Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Triennale in Informatica

Sicurezza e Multihoming in Sistemi VoIP su Symbian

Tesi di Laurea in Reti di Calcolatori

Relatore:
Dott.
Vittorio Ghini

Presentata da:
Marco Oliviero

II Sessione
Anno Accademico 2009-2010

Dedicata ai nonni Gino e Giuliana ...

Indice

Introduzione

Negli ultimi anni abbiamo assistito a un crescente sviluppo delle tecnologie *VoIP*, concomitante a un proliferare di reti wireless. La crescente diffusione di terminali mobili *multi-homed*, rappresenta una promessa interessante riguardo la possibilitá di utilizzare questi device per supportare la trasmissione in movimento di traffico telefonico a basso costo, sfruttando a pieno le possibilitá offerte dalle interfacce multiple disponibili.

Nonostante la possibilitá teorica di un tale utilizzo, i limiti prestazionali legati alle tecnologie wireless a media distanza come *IEEE 802.11* e la mancata diffusione di protocolli a supporto della mobilitá come *Mobile IPv6*[22], hanno finora impedito di utilizzare tali dispositivi per effettuare traffico *VoIP*.

L'architettura *ABPS* che verrá presentata in questa tesi, offre soluzioni originali a queste problematiche, permettendo a un device mobile dotato di piú interfacce di effettuare traffico *VoIP*, rispettando i requisiti di interattivitá richiesti dalle applicazioni multimediali. La soluzione *ABPS* é di particolare rilevanza poiché non richiede alcuna modifica ai protocolli di rete attualmente in uso, rendendosi pienamente compatibile con le tecnologie esistenti.

L'obbiettivo implementativo di questa tesi é di completare il sottosistema *ABPS* di autenticazione e cifratura dei protocolli *SIP/RTP*, su un dispositivo mobile con sistema operativo Symbian. Lo scopo primario del sottosistema di sicurezza, oltre a garantire la confidenzialitá delle comunicazioni, é quello di identificare univicomanente il mittente di un determinato pacchetto destinato al proxy server *ABPS*, a prescindere da quale sia l'indirizzo IP di provenienza,

in maniera tale da garantire al dispositivo mobile la possibilitá di utilizzare contemporaneamente tutte le interfacce di rete a disposizione.

Di seguito verranno presentati alcuni tra i protocolli piú evoluti, che sono di supporto alle tecnologie *VoIP*, e saranno analizzate alcune problematiche relative alle esigenze di *Quality of Service* legate al mondo della telefonia attraveso Internet.

Sará effettuata una breve descrizione dello standard IEEE 802.11, con lo scopo di introdurre il lettore ai limiti e alle problematiche posti da tale tecnologia.

Verrano quindi presentate alcune delle soluzioni classiche ai problemi di sicurezza delle comunicazioni, con lo scopo di introdurre i protocolli piú sofisticati utilizzati dall'architettura *ABPS*.

Come é stato detto in precedenza, il lavoro di sviluppo ha riguardato l'implementazione del sottosistema di autenticazione e cifratura per il sistema operativo Symbian. Il software di partenza, da cui é iniziato il lavoro, era composto da un client *VoIP* per device Symbian, e da un proxy server *ABPS* funzionante su sistema operativo Linux.

Il client Symbian era affetto da alcuni bug e problemi strutturali che ne limitivano il funzionamento: di fatto non riusciva a completare con successo alcun tentativo di autenticazione al sistema *ABPS*.

Anche il proxy server aveva alcuni difetti, che é stato possibile scoprire grazie all'interazione con il client stesso il quale, proprio per le caratteristiche peculiari dell'architettura su cui é implementato, ha permesso di identificare problemi fino ad ora sconosciuti.

I test delle applicazioni sono avvenuti sfruttando i server della facoltá di Informatica dell'Universitá di Bologna. La rete stessa, a causa della presenza di alcuni firewall, ha consentito di individuare alcune problematiche che sono state solo parzialmente risolte.

Le modifiche apportate al proxy server *ABPS* hanno riguardato principalmente la correzione di bug, il miglioramento delle prestazionie e l'adozione di soluzioni temporanee che impedissero ai firewall Unibo di modificare i

pacchetti in transito.

Un'analisi piú approfondita dei dettagli implementativi e delle soluzioni tecniche adottate, verrá presentata nella relativa sezione.

Telefonia VoIP

Con il termine *VoIP* (Voice over IP) ci si riferisce a un'insieme di tecnologie volte alla trasmissione di comunicazioni vocali, attraverso reti basate sul protocollo *IP*.

Svariati elementi caratterizzano la differenza che intercorre tra questa tecnologia e le classiche reti telefoniche a commutazione di pacchetto *PSTN (Public Switched Telephone Network*).

In primo luogo si evidenzia la sostanziale differenza nella gestione della codifica vocale. Se attraverso le classiche infrastrutture telefoniche, il traffico vocale é tipicamente veicolato 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, in ogni caso, soprattutto per quanto riguarda il cosiddetto *ultimo miglio*, persiste la presenza della trasmissione analogica.

Storia e Caratteristiche

I primi studi in materia risalgono al 1974 quando l'*IEEE* (Institute of Electrical and Electronic Engineers) pubblicó un articolo intitolato "*A Protocol for Packet Network Interconnection*" [38], sebbene la standardizzazione del procollo *IPv4* non avvenne prima del 1981 [32]. La ricerca verteva sulla possibilitá di offrire supporto per la condivisione di risorse tra reti commutate eterogenee.

I primi sviluppi significativi ci furono dopo l'avvento di Internet, in particolare a metá degli anni novanta quando vennero rilasciati i primi software che permettevano di effettuare chiamate vocali tra computer(1995 - VