudiabilità*. In certi stati (es. Utah e Washington) le leggi prescrivono che, se la chiave pubblica di un uten-

te è stata firmata da un'autorità di certificazione, questo è legalmente responsabile per l'utilizzo che ne viene fatto.

- **Quanto sicuro è il computer che effettua la verifica?**
  La verifica di un certificato necessità dell'utilizzo della sola chiave pubblica, quindi non ci sono segreti da proteggere. Tuttavia si fa uso di una o più *root public keys*. Se un attacante è in grado di aggiungere la propria chiave pubblica a quella lista, allora può emettere il proprio certificato che verrebbe trattato come legittimo. Non sarebbe utile conservare le *root keys* all'interno di *root certificate*, poiché un tale certificato sarebbe *self-signed*, non offrendo ulteriori garanzie di sicurezza.

- **Di che John Belushi si tratta?**
  I certificati solitamente associano una chiave pubblica a un nome. Come è stata effettuato questa associazione? Che garanzia ha l'utente che il nome associato ad una chiave pubblica, corrisponda effettivamente a quello della persona che cerca?

- **Il *CA* è un *authority*?**
  Ammesso che una *CA* abbia un'autorità nel rilasciare certificati, ma ha anche qualche autorità sul contenuto di questi? Per esempio un certificato *SSL* contiene il nome *DNS* del *server*. Esistono autorità qualificate per l'assegnazione di nome *DNS*, ma nessuna *CA* presente nella lista dei *browser* più comuni ha un'autorità di questo tipo.

- **L'utente fa parte del design di sicurezza?**
  L'applicazione che usa i certificati si occupa dell'utilizzo che l'utente fa di questi, oppure controlla solo gli aspetti crittografici del protocollo? Un normale utente prende decisioni su un acquisto web, attraverso una pagina protetta da un certificato *SSL*, sulla base di quello che viene mostrato nella pagina. Il certificato non viene mostrato e non ha necessariamente una relazione con il contenuto.

- **Si trattava di un *CA* o di un *CA* associato a una *registration authority*?**
  Alcune *CA*, per rispondere all'obiezione sulla non autorità per i contenuti certificati, hanno sviluppato una struttura di certificazione divisa

in due parti: una *RA* (*Registration Authority*) gestita dall'organizzazione che ha autorità sui contenuti, in comunicazione attraverso un canale sicuro con la *CA* che emette i certificati. Il modello *RA-CA* è senza dubbio meno sicuro, in quanto permette ad un entità priva di autorità sul contenuto (la *CA*), di rilasciare un certificato con quei contenuti.

- **Come ha fatto il *CA* ad identificare il proprietario del certificato?**
  Un'autorità di certificazione dovrebbe identificare un soggetto prima di rilasciargli un certificato. Come può farlo?

- **Quanto sicura è la pratica dei certificati?**

- **Perché nonostante tutto adottiamo l'utilizzo dei *CA*?**

Quanto esposto fino ad ora sui certificati, non è da intendersi come un'analisi approfondita della tematica, piuttosto come uno spunto per apprezzare le novità che offre il protocollo *ZRTP* esposto in seguito.

3.5 Tecniche di attacco

Verrano di seguito accennate alcune tra le più comuni tecniche di attacco ai sistemi crittografici.

- ***Man In The Middle* (*MITM*)**
  Questa tecnica è una variante attiva di *eavesdropping* . L'attaccante crea connessioni indipendenti coi due *end-point* vittima, e si interpone nella comunicazione occupandosi dell'inoltro dei messaggi tra di essi. L'attaccante di solito utilizza tecniche di *arp-poisoning* o di *dns-poisoning* per redirigere il traffico delle vittime inconsapevoli verso di lui. È uno degli attacchi più pericolosi. Qualsiasi protocollo di scambio di chiavi, che non si avvale di una qualche forma di autenticazione tra le parti coinvolte, è suscettibile a questo tipo di attacco. Il protocollo *Diffie-Hellman*, nella sua forma originale ne è vulnerabile.

- ***Replay Attack***
  Questa tecnica prevede la ritrasmissione di un messaggio precedentemente registrato, nel tentativo di exploitare un servizio o protocollo. Se un protocollo non è adeguatamente sicuro, un attaccante in possesso di un messaggio, (es. un pacchetto in transito per la registrazione ad un servizio), può ritrasmetterlo in seguito per provare ad autenticarsi verso il destinatario.

  Viene contrastato tipicamente attraverso l'uso di *nonce* (*number used once*) o *time-stamp* per la sincronizzazione.

- ***Brute Force Attack***
  Consiste nel violare un cifrario, provando tutte le chiavi possibili. Uno spazio delle chiavi sufficientemente grande, e la scelta di una chiave casuale, prevengono questo tipo di attacco.

- ***Bid-down attack***
  Spesso i primi messaggi scambiati tra due entità, che vogliono accordarsi sui parametri per stabilire una comunicazione sicura, non sono protetti poichè non è ancora avvenuta una reciproca autenticazione. Un attacante, sfruttando un *MiTM*, può rimuovere dalla lista degli algoritmi supportati quelli più sicuri, lasciando quelli più deboli. Il risultato di questo attacco è che, pur supportando meccanismi sicuri di autenticazione, le due entità coinvolte si accorderanno per utilizzare un metodo vulnerabile.

3.6 ZRTP

ZRTP [41] è un protocollo per lo scambio di chiavi crittografiche, che utilizza il protocollo *Diffie-Hellman* durante la fase *setup* della chiamata attraverso un *media path*. È trasportato attraverso la stessa porta usata da un flusso multimediale *RTP*, precedentemente stabilito attraverso un protocollo di *signaling* come *SIP*. Il protocollo permette di generare un segreto condiviso tra gli *end-point*, che può essere utilizzato per generare le chiavi e i *salt* per una trasmissione *SRTP*.

ZRTP possiede alcune caratteristiche interessanti, assenti in molti altri approcci alla cifratura di una sessione multimediale. Sebbene utilizzi un algoritmo di cifratura asimmetrico, non fa affidamento su alcuna *PKI*. Inoltre non utilizza nessuna chiave pubblica persistente.

Permette di rilevare un attacco *MiTM*, attraverso la visualizzazione di una *Short Authentication String* (*SAS*), che gli utenti possono verificare confrontandola verbalmente durante la chiamata.

Possiede la cosiddetta proprietà di *perfect forward secrecy*, distruggendo le chiavi di cifratura al termine di ogni sessione, precludendo in questa maniera la compromissione retroattiva del traffico telefonico, anche qualora la chiave venga scoperta in futuro.

Qualora gli utenti fossero troppo pigri per verificare di volta in volta la *SAS*, il protocollo offre comunque una ragionevole protezione contro gli attacchi *MiTM*, utilizzando una sorta di continuità sulla chiave. Viene memorizzato parte del materiale crittografico, per poter essere utilizzato in sessioni successive, che sarà mischiato col valore della chiave condivisa *DH* nella successiva chiamata. Le proprietà di continuità sulla chiave, sono analoghe a quelle del protocollo *SSH* [38].

Tutto questo è realizzato senza fare affidamento su *PKI*, certificati, *trust model*, *CA*, e altre complessità che affliggono il mondo della cifratura della posta elettronica. Inoltre, per la gestione delle chiavi, non fa alcun affidamento sul protocollo *SIP*, senza quindi necessitare di nessun *server* ausiliario. Realizza un scambio di chiavi *peer-to-peer* sopra uno *stream* di pacchetti *RTP*. Può essere usato scoprendo che un *client* lo supporta, senza una precedente indicazione del protocollo di *signaling*. Questo fornisce capacità *best effort* a *SRTP*. Riduce inoltre la complessità dell'implementazione, minimizzando la dipendenza dal livello di *signaling* e quello multimediale. Tuttavia, se il supporto a *ZRTP* viene indicato nella fase di *signaling*, attraverso l'attributo *zrtp-hash* nell'*header SDP*, allora il protocollo gode di alcune proprietà aggiuntive. Inviando un *hash* del messaggio *ZRTP HELLO*, il protocollo fornisce un *binding* tra il canale di *signaling* e quello multimediale. Se questo viene fatto attraverso un canale di *signaling* che fornisce protezione *ent-to-end* sull'integrità, allora lo scambio di chiavi è protetto automaticamente da un attacco *MiTM*. È concepito per sessioni multimediali *unicast*. Per le con-

ferenze multimediali, ogni coppia partecipante alla sessione deve effettuare una negoziazione *ZRTP* separata.

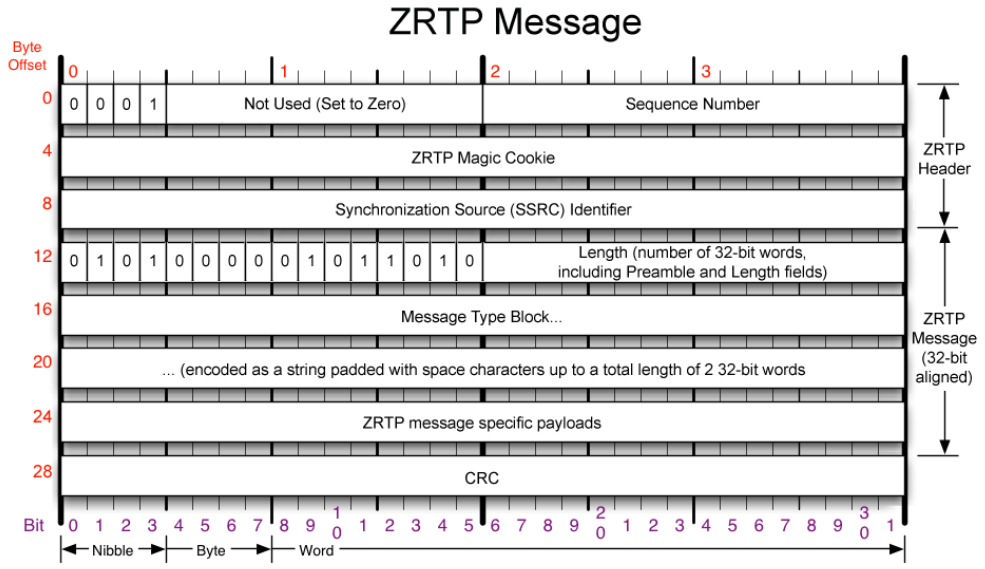

Figura 3.3: *Messaggio ZRTP*

In Figura 3.3 è mostrata la struttura di un pacchetto *ZRTP*. L'*header ZRTP* utilizza un formato che è in parte comune a quello *RTP*.

Esso specifica inoltre:

- Un numero di sequenza che viene inizializzato ad un valore casuale ed incrementato ogni volta che viene spedito un pacchetto *ZRTP*. Il numero di sequenza viene usato per stimare il numero di pacchetti persi o arrivati fuori ordine.

- Un valore *SSRC* (*Synchronization Source Identifier*) del flusso *RTP* con il quale *ZRTP* condivide gli *endpoint*.

Un pacchetto *ZRTP* trasporta un messaggio *ZRTP*, che comincia con un preambolo, seguito dalla lunghezza del messaggio, il tipo di messaggio ed il *payload*. Il pacchetto termina con un *CRC* (*Cyclic Redundancy Check*) di 32 bit per rilevare errori nello schema di trasmissione, come meccanismo supplementare al *checksum UDP*.

3.6.1 Analisi del protocollo

Il protocollo può essere suddiviso in 3 fasi:

- ***Discovery***
  Durante questa fase gli *end-point ZRTP* si scambiano informazioni sugli algoritmi supportati e le opzioni. Le informazioni sono trasportate attraverso gli ***HELLO message***.

  Quando possibile l'*hash* di questo messaggio dovrebbe essere stato precedentemente incluso nell'*header SDP*.

  L'*HELLO message* contiene i seguenti campi:

  - ***ZRTP version***;
  - ***hash type***;
  - ***cipher type***;
  - ***authentication method and tag lenght***;
  - ***key agreement type***;
  - SAS algotithms supported;
  - ***ZID (ZRTP IDentifier)***.

    Dopo aver terminato la fase di *discovery*, i due *end-point* possono scegliere gli algoritmi da utilizzare. Viene calcolata l'intersezione tra gli insieme di algoritmi supportati e, per ogni possibile scelta, viene utilizzato quello più robusto.

- ***Commit Contention***
  Dopo che entrambe le parti hanno ricevuto dei messaggi di *HELLO* compatibili, un messaggio ***COMMIT*** può essere inviato per inziare lo scambio di chiavi. L'*end-point* che invia il *COMMIT* viene chiamato *Initiator*, la sua controparte è chiamata *Responder*.

  In caso entrambi inviino un messaggio, il protocollo stabilisce delle regole di disambiguazione per assegnare i ruoli ai due *end-point*.

- ***Matching Shared Secret Determination***
  Questa parte del protocollo descrive come gli *end-point* debbano usare

l'insieme dei segreti condivisi s_1, *auxsecret* e *pbxsecret*, attraverso lo scambio dei messaggi ***DHPART1*** e ***DHPART2***.

Ogni *end-point* mantiene una *cache* dei segreti negoziati in precedenza con l'altra parte. Lo *ZID* è usato come indice di questa cache.

I messaggi *DHPART1* e *DHPART2* contengono una lista di *hash* di questi segreti, che permettono alle due entità coinvolte un confronto per capire quali segreti usare per il calcolo della chiave di sessione. Se non è disponibile nessun segreto condiviso, viene comunque trasmesso un valore casuale che assicuri il fallimento del confronto fra gli *hash*. Questa scelta evita che un attaccante possa sapere quali segreti siano condivisi tra gli end-point.

Il segreto condiviso ausiliario (*auxsecret*) può essere definito dall'*UA VoIP* come chiave disponibile *out-of-band*. In alcuni casi può essere fornito dal livello di *signaling*. Ci si aspetta che la maggiorparte dei classici endpoint *ZRTP* non usino questo tipo di chiavi.

Sia per l'*Initiator* che per il *Responder*, i segreti condivisi s_1, s_2 e s_3 sono calcolati in maniera tale che possano essere tutti utilizzati in seguito per calcolare s_0.

Il valore di s_1 può essere il valore rs_1 o rs_2 dell'*Initiator*, oppure *null*. La scelta viene effettuata nella seguente maniera:

- se il valore rs_1 dell'*Initiator* corrisponde al valore rs_1 o rs_2 del *Responder*, allora $s_1 = rs_1$
- se e solo se il confronto precedente fallisce, e se il valore rs_2 dell'*Intiator* corrisponde al valore rs_1 o rs_2 del *Responder*, allora: $s_1 = rs_2$;
- altrimenti non sono disponibili chiavi precondivise e $s_1 = null$.

Il segreto condiviso s_2 corrisponde al valore *auxsecret* se e solo se entrambi gli *end-point* calcolano lo stesso valore di *auxsecret*, altrimenti viene impostato a *null*. Lo stesso vale per il segreto condiviso s_3 che eventualmente può assumere il valore *auxsecret*.

Entrambe le parti calcolano l'*hash* dei messaggi di *HELLO*, scambiati attraverso il messaggio *DHPART1*. Gli *hash* vengono troncati ai 64 bit più significativi.

Il calcolo è il seguente:

1. $rs_1ID_r = MAC(rs_1, Responder)$;
2. $rs_2ID_r = MAC(rs_2, Responder)$;
3. $auxsecretID_r = MAC(auxsecret, Responder'sH3)$;
4. $pbxsecretID_r = MAC(pbxsecret, Responder)$;
5. $rs_1ID_i = MAC(rs_1, Initiator)$;
6. $rs_2ID_i = MAC(rs_2, Initiator)$;
7. $auxsecretID_i = MAC(auxsecret, Initiator'sH3)$;
8. $pbxsecretID_i = MAC(pbxsecret, Initiator)$;

Il *Responder* invia rs_1ID_r, rs_2ID_r, $auxsecretID_r$, e $pbxsecretID_r$ nel messaggio *DHPART1*. L'*Initiatior* invia rs_1ID_i, rs_2ID_i, $auxsecretID_i$, e $pbxsecretID_i$ nel messaggio *DHPART2*. Il *Responder* usa i valori calcolati localmente per effettuare il confronto.

- ***Key-Agreement***
  ZRTP mette a disposizione tre modalità per scambiare le chiavi:
  - *Diffie-Hellman mode*;
  - *Pre-shared mode*;
  - *Multistream mode*.

  Verrà analizzato solo lo scambio *Diffie-Hellman*, in quanto l'unico rilevante al fine dell'esposizione. La versione classica di questo algoritmo è stata presentata in precedenza, quindi ora verranno analizzate solamente le specificità introdotte dal protcollo *ZRTP*.

  Lo scambio di chiavi inizia con l'*Initiator* che sceglie un nuovo valore *Diffie-Hellman* in maniera casuale *svi* (*secret value initiator*).

La chiave pubblica da trasmettere è:

$$pvi = g^{svi} \bmod p$$

L'*hash commitment* viene eseguito dall'*Initiator*.
Il valore *hvi* (*hash value of Initiator*) viene calcolato così:

$$hvi =$$

$$hash(\text{Initiator's DHPART2 message} \,||\, \text{Responder's HELLO message})$$

Il valore calcolato viene troncato ai primi 256 bit. Le informazioni sull'*HELLO Message* del *Responder* sono incluse per prevenire un *bid-down attack*.

L'*Initiator* può quindi mandare il valore *hvi* all'interno del *COMMIT message*. L'uso dell'*hash commitment* all'interno del protocollo *Diffie-Hellman*, costringe un potenziale attaccante ad avere un solo tentativo a disposizione per indovinare il valore *SAS*. Questo fatto implica che il *SAS* può essere relativamente corto: un valore a 16 bit, per esempio, lascerebbe all'attaccante una possibilità su 65535 di non essere individuato.

Dopo che il *Responder* ha ricevuto il *COMMIT Message*, genera a sua volta una chiave segreta casuale *svr* (secret value responder), e calcola la chiave pubblica *pvr* così:

$$pvi = g^{svr} \bmod p$$

in maniera analoga a quanto spiegato per l'*Initiator*.

Dopo aver ricevuto il *DHPART2*, il *Responder* deve controllare che la chiave pubblica dell'*Initiator* non sia uguale a 1 o $p-1$. Un attaccante avrebbe potuto infatti immettere un valore di questo tipo, all'interno di un falso *DHPART2*, che avrebbe effetti disatrosi sulla sicurezza del protocollo. Se viene ricevuto uno di questi valori, la negoziazione delle chiavi deve essere terminata.

Se la procedura è andata a buon fine, il *Responder* può calcolare il proprio valore per l'*hash commitment* usando la chiave pubblica *pvi*

ricevutA con *DHPART2* e con l'*HELLO Message*, e confrontare il valore ottenuto con il valore *hvi* ricevuto nel messaggio di *COMMIT*. Se i valori calcolati sono differenti, allora è in corso un attacco *MITM* e lo scambio di chiavi va terminato.

Il *Responder* può quindi calcolare il risultato *Diffie-Hellman*:

$$DHResult = pvi^{svr} \bmod p$$

Dopo aver ricevuto *DHPART1*, l'*Initiator* effettua il controllo sulla chiave pubblica del *Responder*, affinché non risulti uguale a 1 o $p - 1$, come illustrato in precedenza.

L'*Initiator* può quindi inviare *DHPART2* e calcolare il risultato *Diffie-Hellman*:

$$DHResult = pvr^{svi} \bmod p$$

Capitolo 4

L'architettura ABPS

4.1 Stato dell'arte

Le architetture per l'integrazione di reti eterogenee, note con il nome di *Seamless Host Mobility Architectures*, sono responsabili di identificare univocamente un nodo *multihomed* (*MN*), permettendogli di poter essere raggiunto dagli altri nodi con cui ha già effettuato una connessione e selezionando la rete migliore in modo da proseguire la conversazione. Queste architetture non hanno una posizione ben definita all'interno del classico stack *ISO/OSI*, possono essere invece implementate ad ogni differente livello.

Nelle comunicazioni *VoIP*, l'indirizzo *IP* ha il compito di identificare il nodo, in quanto rappresenta l'univoca destinazione per i pacchetti provenienti dagli altri nodi con cui comunica. Trattandosi di un nodo mobile, esso riceverà un diverso indirizzo *IP* ad ogni ricollegamento, perdendo ad ogni selezione l'identità precedente. In questo modo, i nodi corrispondenti (*CN*) non riusciranno nuovamente a contattarlo prima di essere a conoscenza del suo nuovo indirizzo *IP*. Ne risulterà quindi una discontinuità all'interno della comunicazione. La soluzione comune a tutte le architetture di *Mobile Management*, anche se implementate in strati *ISO/OSI* differenti, si basa su due semplici principi:

- Definire un identificatore univoco per il nodo, indipendente alla provenienza dell'*host*;

- Fornire un servizio di localizzazione, sempre raggiungibile dai nodo corrispondente, per mantenere un'associazione tra l'identificatore univoco del nodo e il suo reale indirizzo di provenienza.

Il servizio di localizzazione è composto da un *Location Registry* (*LR*), attivo su un *server* con indirizzo *IP* fisso e pubblico e raggiungibile da qualsiasi *host*. Se il *CN* è a conoscenza dell'identificatore univoco del *MN*, sarà sufficiente mettersi in contatto con il *Location Registry* per recuperarne l'indirizzo *IP*, per poi inizializzare o ripristinare una comunicazione diretta. Il *Location Registry* è rappresentato da una funzione di *mapping* simile al *DNS* che opera come servizio esterno alla rete di provenienza dei nodi. Ogni *MN* utilizza il protocollo *SIP* per inviare un messaggio *REGISTER* al *server*, aggiornando la propria localizzazione. Quest'ultima soluzione non risulta efficiente perché, quando avviene una riconfigurazione dell'indirizzo *IP*, il tempo impiegato dal nodo mobile per comunicare al *server* l'aggiornamento introduce un ritardo inaccettabile, durante il quale viene interrotta la comunicazione.

4.2 Supporto per la Seamless Mobility

Viste le limitazioni delle soluzioni attualmente esistenti, ci si rende conto della necessità di una soluzione specifica. I requisiti essenziali per un completo supporto alla *seamless mobility* nello scenario appena descritto sono i seguenti:

- **Trasparenza a livello utente**: il *roaming* deve essere concluso il più velocemente possibile. L'utente non deve notare interruzioni nella comunicazione e quando questo non risulti possibile, è necessario ridurre al minimo tali interruzioni.

- **Trasparenza a livello rete**: non deve essere richiesto esplicito supporto nelle varie reti di accesso. Queste devono solo garantire connettività su protocollo *IP*.

- **Compatibilità**: la soluzione deve essere completamente compatibile con lo scenario preesistente, ovvero con relative entità e protocolli. In una comunicazione tra un terminale mobile ed uno fisso non deve essere

richiesto supporto specifico da parte del terminale fisso. Questo quindi deve rimanere ignaro della mobilità del terminale corrispettivo.

- ***QoS***: la mobilità del terminale *MN* deve essere gestita rispettando adeguatamente i requisiti di *QoS*;
- ***Full-Mobile***: deve essere supportata la possibilità che entrambi i terminali in comunicazione siano mobili. Essi devono quindi essere in grado di proseguire una comunicazione indipendentemente dai rispettivi spostamenti.
- ***NAT-Friendly***: la soluzione deve essere compatibile con la presenza di politiche di *NAT* sulle reti di accesso, in modo da non risultare di ostacolo alle preesistenti tecniche di *NAT-Traversal*.

Si nota che, in base al requisito *trasparenza a livello rete*, le reti di accesso sulle quali si sposta il terminale devono solo fornire connettività su protocollo *IP*. Non si vuole infatti porre alcun limite alle tipologie di *roaming* gestibili, in modo da fornire una mobilità completa. L'idea è quella di sfruttare il *Multihoming* per fornire la continuità della comunicazione in piena mobilità, gestendo opportunamente eventuali riconfigurazioni dell'indirizzo *IP* utilizzato.

4.3 Scenario ABPS

Lo scenario a cui fa riferimento il modello *Always Best Packet Switching* [23] ha come oggetto principale la comunicazione VoIP su un dispositivo mobile, che può essere un *laptop* o, con maggiore probabilità, un dispositivo di telefonia mobile, equipaggiato con più di un'interfaccia di rete (*NIC*) *wireless* come *WiMax*, *Wi-Fi* (*IEEE802.11/a/b/g/n*), *GPRS*, *EDGE*, *1xRTT*, *ZigBee* o altre.

La figura 4.1, a noi già familiare, in quanto molto simile a quella utlizzata per spiegare il concetto di *Multihoming*, illustra lo scenario *ABPS*, composto *in primis* da un terminale *VoIP multihomed* equipaggiato con due o più interfacce *wireless*, una *Wi-Fi 802.11b/g/n* e una *3G*, posizionato in una tipica area metropolitana che offre sia copertura *3G* che *Wi-Fi*.

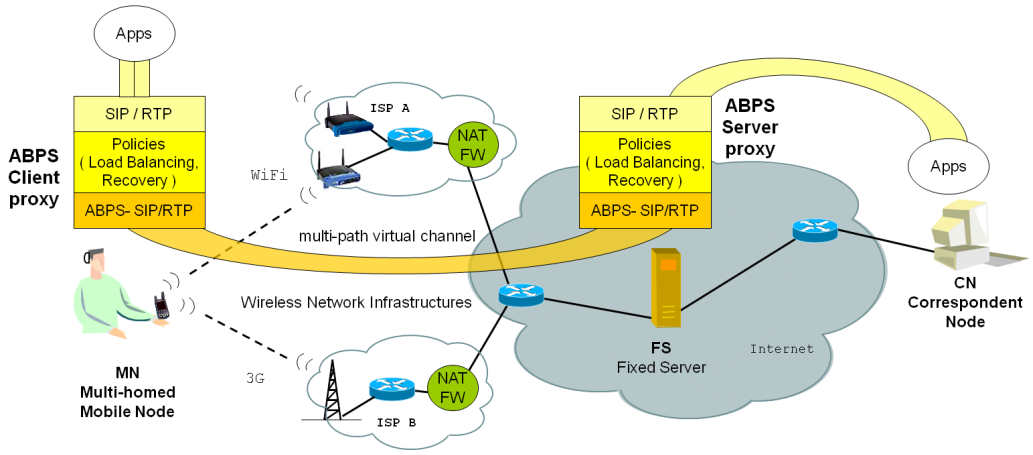

Figura 4.1: *Il sistema ABPS*

Il vantaggio di avere a disposizione interfacce di rete eterogenee sullo stesso dispositivo mobile è evidente soprattutto nel caso di interfacce *wireless*, perché rende possibile la connettività nel caso venga meno una delle connessioni, estendendo la copertura delle aree *Wi-Fi* con la copertura *GPRS/UMTS*, problema già affrontato in precedenza, e definito dalle specifiche *3GPP* come *Voice Call Continuity*(*VCC* [9]. Un altro vantaggio è la selezione, nel caso sia disponibile più di un'interfaccia, del *NIC* preferito per accedere ad *Internet*. Quest'ultimo approccio, già affrontato in precedenza, è utilizzato dalla funzione di *mobility management* delle reti *wireless*, seguendo il modello di *Always Best Connected*(*ABC*) [24], che, ricordiamo, suggerisce appunto di selezionare la migliore interfaccia *NIC* da usare come singolo punto di accesso a *Internet*. Nel caso in cui le prestazioni di un'interfaccia degradino eccessivamente, il dispositivo mobile rileverà una nuova interfaccia *NIC* preferita e la sostituirà alla prima.

L'architettura *ABPS* si basa su questo modello, ponendosi come scopo quello di offrire all'utente mobile il massimo della qualità di comunicazione, sfruttando al meglio le capacità aggregate di tutte le interfacce di rete disponibili. A causa di limitazioni tecniche ed economiche, l'utilizzo di servizi multimediali come le conferenze audio/video su *Internet* mediante i dispositivi precedentemente descritti presentano ancora problematiche irrisolte, nonostante le continue innovazioni in campo. Le applicazioni multimediali

VoIP e *Video on Demand* (*VoD*) risentono maggiormente di queste limitazioni tecniche perché richiedono esigenze di *Quality of Service* restrittive per poter offrire all'utente un buon livello di *Quality of Experience*.

Il sistema *ABPS* è una soluzione completa ed efficace di *QoS* e *Terminal Mobility* per il *VoIP wireless* basato su *SIP*, ed è composta da due entità:

- ***SIP Mobility***: un'entità realizzata a livello applicazione che gestisce le conseguenze di un *handover layer-3* (ovvero la riconfigurazione dell'indirizzo *IP*), che rispetti i requisiti precedentemente elencati;
- ***Vertical Mobility***: un'entità realizzata a livello *data-link* e *network* per monitorare lo stato di ogni interfaccia di rete presente sul dispositivo, gestendo in base ad opportune metriche e politiche la selezione dell'interfaccia di trasmissione.

La prima entità è realizzata mediante un *server* posto sulla rete pubblica, mentre la seconda è una applicazione eseguita direttamente sul terminale mobile. Il *server* rappresenta il punto d'ancora del terminale mobile ed ogni comunicazione *VoIP* del terminale passa attraverso di esso: qualunque entità *VoIP-SIP* che debba comunicare con il terminale contatta il *server* stesso, credendo di dialogare effettivamente con il terminale. In questo modo la mobilità del terminale viene gestita direttamente ed esclusivamente dal *server*.

In conseguenza di una scelta progettuale dovuta soprattutto a questioni pratiche, la versione corrente del sistema *ABPS* prevede che l'applicazione includa diversi livelli con diverse funzioni. L'idea originale era infatti quella di utilizzare un *proxy server ABPS* in locale, sul dispositivo *Mobile Node* (*MN*), per aggiungere ai pacchetti *SIP* ed *RTP* gli *header ABPS*, interfacciandosi con il *Fixed Server* (*FS*) in modo del tutto trasparente rispetto all'applicazione *client*. In questa nuova implementazione invece viene integrato tutto in un'unica applicazione *client* specifica per *ABPS*, divisa in più livelli, ad ognuno dei quali è assegnato un compito specifico dell'architettura. In particolare l'applicazione completa è composta da un livello più basso di selezione dell'interfaccia di rete per effettuare *handover* e *load balancing*, un secondo livello formato da un insieme di funzioni e strutture per effettuare

autenticazione mutuale e firma dei pacchetti, e di un terzo e un quarto livello che riguardano il *client SIP* vero e proprio, ovvero l'utilizzo dei livelli sottostanti per trasmettere al *Proxy Server FS* e lo sviluppo della *GUI* di un *softphone SIP*.

Le due entità descritte, il *client ABPS* su *MN* e il *proxy* su *FS* creano un *tunnel logico* tra esse permettendo la continuità della comunicazione in modo completamente trasparente, sia al terminale che alle altre entità in comunicazione con quest'ultimo.

4.4 Estensioni ABPS

Le estensioni *ABPS* progettate per i protocolli *SIP* e *RTP*, sono entrambe concepite per identificare univocamente il mittente di un messaggio, anche qualora questo cambi indirizzo *IP*. Forniscono inoltre le proprietà di autenticazione, integrità e, opzionalmente, confidenzialità ai messaggi *SIP* scambiati tra *client* e *server ABPS*.

Sono state studiate soluzioni differenti per i due protocolli in questione, al fine di soddisfare i requisiti di entrambi. In particolare, poichè le applicazioni multimediali scambiano un numero elevato di piccoli *datagram RTP*, è essenziale limitare sia la dimensione dei dati trasmessi, che il costo computazionale per i processi di cifratura e decifratura.

4.4.1 Estensioni ABPS al protocollo SIP

ABPS-SIP è un'estensione del protocollo *SIP* che permettte di stabilire un canale di comunicazione sicuro *SIP*, tra le due estremità di un canale virtuale *multi-path*. Ogni *client* condivide con il *server* una chiave pre-condivisa, scelta durante la fase di configurazione *off-line*. Seguendo le direttive del *National Institute of Standards and Technology* [15], la chiave pre-condivisa è usata senza mai essere trasmessa nella fase preliminare di autenticazione. In questa fase i messaggi e le funzioni di derivazione delle chiavi basate su funzioni crittografiche *HMAC* (*Keyed-Hash Message Authentication Code*), generano una chiave di sessione temporanea, condivisa tra *client* e *server*.

Per ogni messaggio *SIP* da trasmettere, viene utilizzata questa chiave insieme con funzioni *HMAC*, al fine di generare un *fingerprint* del messaggio.

Il protocollo *ABPS-SIP* aggiunge un *header* di 36 byte contenente:

- ***User Identifier* (*UID*)**: identifica univocamente il mittente;
- ***Sequence Number***: numero di sequenza che identifica univocamente un messaggio, usato per prevenire i *replay attack*;
- ***Fingerprint***: il *fingerprint* del messaggio trasmesso (nel calcolo sono inclusi i campi dell'estensione *ABPS-SIP* con esclusione del campo *fingerprint* stesso), usato per verificare l'integrità.

In questa maniera l'estensione *ABPS-SIP* garantisce le proprietà di autenticazione e integrità ai messaggi *SIP* trasmessi.

4.4.2 Estensioni ABPS al protocollo RTP

ABPS-RTP è un'estensione progettata come un insieme di protocolli standard di livello applicativo, che cooperano tra loro per fornire un canale *RTP* sicuro tra due *end-system*. In particolare, garantisce le proprietà di autenticazione, integrità e confidenzialità al flusso dati *RTP*, prescindendo dall'indirizzo *IP* del mittente. Il destinatario di un messaggio è in grado di distinguere due flussi *RTP* e identificare il mittente, anche quando questo cambia indirizzo *IP*. In questa maniera, *ABPS-RTP* supporta lo smistamento dinamico dei messaggi *RTP*, attraverso tutti i percorsi disponibili tra *client* e *server ABPS*.

Per la consegna dei messaggi, viene usata un'estensione standard di *RTP* chiamata *SRTP* (*Secure Real-time Transport Protocol*) [11]. *STRP* aggiunge un *header* con un codice di autenticazione, calcolato usando una chiave di cifratura associata ad un singolo flusso *RTP*.

Prima di poter configurare un flusso *SRTP* tra due *end-point*, è necessario generare una chiave condivisa utilizzando un protocollo di *key-agreement*. *SRTP* non specifica un protocollo particalore a riguardo: sono utilizzabili *MIKEY*, *SDES*, *DTLS*, *ZRTP*.

L'utilizzo del protocollo *ZRTP* all'interno dell'architettura *ABPS* è motivato dalle sue prestazioni, oltre che dalle forti garanzie che offre in termini di sicurezza.

4.5 Sicurezza e Autenticazione

Oltre alle estensioni ai protocolli *SIP* e *RTP* descritte in precedenza, l'architettura *ABPS* fornisce alcune innovazioni a supporto della sicurezza. In particolare viene esteso il meccanismo di *challenge-response* descritto in precedenza, e viene utilizzato il protocollo *ZRTP* per lo scambio di chiavi da utilizzare attraverso *SRTP*.

4.5.1 Autenticazione Challenge-Response

L'autenticazione *challenge/response* progettata per *ABPS*, e richiesta dal *server proxy ABPS*, è un'estensione dell'autenticazione *SIP/HTTP Digest* descritta in precedenza, con opportuni cambiamenti che garantiscono l'autenticazione mutuale e l'integrità del messaggio di autenticazione stesso.

Le prime due fasi dell'autenticazione sono identiche a quanto descritto in precedenza per *SIP/HTTP Digest*.

Nella fase 3 e 4 invece, sia *client* che *server ABPS* allegano al messaggio *SIP* il loro fingerprint calcolato con l'algoritmo *SHA-1*, che ne garantisce l'autenticità.

Di seguito viene esposta nel dettaglio la procedura di autenticazione *ABPS*, mostrata anche in Figura 4.2:

1. **Fase 1 - Richiesta Iniziale**: il *SIP User Agent Client* (*UAC*) invia una richiesta (es. *REGISTER*), per la quale è necessaria l'autenticazione al *server* (il *SIP Registrar*). (Identico a *SIP/HTTP Digest*).

2. **Fase 2 - *Challenge***:

   - il *server* sfida il *client* rispondendo con un *errore 401* (*Unauthorized*), o *407* nel caso si tratti di un *server proxy*. (Identico a *SIP/HTTP Digest*).

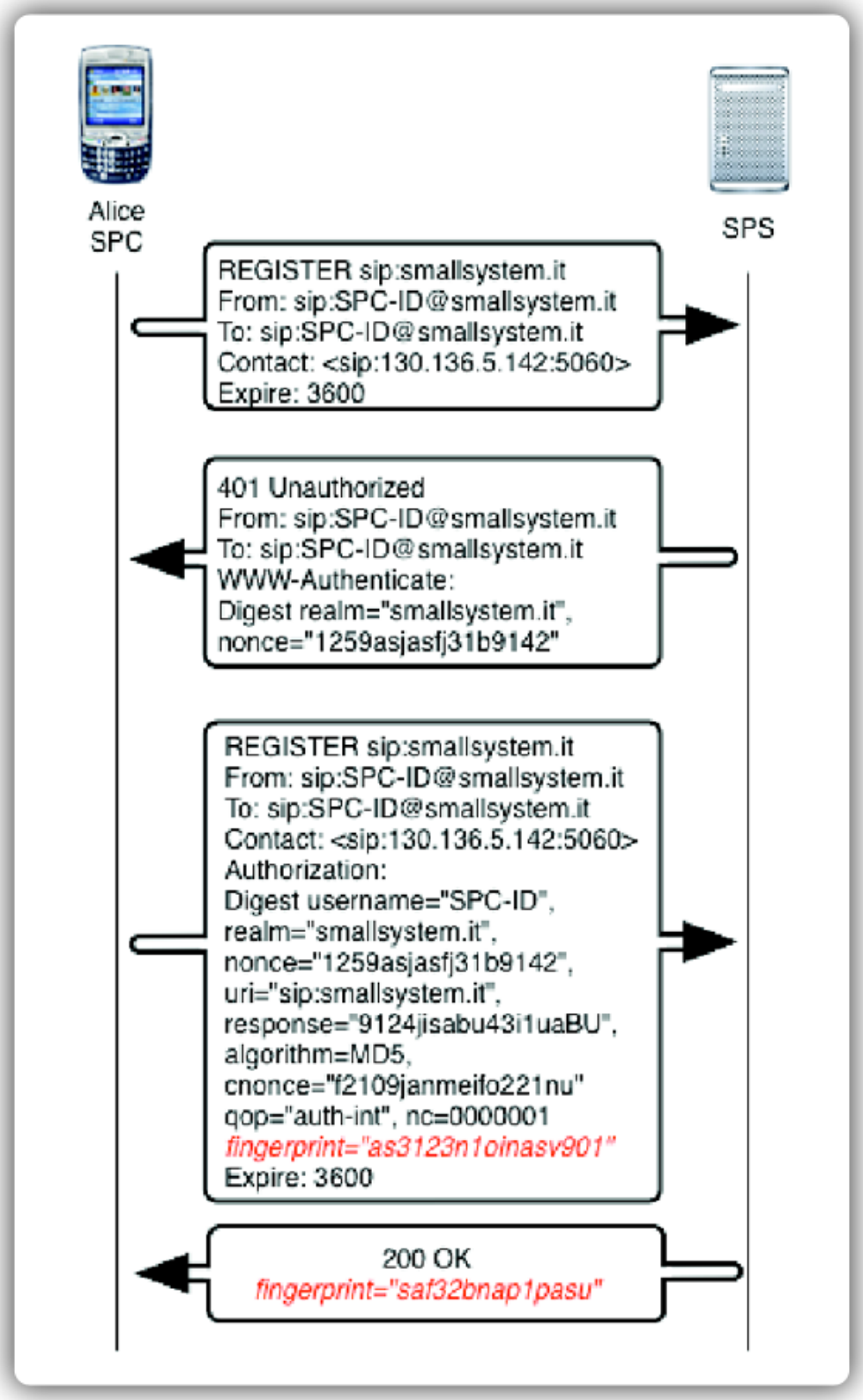

Figura 4.2: *Four-Way Handshake nell'autenticazione ABPS*

- calcola l'*hash* del messaggio inviato attraverso l'algoritmo *SHA-1* e lo salva in memoria. Verrà utilizzato nella *fase 4*:

$$hash\text{-}message\text{-}challenge\text{-}server = SHA-1\ (message\text{-}challenge).$$

3. **Fase 3 - Response**:

- il *client* calcola l'*hash* del messaggio ricevuto contenente il *challenge*, e lo salva per utilizzarlo in seguito:

$$hash\text{-}message\text{-}challenge\text{-}client = SHA-1\ (message\text{-}challenge);$$

- calcola il *response* in maniera analoga a *SIP/HTTP Digest*, utilizzando come *password* la *master-key* (chiave precondivisa con il *server*);
- aggiunge l'*header* di autenticazione, contenente il *response*, e costruisce il messaggio da inviare al *server*;
- calcola l'*hash* del messaggio generato, comprensivo di *response*:

$$hash\text{-}message\text{-}response\text{-}client = SHA-1\ (message\text{-}response);$$

- calcola una chiave temporanea, basata sulla *master-key* e su alcune credenziali del messaggio *response*:

$$masq\text{-}master\text{-}key =$$

$$SHA{-}1(\text{ “ 20 : ” } ||\ master\text{-}key\ ||\ \text{ “ : ” } ||\ cnonce\ ||\ \text{ “ : ” } ||\ nonce\ );$$

- calcola il *fingerprint* utilizzando la funzione crittografica *HMAC*, usando come funzione di *hash SHA-1*:

$$fingerprint =$$

$$HMAC - SHA-1\ (message\text{-}response,\ masq\text{-}master\text{-}key);$$

- aggiunge il campo *fingerprint* al messagio *response* generato in precedenza e lo invia al *server*.

4. **Fase 4 - Autenticazione Mutuale**:

   - il *server* rimuove dal messaggio *response* ricevuto il campo *fingerprint*;
   - calcola la chiave temporanea ricevuta usando la *master-key* e i valori *nonce* e *cnonce*, presenti nel messaggio ricevuto:

$$masq\text{-}master\text{-}key =$$

$$SHA{-}1(\text{ “ } 20 : \text{” } || \ master\text{-}key \ || \text{ “ : ” } || \ cnonce \ || \text{ “ : ” } || \ nonce \ );$$

   - calcola il *fingerprint* del messaggio ricevuto contenente il *response*:

$$fingerprint =$$

$$HMAC - SHA - 1 \ (message\text{-}response, \ masq\text{-}master\text{-}key);$$

   - confronta il *fingerprint* appena calcolato, con quello ricevuto in allegato al messaggio *response*. Se i valori non corrispondono crea un messaggio di errore (*codice 503 - Unauthorized*) e lo invia al *client* terminando il processo di autenticazione. Se la verifica è andata a buon fine, crea un messaggio di *acknowledgement*.
   - calcola la chiave di sessione, utile per l'autenticazione dei messaggi:

$$session\text{-}key\text{-}ia =$$

$$HMAC - SHA - 1(\ (hash\text{-}message\text{-}response\text{-}client \, || \text{ “ : ” } \ ||$$

$$hash\text{-}message\text{-}challenge\text{-}client \, || \, 0x00), \ masq\text{-}master\text{-}key);$$

   - calcola il *fingerprint* del messaggio *200 OK* costruito in precedenza, utilizzando la chiave di sessione appena costruita:

$$fingerprint =$$

$$HMAC - SHA - 1(message\text{-}200\text{-}OK \ ; \ session\text{-}key\text{-}ia);$$

   - aggiunge il *fingerprint* calcolato al messaggio *200 OK* salvato in precedenza e lo invia al *client*.

5. **Fase 5:**

   - il *client* rimuove il *fingerprint* dal messagio *200 OK* ricevuto;
   - calcola la chiave di sessione utilizzando i valori salvati in precedenza (*fase 3*):

$$\textit{session-key-ia} = HMAC-SHA-1(\ (\textit{hash-message-response-client} \,||\, \text{“ : ”} \ || \ \textit{hash-message-challenge-client} \,||\, \text{0x00}),\ \textit{masq-master-key});$$

   - calcola il *fingerprint* del messaggio ricevuto con la chiave di sessione appena calcolata:

$$\textit{fingerprint} = HMAC-SHA-1(\textit{message-200-OK}\ ;\ \textit{session-key-ia});$$

   - confronta il *fingerprint* calcolato con quello allegato al messaggio *200 OK*. Se la verifica va a buon fine, *client* e *server* sono mutualmente autenticati. Possono usare la chiave di sessione *session-key-ia* per autenticare i messaggi *SIP* che scambieranno in futuro. In caso contrario la procedura fallisce e il *client* può decidere di ritentare l'autenticazione.

Si noti che la descrizione dell'autenticazione *ABPS SIP/HTTP Digest* riguarda lo stato attuale dell'implementazione, al momento ancora in fase di sviluppo. In realtà il response non dovrebbe essere mai calcolato usando direttamente la chiave precondivisa tra client e server *ABPS*, ma piuttosto con una chiave derivata da questa, utilizzando una *Key Derivation Function*, in conformità alle direttive espresse in [15].

4.5.2 Sicurezza del traffico RTP

Premessa

L'architettura *ABPS*, come spiegato in precedenza, permette di realizzare un canale virtuale *multi-path* tra il *client* e il *proxy server*, dove quest'ultimo agisce sia da *proxy SIP* che *RTP*.

Si noti come questa impostazione, necessaria per garantire continuità alle comunicazioni del *client* mobile, rompa la tipica impostazione *"trapezoidale"* del protocollo *SIP*. In uno scenario classico, infatti, i *server SIP* vengono utilizzati solamente come punto di appoggio durante la fase di instaurazione della chiamata, dopodiché il flusso dati *RTP* viene instaurato direttamente tra i due *end-point* coinvolti nella comunicazione.

Il proxy server *ABPS*, posizionato al di fuori di ogni *firewall* e *NAT*, è un punto di passaggio obbligatorio anche per il traffico *RTP*, in quanto le classiche tecnologie per il superamento dei *NAT*, come *STUN* e *ICE*, fanno affidamento sulla coppia indirizzo *IP* e porta e non si adattano ad uno scenario in cui un *client* cambi punto di ancoraggio alla rete.

Va inoltre considerato il fatto che l'architettura *ABPS* è concepita per essere pienamente compatibile con il protocollo *SIP*, non per offrire un sistema di comunicazioni alternativo in cui tutti gli *UA* che intendono comunicare tra loro debbano essere costretti ad implementare il protocollo stesso. In altre parole, un *client ABPS* deve poter raggiungere, e allo stesso tempo essere raggiunto, anche un *end-point SIP* classico. Quest'ultimo è all'oscuro dell'esistenza del protocollo *ABPS* e comunica con il primo come se esso fosse il *proxy server ABPS*, cioè il suo punto di ancoraggio alla rete.

Questo scenario implica che il protocollo *ZRTP* non possa essere, almeno in una prima istanza, usato direttamente tra i due estremi di una comunicazione al fine di cifrare il traffico *end-to-end*.

Il flusso dati risulta quindi autenticato e cifrato nel percorso che va dal *client* al *proxy ABPS*. Una cifratura *end-to-end* del traffico audio non è, per il momento, presa in considerazione da questa architettura.

ZRTP in ABPS

Sulla base delle considerazione precedenti, lo scambio di chiavi viene effettuato usando il protocollo *ZRTP* configurato in modalità *Diffie-Hellman*.

Per garantire che la negoziazione delle chiavi avvenga in modalità autenticata, prevenendo in tale maniera un attacco di tipo *MiTM*, client e server *ABPS* possono sfruttare il segreto condiviso negoziato attraverso la fase precedente di autenticazione challenge-response (*session-key-ia*). Un *client*,

dopo essersi autenticato presso il server *ABPS*, per effettuare una chiamata esegue la seguente procedura:

1. Genera un *hash* del messaggio *ZRTP Hello* che verrà spedito in seguito;

2. Aggiunge all'*header SDP* il campo *hello-hash*, contenente l'*hash* calcolato in precedenza; inizia quindi la procedura di instaurazione della chiamata inviando un messaggio *INVITE* al destinatario attraverso il *proxy ABPS*. Il messaggio inviato è protetto da manomissioni grazie al *fingerprint* allegato, calcolato grazie alla chiave di sessione precedentemente negoziata durante l'autenticazione *challenge-response ABPS*.

3. Riceve il messaggio di risposta all'*INVITE* e memorizza il valore *hello-hash* presente nell'*header SDP*. Questo valore è l'*hash* del messaggio *ZRTP HELLO* che riceverà in risposta dal *proxy server*, dopo aver iniziato lo scambio di chiavi.

4. Avvia la normale procedura di negoziazione delle chiavi *ZRTP* in *Diffie-Hellman mode*. Si noti come ora lo scambio di chiavi può avvenire prevenendo un attacco di tipo *MiTM*: considerato che le due entità coinvolte nello scambio sono a conoscenza degli *hash* dei rispettivi messaggi *ZRTP HELLO*, la comunicazione risulta autenticata.

5. Sono ora disponibili tutti i parametri necessari ad instaurare il canale *SRTP*, che può quindi essere avviato per trasportare il traffico vocale.

In Figura 4.3 è rappresentata la fase inerente a *ZRTP*. Questo scenario parte dalla fase successiva allo scambio di messaggi di *INVITE* (contenenti l'*hash* dell'*HELLO ZRTP*): la sessione *SIP* è già terminata e viene instaurata quella *RTP/SRTP*.

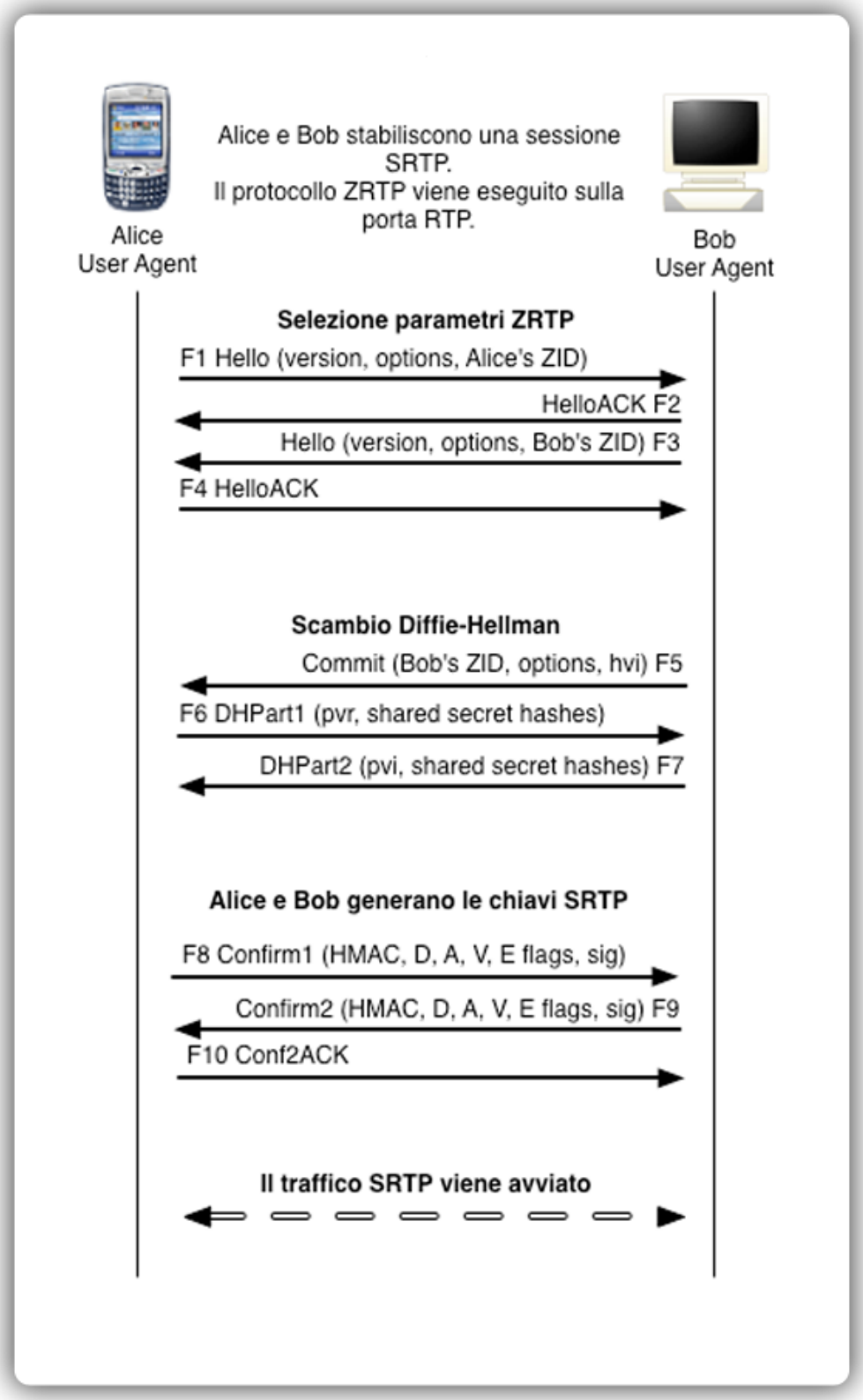

Figura 4.3: *Sessione ZRTP*

Capitolo 5

Symbian OS: il sistema operativo Nokia

Nel corso degli ultimi decenni i dispositivi mobili, *Personal Digital Assistant* (PDA) e *smarthphone*, hanno conquistato fette di mercato sempre più rilevanti.

Le funzionalità di cui questi dispositivi nel corso degli anni si sono arricchiti erano fino a poco tempo fa impensabili. Con lo sviluppo di hardware sempre più sofisticato è stato possibile fornire tali dispositivi di caratteristiche sempre più complesse. Con l'aumento delle potenzialità legate alle capacità di calcolo dei dispositivi mobili, i produttori hanno dovuto accrescere i propri sforzi nello sviluppo di sistemi operativi, in modo da presentare soluzioni software sempre più efficienti e complete.

Attualmente i sistemi operativi più usati per gli smartphone sono *Symbian OS* (*Symbian Foundation*), *Palm OS* (*PalmSource Inc.*), *Windows CE* (*Microsoft*), *Windows Mobile* (*Microsoft*), *BREW* (*Qualcomm*), *Linux*, *Android* e *iPhone OS*.

Fino al 2010 *Symbian OS* era il più diffuso, ma, dall'inizio del 2011, *Android*, il sistema operativo sviluppato da *Google*, lo ha superato, accaparrandosi una fetta pari al 33% del mercato degli *smartphone*, contro il 31% di *Symbian*, seguito da *iPhone OS* di casa *Apple* con il 16%.

Symbian OS, è uno dei sistemi operativi usati da *Nokia* per cellulari e *smartphone*, nato come figlio del sistema *EPOC* di *Psion*. La piattaforma

Symbian è stata creata grazie alla fusione e integrazione di diverse tecnologie, in particolare il *core* del sistema chiamato *S60* e parti delle interfacce utente *UIQ* e *MOAP(S)*. Il progetto è stato realizzato in particolare grazie al contributo di società come *Nokia*, *NTT DoCoMo*, *Sony Ericsson*, *Texas Instruments*, *Vodafone*,

Nel 2008, la società *Symbian Software Limited* fu acquistata da *Nokia* e successivamente trasformata in un'organizzazione *no-profit* chiamata *Symbian Foundation*.

Il software, inzialmente rilasciato con licenza *SFL* (*Symbian Foundation License*), a causa di problemi burocratici legati a parti del sistema sviluppate da terzi, è stato rilasciato completamente sotto licenza *EPL* (*Eclipse Public License*, una licenza *open-source*) solo all'inizio del 2010.

5.1 Caratteristiche tecniche

Symbian è un sistema operativo disegnato per fronteggiare le problematiche relative alle caratteristiche hardware *embedded* degli *smartphone*, molto diverse per alcuni aspetti da quelle dei calcolatori, in particolare in termini di consumo di potenza e quantità di memoria disponibile, oltre ad un pieno supporto per i requisiti specifici del trasporto dati.

Le caratteristiche fondamentali di *Symbian* [5] sono:

- ***Performance***: progettato per massimizzare la durata della batteria attraverso un *power management* specifico, in base alle caratteristiche del dispositivo;
- ***Multitasking***: telefonia, messaggistica e comunicazioni sono componenti fondamentali. Tutte le applicazioni sono progettate per funzionare in parallelo.
- ***Standard***: l'uso di tecnologie basate sugli *standard* di settore è un principio fondamentale di *Symbian OS*, ciò assicura l'interoperabilità delle applicazioni sviluppate.
- **Software *object-oriented***: *Symbian OS* nasce con una struttura *object-oriented* ed ha un'architettura altamente modulare;

- **Gestione ottimizzata della memoria**: gli eseguibili per questo sistema operativo hanno una dimensione decisamente ridotta e i requisiti di *runtime* sono minimizzati;

- **Meccanismi di sicurezza**: sono presenti funzionalità al fine di consentire comunicazioni sicure e l'archiviazione sicura dei dati;

- **Internazionalizzazione**: il sistema ha incorporata la gestione del set di caratteri *Unicode* per garantire un semplice sviluppo delle funzionalità di localizzazione.

Tutte le versioni rilasciate fino ad ora di *Symbian OS* supportano solo processori *ARM*. È noto il porting di tale sistema per *CPU Atom* di *Intel*, ma per motivi legati a strategie commerciali, questa versione non è stata rilasciata.

Il kernel attualmento in uso è conosciuto sotto il nome di *EKA2* (*EPOC Kernel Architecture 2*), le cui caratteristiche principali sono:

- Architettura *Microkernel*;

- *Preemptive multitasking*;

- Protezione della memoria;

- Supporto per i *thread* sia in *Kernel-space* che in *User-space*;

- Garanzie per un supporto *real-time*;

- *Pluggable memory models*.

5.2 Licenza e Sviluppo

Come già detto, con la nascita della *Symbian Foundation*, ed il conseguente rilascio di *Symban* con licenza *open-source*, si va incontro ad un radicale cambiamento, che ha permesso una sempre più fiorente crescita di una community di sviluppatori in tutto il mondo.

Inoltre il passaggio da sistema proprietario a sistema completamente gratuito ha portato un cambiamento per quanto riguarda l'ambiente di sviluppo *IDE* (*Integrated Development Environment*), passando da un ambiente *CodeWarrior* a *Carbide.c++*, che tuttora detiene il primato come primo ambiente di sviluppo per applicazioni su Symbian.

Symbian OS è utilizzato in molteplici smartphone, ad esempio i *Nokia Series 60*, *Series 80* e *Series 90*, e *UIQ* (la piattaforma di *UIQ Technology*).

Ogni piattaforma ha bisogno di un *SDK* (*Software Development Kit*) specifico, ed attualmente *Symbian Foundation* non ha rilasciato una sua versione dell'*SDK*. Per il momento quindi non è ancora disponibile una versione *open-source* dell'*SDK*, e per lo sviluppo bisogna appoggiarsi all'*SDK* realizzato da *Nokia*.

Quest'ultimo contiene le librerie e gli *"header" files* necessari per sviluppare applicazioni per *Symbian OS*, ed un emulatore che consente di effettuare dei test utilizzando un normale pc. L'*SDK* della nokia deve essere usato in concomitanza con l'*Application Development Toolkit* (*ADT*). L'*SDK* consente l'accesso alle *API* pubbliche, e, lavorando esclusivamente su quest'ultime, si ha la sicurezza che le applicazioni sviluppate funzioneranno su un ampio ventaglio di device disponibili attualmente o che verranno realizzati in futuro.

Il sistema operativo *Symbian* supporta lo sviluppo di applicazioni in *Java* e in liguaggio *C++*, a cui sono state aggiunte particolari estensioni proprie di *Symbian*.

Capitolo 6

Progettazione e Sviluppo

6.1 Introduzione

Il lavoro implementativo ha riguardato il *client Symbian*, tramite la libreria *PJSIP* con cui esso è sviluppato.

Questa libreria, come verrà descritto successivamente, ha una composizione modulare che permette estensioni e personalizzazioni della stessa, nel rispetto della propria struttura. L'architettura *ABPS* tuttavia, non ha una struttura completamente definita, in quanto rappresenta ancora un sistema in fase di studio, sia per quanto riguarda la valutazione degli aspetti prestazionali, che per quanto concerne alcune soluzioni pratiche e dettagli implementativi legati al contesto di utilizzo.

Nello sviluppo del software si è cercato di tenere in considerazione questi due aspetti, privilegiando, laddove ce ne fosse bisogno, soluzioni che portassero ad una rapida e funzionante implementazione, piuttosto che un'estensione organica della libreria stessa.

Si noti che al momento della stesura di questo elaborato, *PJSIP* non offre un supporto diretto al protocollo *ZRTP*. La fase iniziale di progettazione, infatti, ha richiesto la modifica dell'integrazione di *LIBZRTP* (sviluppata in precedenza da altri tesisti, ma affetta da numerosi errori riscontrati durante l'implementazione) dentro a *PJSIP*.

Prima di descrivere tale fase di integrazione, verranno elencati gli strumenti utilizzati durante la fase di sviluppo, ed in seguito sarà descritto bre-

vemente lo stato delle applicazioni al momento dell'inizio del lavoro implementativo.

6.2 Librerie e Strumenti di Sviluppo

6.2.1 PJSIP

PJSIP è una libreria *open-source* che implementa uno stack *SIP* e uno *stack multimediale* di supporto al *VoIP*, *instant messaging* e comunicazioni multimediali.

Le sue caratteristiche principali sono:

- ***open-source***: il codice è rilasciato sotto licenza *GPL 2.0*;
- **elevate prestazioni**: è in grado di processare centinaia di chiamate al secondo, sfruttando una macchina *Desktop* con processore *Intel P4/2.4GHz*. Ci si possono aspettare prestazioni superiori utilizzando un *server* con hardware adatto e un processore più potente.
- **dimensioni contenute/scalabilità**: è in grado di scalare verso il basso per essere utilizzata con device a basse prestazioni e ridotte risorse di memoria, così come sfruttare le potenzialità offerte da *server* multiprocessore. Il tutto usando lo stesso stack *SIP*.
- **portabilità**: funziona su architetture a *32 bit*, *64bit*, *big endian* e *little endian*. Esso è portabile su praticamente ogni sistema operativo esistente.
- **documentazione esaustiva**: è dotata di una documentazione esaustiva, sia generata a partire dai sorgenti, che corredata di articoli scritti a mano con spiegazioni articolate sull'architettura.

In realtà utilizzare la terminologia *"Librerie PJSIP"* non è del tutto corretto, in quanto *PJSIP* rappresenta appunto solo una delle librerie connesse (Figura 6.1) componenti uno stack di librerie.

In particolare l'insieme è composto da:

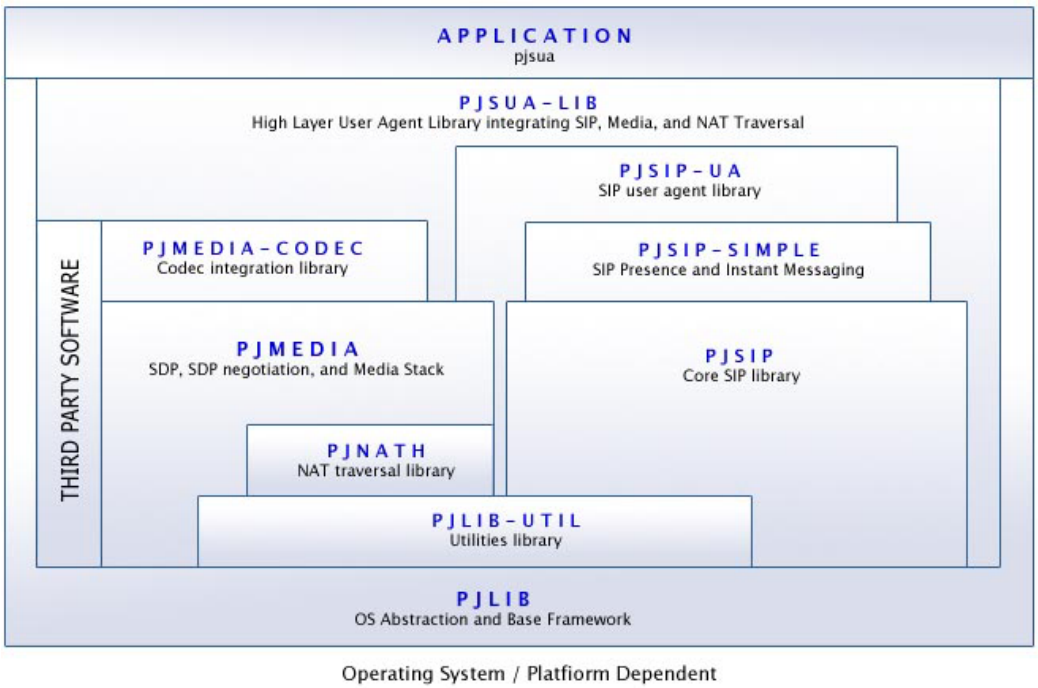

Figura 6.1: *Stack delle librerie PJSIP*

- ***PJSIP***: rappresenta uno stack *SIP* che supporta un insieme di features ed estensioni del protocollo *SIP*;

- ***PJLIB***: rappresentante la libreria a cui le restanti si appoggiano. Si occupa di garantire funzionalità di base (es. l'astrazione rispetto al sistema operativo sottostante). Può essere considerata una replica della libreria *libc* con l'aggiunta di alcune features come la gestione dei *socket*, funzionalità di *logging*, gestione *thread*, mutua esclusione, semafori, *critical section*, funzioni di *timing*, gestione eccezioni e definizione di strutture dati di base (liste, stringhe, tabelle, ecc...).

- ***PJLIB-UTIL***: anch'essa come *PJLIB* è una libreria di appoggio, ma a differenza della prima implementa funzioni più complesse utili per la crittografia, come gli algoritmi *SHA1*, *MD5*, *HMAC*, *CRC32*, ed anche funzioni per il *parsing* e la manipolazione di testi;

- ***PJMEDIA***: libreria il cui scopo è la gestione ed il trasferimento dei dati multimediali;

- ***PJSUA***: rappresenta il livello più alto fungendo da *wrapper* verso le altre librerie. Inoltre agevola notevolmente la scrittura di applicazioni.

Caratteristiche tecniche

Come precedentemente detto *PJSIP* è un vero e proprio composto di librerie.

In questo paragrafo ci si concentrerà sull'analisi dei metodi di collaborazione con le altre librerie e della struttura di *PJSIP*, in quanto rappresenta il fulcro dell'insieme delle librerie.

Inoltre in essa sono state apportate le principali modifiche attue a implementare il sistema *ABPS* su *Symbian OS*.

Le principali componenti di PJSIP sono:

- ***Endpoint***: rappresenta il cuore dello *stack*, ed ha principalmente il ruolo di gestire la *"pool factory"*, allocando memoria per tutti i componenti *SIP*. Inoltre agendo da *scheduler* per i vari componenti *SIP*, gestisce la varie istanze del *Transport Manager* (che si occupa dell'instradamento dei messaggi), detiene una singola istanza della *I/O queue*, gestisce i moduli *PJSIP* ed infine riceve i messaggi entranti dal *Transport Manager*, occupandosi anche della redistribuzione ai moduli soprastanti.

- ***Transport***: i *Transport*, come suggerisce il nome, hanno il principale scopo di invio e ricezione dei messaggi provenienti dalla rete. In generale è possibile dire che il livello di *transport* ha come fulcro il *Transport Manager*, il cui scopo è gestire la creazione di tutti i *transport* necessari, oltre ad offrire vari servizi, quali l'instradamento dei pacchetti provenienti dai vari *transport* per poi passarli all'*endpoint*, la ricerca del *transport* adeguato per l'invio dei messaggi in base al tipo di messaggio da spedire e all'indirizzo remoto, la gestione delle *factories* dei trasporti e la diretta gestione della vita dei *transport*, basandosi su un sistema di conteggio delle *reference* ed un *timer* di inattività.

La libreria *PJSIP* alloca un solo *Transport Manager* per *endpoint*, che normalmente non risulta visibile alle applicazioni che devono utilizzare le funzioni esposte dall'*endpoint*.

- ***Transaction***: il compito di questo *"livello"* è quello di attuare le giuste fasi di transazione in base ai messaggi *SIP* in entrata e/o uscita: possibile ritrasmissione messaggi di *INVITE* o *REGISTER* in mancanza di una risposta dei primi, risposte in base ai messaggi ricevuti dal *server* (come un *200 OK* o un *503 Service Unavailable*). Esso possiede inoltre delle *API* necessarie a comunicare all'*endpoint* i messaggi in uscita, e delle funzioni di *callback* necessarie al monitoraggio delle trasmissioni.

- ***UA***: il concetto astratto dato dallo *User Agent* (in particolare *UA* introduce sia la classe *user_ agent* generica sia la classe *dialog*) è quello della creazione, distruzione ed identificazione delle sessioni (*INVITE*, *REGISTER*, *SUBSCRIBE/NOTIFY*, ecc...) necessarie ad implementare correttamente le fasi di una comunicazione *SIP*.

- ***DataBuffer***: ogni messaggio ricevuto viene passato attraverso i vari componenti software, ed incapsulato in una struttura che contiene informazioni aggiuntive, come ad esempio l'istante temporale di ricezione, o l'indirizzo *IP* del mittente del messaggio stesso. La dichiarazione dei *buffer* è presente nel file *pjsip/sip_transport.h*: in tale file viene descritto anche il *transmit data buffer* che è il *buffer* che viene usato per l'invio dei messaggi.

- ***I/O Queue***: anche se non direttamente appartenente alla struttura di *PJLIB* (*I/O Queue* è un modulo appartenente a *PJLIB*) è importante citare questo insieme di metodi e funzioni in quanto esse comunicano direttamente con lo strato *Transport*. *I/O Queue* ha il principale compito di fornire un insieme di *API* per la gestione delle principali operazioni *asincrone* di *Input/Output*. In particolare essa può lavorare sia su *socket* sia su *descrittori di files*, lavorando in maniera nativa in sistemi che, come *Symbian OS*, gestiscono correttamente le operazioni asincrone. In casi differenti, invece, si limita ad effettuare un *polling* per simularne il funzionamento.

- ***Callback***: *PJSIP* prevede un meccanismo di *callback* che ha origine nel *Transport Manager*, il quale effettua il *parsing* del messaggio, che transita dell'*endpoint* e poi viene propagato da esso verso le altri parti del sistema.

6.2.2 LIBZRTP

LIBZRTP [8] è la libreria che implementa le funzioni di *key-agreement ZRTP*, sviluppata dall'autore del protocollo stesso, Philip Zimmermann (anche creatore di *PGP*, acronimo di *Pretty Good Privacy*, il più usato software di cifratura per e-mail), e utilizzata all'interno dello *Zfone Project*.

Scritta in linguaggio *C*, è compatibile con i seguenti sistemi operativi:

- *Linux*;
- *Windows*;
- *Symbian OS*.

Per semplicità di comprensione ed utilizzo è possibile suddividere a grandi linee la complessa organizzazione delle librerie *LIBZRTP* in vari "sottogruppi" a seconda del ruolo ricoperto.

1. **Strutture e funzioni di configurazione**
   Un insieme di strutture dati e funzioni il cui scopo è quello di costruire un profilo iniziale su cui l'intero protocollo *ZRTP* si appoggerà. A capo di tutto ci sono due fondamentali strutture *zrtp_config_t* e *zrtp_profile_t*.

   La prima racchiude un insieme di dati fondamentali, tra cui i più importanti sono un *id* simbolico da associare al *client* in uso, ed un puntatore a funzioni di *callback*. Tali *callback* verranno richiamate al momento del bisogno, in particolare hanno il compito di monitorare eventuali eventi (incluse eccezioni) causati da un cambio di stato o configurazione, e di gestire il traffico dei pacchetti in arrivo ed in uscita. È pertanto necessario in fase di configurazione associare le giuste funzioni di *callback* in maniera tale da permettere l'applicazione del protocollo *ZRTP* all'interno della sessione *RTP*.

zrtp_profile_t è invece una struttura che rappresenta un profilo da associare ad una specifica sessione *ZRTP* (per ogni sessione *ZRTP* può essere associato uno ed un solo specifico profilo). Le componenti principali del profilo sono una serie di preferenze per il settaggio dei parametri di crittografia (calcolo *SAS*, *Hash*, *Public Key*, ecc...). La configurazione di un profilo può avvenire tramite un settaggio specifico determinato dal programmatore oppure è possibile lasciare al sistema il compito di settare i parametri con i giusti criteri.

2. **Strutture e funzioni di inizializzazione**
   zrtp_config_defaults, *zrtp_init* e *zrtp_down* sono le principali funzioni responsabili dell'inizializzazione delle librerie *ZRTP*.
   zrtp_config_defaults ha il compito, in caso in cui il profilo non sia stato configurato dal programmatore, di settare i parametri adeguati di un profilo (ovviamente la funzione necessita di una chiamata esplicita).
   zrtp_init rappresenta il *main* delle librerie *LIBZRTP*, in quanto inizializza tutti i suoi componenti, i dati globali, compresa la gestione della memoria.
   zrtp_down ha il compito di deallocare tutte le risorse utilizzate da *LIBZRTP*.

3. **Creazione e configurazione di una sessione *ZRTP***
   Prima di tutto è necessario chiamare alcune funzioni il cui compito è di creare ed allocare memoria per la sessione *ZRTP* in oggetto. *zrtp_session_t* è la struttura rappresentante la sessione da inizializzare (compito affidato alla funzione *zrtp_session_init*), mentre *zrtp_stream_t* rappresenta lo *stream* su cui avverrà la sessione *ZRTP*. In particolare *zrtp_stream_t* viene associato nel momento della sua configurazione alla sessione corrente in uso.

4. **Attaching di uno stream *ZRTP* e protocollo di inizializzazione**
   Sono presenti una serie di funzioni il cui compito è quello di eseguire un *attachment* (associazione) tra sessione e *stream*, in modo tale da garantire che tutti i parametri settati sul profilo, sullo *stream* e sulla sessione operino nella stessa fase.

5. **Traffico *RTP*, *SRTP* e *RTCP***
   LIBZRTP di per sè non implementa il trasporto dei pacchetti, lasciando tale compito ad eventuali librerie a cui fa supporto (nel nostro caso *PJLIB*). Per questo le funzioni appartenenti a questo "gruppo" hanno il solo compito di processare i pacchetti in entrata ed in uscita dal canale *RTP* assicurando che i parametri crittografici vengano rispettati durante la trasmissione.

6.2.3 Tools

Come strumenti di supporto allo sviluppo e al *debug*, in aggiunta all'ambiente *Carbide.c++* e all'*SDK*, si sono utilizzati alcuni tool, tra i quali **Wireshark**, software *open-source* per lo *sniffing*, utilizzato per l'analisi e il debug dei protocolli di rete utilizzati, e **TShark**, la versione a riga di comando di *Wireshark*, utile per le prove da remoto.

6.3 Il Software di partenza e problematiche

Il lavoro di sviluppo è iniziato avendo a disposizione due programmi:

- il *Client VoIP* sviluppato per piattaforme *Symbian*;
- il *Server Proxy ABPS* sviluppato per piattaforme *Linux*.

Al momento dell'inizio della fase di progettazione, il *Client VoIP* (che chiameremo da ora in poi ***Symbian_ UA***), riusciva correttamente a registrarsi in primo luogo presso il *Server ABPS*, tramite un'estensione dell'autenticazione *SIP/HTTP Digest*, già affrontata in precedenza, ed, in secondo luogo, presso il *SIP Provider Registrar Server* (nel nostro caso *Ekiga*).

Il *Server Proxy ABPS* godeva di una stabilità sicuramente maggiore, dovuta al fatto che numerosi altri test sono stati condotti in precedenza, usando client *VoIP/ABPS* sviluppati per altri sistemi operativi (es. *Linux*, *Android*) e perfettamente funzionanti.

Il client *VoIP* soffriva però di alcuni problemi riguardanti le fasi successive alla sessione *SIP* di *REGISTER*. In particolare è stato un riscontrato un errore relativo alla fase di *INVITE*.

Il problema risiedeva nel fatto che, al momento della costruzione del messaggio *INVITE*, nell'*header "FROM"*, veniva erroneamente inserito l'indirizzo *IP* locale della macchina su cui veniva eseguito il *Symbian_ UA*. Con questa situazione, il *Server Proxy*, nel momento in cui doveva inoltrare un messaggio di risposta (es. *200 OK*) proveniente dal ***Corrispondent Node*** (***CN***) verso il *Symbian_ UA*, mandava tale messaggio all'indirizzo presente nell'*header*, che ovviamente non veniva ricevuto dal *client*, in quanto tutto il traffico *SIP* deve passare obbligatoriamente tramite il *Server* di registrazione.

Per risolvere questo problema è bastato sostituire l'indirizzo *IP* locale con l'indirizzo di registrazione al *SIP Registrar Server* esterno (un account della tipologia *username@sip_provider.net*, nel nostro caso *username@ekiga.net*).

In questo modo, la fase di *INVITE* viene completata con successo, facendo sì che i messaggi provenienti dal *CN*, passino attraverso il *Server ABPS* e vengano inoltrati al *Symbian_ UA*.

La prima parte dello sviluppo ha quindi riguardato la correzione dei bug riscontrati nelle fasi precedenti.

Nelle sezioni successive saranno affrontate le fasi riguardanti l'integrazione della libreria *LIBZRTP* dentro a *PJSIP*, e la realizzazione vera e propria dell'omonimo protocollo, tramite le chiamate alla libreria.

6.4 Integrazione di LIBZRTP all'interno di PJSIP

Per quanto riguarda la libreria *LIBZRTP*, essa era già stata parzialmente integrata nelle librerie *PJSIP*, come libreria esterna, ma, in seguito a numerose problematiche successivamente affrontate, si è scelto per una modifica quasi totale di tale integrazione. Era già stata inoltre effettuata la fase relativa al calcolo dell'*hash* del messaggio *HELLO* e del suo conseguente inserimento nel campo *SDP* dell'*INVITE*, ma anche questa fase è risultata errata.

In primis, quindi, si è optato per creare un *"header file"* (*struct_ zrtp.h*), in maniera tale che, tutte le strutture *ZRTP* e le variabili di supporto dichiarate al suo interno, potessero essere richiamate da tutti i file che volessero

accedervi. Non è stato banale giungere ad una soluzione corretta e funzionante, in quanto i file che dovevano manipolare le strutture di *ZRTP* erano situati a diversi livelli nello stack delle librerie di *PJSIP* (in particolare *pjsua_media.c* facente parte della libreria PJSIP, e *stream.c* e *sip_inv.c* della libreria *PJMEDIA*.

Si è optato poi per spostare la fase di inizializzazione della libreria *ZRTP*. Inizialmente questa fase era posta nel file *ua.cpp*, dopo che avveniva con successo la doppia registrazione e prima di iniziare la fase di *INVITE*.

Tutta la procedura di inizializzazione è stata spostata nel file *pjsua_media.c* della libreria *PJSIP*. Il motivo di questa scelta risiede nel fatto che, una funzione di *callback*, chiamata *zrtp_on_send_packet*, e richiamata ogni qual volta si debba effettuare l'invio di un pacchetto *ZRTP*, dev'essere in grado di poter accedere alle strutture relative ai *transport* (in particolare alla struttura *pjmedia_transport*), non accedibili in alcun altro modulo. Con questa soluzione si ha che l'inizializzazione di *LIBZRTP*, spostata nella funzione *pjsua_media_subsys_start*, viene effettuata allo *startup* dell'applicazione, dopo che vengono creati i *socket* per le comunicazioni ma ancora prima che vengano iniziate le fasi di registrazione.

La chiamata alla funzione *zrtp_stream_start*, il cui compito è quello di iniziare lo stream *ZRTP* vero e proprio, tramite l'invio di un pacchetto di *HELLO*, è stata spostata anch'essa all'interno del file *pjsua_media.c* ma nella funzione *pjsua_media_channel_update*, la quale è chiamata dopo aver ricevuto il messaggio *200 OK* di risposta all'*INVITE* e dopo aver effettuato la negoziazione dei parametri contenuti nel campo *SDP* di tale messaggio.

Per quanto riguarda la fase di inserimento dell'*hello-hash* nel campo *SDP* dell'*INVITE*, il problema riguardava il fatto che questo attributo veniva inserito come "Session-Attribute", mentre, per risultare corretto, doveva essere inserito come "Media-Attribute", in quanto esso caratterizza univocamente il *media-stream* che si sta cercando di instaurare. Dopo questa ulteriore correzione, il *Server Proxy*, che riceveva il messaggio di *INVITE*, proveniente dal *Symbian_UA* e da inoltrare verso il *CN*, vedeva l'attributo di tale messaggio in modo corretto, e salvava tale attributo, in modo da riutilizzarlo per il confronto nel momento in cui riceveva il messaggio *ZRTP HELLO* dal *Symbian_UA*, per prevenire eventuali tentativi di *MiTM attack*.

Nel file *stream.c*, facente parte della libreria *PJMEDIA*, sono state poi aggiunte delle funzioni per processare i pacchetti *ZRTP* in entrata. In assenza di queste funzioni, i pacchetti *ZRTP* in ingresso erano processati come normali pacchetti *RTP*, e veniva infatti generato un errore "*RTP decode error*", dato dal fatto che la libreria falliva nel fare il *parsing* dell'header RTP, in quanto essendo un pacchetto *ZRTP*, come già detto in precedenza, possedeva un header leggermente diverso.

La funzione aggiunta è chiamata *zrtp_process_srtp*, ed è stata posta nella funzione *on_rx_rtp*, che è un *callback* richiamato ogni volta che viene ricevuto un messaggio *RTP*.

Inoltre è stata apportata una banale modifica, non utile alla fase di sviluppo, ma a quella di *debug*: tutte le stampe presenti nella libreria, venivano effettuate tramite la chiamata alla funzione *ZRTP_LOG*, la quale, per motivi ancora sconosciuti, metteva in output caratteri incomprensibili, totalmente inutili a scopi di *debug*. Tale funzione è stata, ove necessario, sostituita con la funzione *PJ_LOG*, funzione analoga implementata dalla libreria *PJSIP*, ma perfettamente funzionante.

6.5 Sessione ZRTP

In seguito alle modifiche descritte nel paragrafo precedente, si è potuti finalmente giungere alla vera e propria instaurazione di una sessione *ZRTP*, utile per effettuare lo scambio di chiavi tra *Symbian_UA* e *Proxy Server*, e per negoziare i parametri di sicurezza per stabilire poi una sessione *SRTP*.

Prima di descriver la sessione *ZRTP* si fa notare che, durante la fase di *INVITE*, si è riusciti a salvare correttamente l'*hello-hash* proveniente dal *Proxy Server*, tramite la chiamata alla funzione *zrtp_signaling_hash_set*, posta nel file *sdp_neg.c* di *PJMEDIA*. Al momento in cui viene ricevuto il messaggio *HELLO* vero e proprio dal *CN*, tuttavia, il confronto con l'*hello-hash* salvato, atto a scongiurare tentativi di *MiTM attack*, genera un errore a cui ancora non si ètrovata soluzione. Tale problema sarà spiegato in maniera più dettagiata successivamente.

La situazione attuale è descritta di seguito:

1. il *Symbian_ UA* calcola l'*hello-hash* e lo inserisce nell'*SDP* del messaggio *INVITE*, che manda poi al *Proxy Server*;

2. il *Proxy Server* riceve l'*INVITE*, si salva l'*hello-hash* ed inoltra il messaggio al *CN*;

3. il *CN* risponde con un messaggio *200 OK*, il quale viene ricevuto dal *Proxy Server*, che, dopo aver calcolato anch'esso l'*hello-hash* del proprio messaggio di *HELLO*, e averlo inserito nel messaggio, lo inoltra al *Symbian_ UA*;

4. il *Symbian_ UA* riceve il *200 OK*. In questo momento viene correttamente salvato l'*hello-hash* contenuto nel messaggio SDP.

5. la procedura *ZRTP* ha inizio: il *Symbian_ UA* manda il primo messaggio *HELLO* al *Proxy Server*.

6. il *Proxy Server* riceve l'*HELLO* e lo confronta con l'*hello-hash* salvato in precedenza. Il confronto restituisce un esito positivo e il *Proxy Server* risponde con un messaggio di *HELLO ACK*.

7. a sua volta il *Proxy Server* manda l'*HELLO*;

8. in questo momento succede una situazione inaspettata: il *Symbian_ UA* teoricamente dovrebbe confrontare l'*hash* del messaggio appena ricevuto con l'*hello-hash* che si sarebbe dovuto salvare in precedenza, ma siccome, come già detto, il confronto non ha esito positivo, il protocollo dovrebbe interrompersi, senza che esso mandi l'*HELLO ACK*. Contrariamente a ciò, il *Symbian_ UA* vedendo ricevere l'*HELLO* contenente i parametri di sicurezza del *Proxy Server*, effettua la lettura di tali parametri selezionando l'algoritmo più sicuro ed altri parametri, ed iniziando comunque la vera e propria negoziazione mandando un messaggio di *COMMIT*, contenente i parametri selezionati.

9. il *Proxy Server* riceve il *COMMIT* ma, visto che è ancora in attesa di ricevere l'*HELLO ACK* dal *Symbian_ UA* scarta il messaggio appena ricevuto in quanto non è ancora pronto per processarlo (in particolare *zrtp_ state_ t* assume il valore *ZRTP_ STATE_ WAIT_ HELLOACK*),

e generando un errore che fa sì che tutta la transazione *ZRTP* venga distrutta.

Concludendo, tutti gli aspetti relativa alla sessione *ZRTP* sono funzionanti, se non per il confronto dell'*hello-hash* con il messaggio *HELLO*. Risolto anche questo problema, infatti, la sessione sarà funzionare da entrami le parti, in quanto, tutti i cambi di stato saranno gestiti correttamente.

Conclusioni

Le tecnologie *VoIP*, i dispositivi mobili, il *Multihoming* e le sue problematiche fin qua spiegate sono un insieme di tecnologie in continua evoluzione, oramai facenti parte di un servizio a disposizione di tutti. Il far sì che queste innovazioni possano interagire con minor problemi possibili e garantire le migliori prestazioni possibili, è quindi un obiettivo molto importante da raggiungere per tecnologia informatica applicata.

Il sistema *ABPS*, implementato tramite l'utilizzo di un framework *PJSIP* ed un protocollo di *key-agreement* garantito da un libreria come *LIBZRTP*, è un ottima soluzione alle problematiche di *seamless mobility* e *Multihoming*, in grado di unire fortemente concetti teorici a concetti pratici. È per questo motivo, infatti, che si necessita di un forte e continuo sviluppo di tale sistema, che in futuro, sarà anche in grado di essere esteso su piattaforme differenti da quella utilizzata.

Entrando nello specifico, sull'implementazione del sistema *ABPS* si è giunti ad una fase in cui la parte di autenticazione risulta completa e funzionante, e la sessione *ZRTP*, utilizzata per lo scambio di chiavi viene inizializzata correttamente. Rimane tuttavia ancora aperta la questione legata al completamento di tale sessione, in particolare al problema del confronto tra l'*hello-hash* nel campo SDP del *200 OK* e del messaggio *HELLO*, e del controllo che lo scambio di chiavi venga effettuato corretamente.

Dopo aver completato la fase relativa a *ZRTP* sarà possibilie, infatti, instaurare un canale sicuro *SRTP* grazie ai parametri negoziati, il quale permetterà la cifratura completa del traffico *RTP*.

Ringraziamenti

Desidero, per prima cosa, ringraziare Vittorio Ghini, relatore di questa tesi, per la sua infinita disponibilità e per l'aiuto che mi ha offerto in questi mesi.

Ringrazio inoltre tutta la mia famiglia, per il supporto morale ed economico, grazie alla quale mi è stato permesso di raggiungere questo traguardo.

Desidero infine ringraziare la mia ragazza Rosa che, anche questa volta, mi ha aiutato a revisionare l'intera tesi per renderla presentabile e mi ha supportato in questi lunghi mesi.

Bibliografia

[1] "Advanced Encryption standard (AES).", National Institute of Standard and Technology, FIPS Pub 197, November 2001. *2001*

[2] "IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications.", *2007*

[3] "New directions in cryptography.", IEEE Transactions on Information Theory 22, 1976. *1976*

[4] "One-way transmission time.", ITU-T Recommendation G.114, May 2003. *2003*

[5] "Symbian.org.", Developer Wiki.

[6] "Voip on the verge.", Telecommuncations Online, 2004. *2004*

[7] "wi-fi.org.", Wi-Fi Alliance. *2007*

[8] "zfone.com.", Libzrtp Detail Description API.

[9] **3GPP**, "Voice Call Continuity (VCC) between Circuit Switched (CS) and IP Multimedia Subsystem (IMS)", *2007*

[10] **J. Arkko, E. Carrara, F. Lindholm, M. Naslund, K. Norrman.** "MIKEY: Multimedia Internet KEYing.", RFC 3830, Internet Engineering Task Force, August 2004. *2004*

[11] **M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman.** "The Secure Real-time Transport Protocol (SRTP).", RFC 3711, Internet Engineering Task Force, March 2004. *2004*

[12] **M. Bishop.** "Introduction to computer security." *2008*

[13] **A. Bosselaers, B.D. Boer.** " Collisions for the compression function of MD5.", Advances in Cryptology, Proceedings of EUROCRYPT. *1994*

[14] **R. Canetti, M. Bellare, H. Krawczyk.** "Keyed hash functions and message authentication.", Proceedings of Crypto'96, LNCS 1109. *1996*

[15] **L. Chen** "Recommendation for key derivation using pseudorandom functions.", NIST Special Publication 800-108, Nov. 2004. *2004*

[16] **J. Davidson, J. Peters, J. Peters, B. Gracely.** "H.323.", Voice over IP fundamentals. Cisco Press. pp. 229–230. ISBN 9781578701681. *1996*

[17] **H. Dobbertin.** "Cryptanalysis of MD5 compress.", Rump Session of Eurocrypt. *1996*

[18] **C. Demichelis, P. Chimento.** "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM).", RFC 3393, Internet Engineering Task Force, November 2002. *2002*

[19] **D. Eastlake** 3^{rd}**, Motorola, P. Jones, Cisco Systems.** "US Secure Hash Algorithm 1 (SHA1).", RCF 3174. *2001*

[20] **C.Ellison, B.Schneier.** "Ten risks of pki: What youre not being told about public key infrastructure.", Computer Security Journal Volume XVI, Number 1, 2000. *2000*

[21] **R. T. Fielding, J. Gettys, J. C. Mogul, H. K. Nielsen, L. Masinter, P. J. Leach, T. Berners-Lee.** "Hypertext Transfer Protocol - HTTP/1.1.", RFC 2616, Internet Engineering Task Force, June 1999. *1999*

[22] **J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart.** "HTTP Authentication: Ba-

sic and Digest Access Authentication.", RFC 2617, Internet Engineering Task Force, June 1999. *1999*

[23] **V. Ghini, L.E. Tomaselli, G. Lodi, F. Panzieri, A. Messina**, "Always Best Packet Switching for SIP-based mobile multimedia services", Dept. of Computer Science, University of Bologna, Italy. *2009*

[24] **E. Gustafsson et al.**, *Always Best Connected*, IEEE Comm. Mag., vol. 10, no. 1, Feb. 2003, pp. 49–55. *2003*

[25] **M. Handley, V. Jacobson.** "SDP: Session Description Protocol.", RFC 2327, Internet Engineering Task Force, April 1998. *1998*

[26] **R. E. Kahn, V. G. Cerf.** "A protocol for packet network intercommunication.", IEEE Transactions on Communications, Vol. 22, No. 5, May 1974, pp. 637-648. *1974*

[27] **T. Keating.** "Internet phone release 4.", Computer Telephony Interaction Magazine. *1995*

[28] **V. Klima.** "Tunnels in hash functions: MD5 collisions within a minute.", IACR Eprint Server. *2006*

[29] **H. Krawczyk, M. Bellare, R. Canetti.** "HMAC: Keyed-Hashing for Message Authentication.", RFC 2104, Internet Engineering Task Force, February 1997. *1997*

[30] **X. Lai, J. Liang.** "Improved collision attack on hash function MD5.", Journal of Computer Science and Technology. *2007*

[31] **D. McGrew, S. Fluhrer.** "Attacks on encryption of redundant plaintext and implications on internet security.", *2000*

[32] **J. Postel.** "Internet Protocol.", RFC 0791, Internet Engineering Task Force, September 1981. *1981*

[33] **J. Postel.** "Transmission Control Protocol", RFC 0793, Internet Engineering Task Force, September 1981. *1981*

[34] **R. Rivest.** "The MD5 Message-Digest Algorithm.", RCF 1321, MIT Laboratory for Computer Science and RSA Data Security, Inc. *1992*

[35] **J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler.** "SIP: Session Initiation Protocol.", RFC 3261, Internet Engineering Task Force, June 2002. *2002*

[36] **H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson.** "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection.", IEEE Transactions on Communications, Vol. 28, No. 4, April 1980, pp. 425 – 432. *1980*

[37] **A. Shamir R. Rivest, L.Adleman.** "A method for obtaining digital signatures and public-key cryptosystems.", Communications of the ACM 21, 1978. *1978*

[38] **T. Ylonen and C. Lonvick.** "The Secure Shell (SSH) Authentication Protocol.", RFC 4252, Internet Engineering Task Force, January 2006. *2006*

[39] **H. Yu, X. Wang.** "How to break MD5 and other hash functions.", EUROCRYPT, vol. 3494 of Lecture Notes in Computer Science. *2005*

[40] **H. Zimmermann.** "RTP: A Transport Protocol for Real-Time Applications.", RFC 3550, Internet Engineering Task Force, July 2003. *2003*

[41] **P. Zimmermann, A. Johnston, Ed.Avaya** "ZRTP: Media Path Key Agreement for Unicast Secure RTP.", Internet-Draft. *2010*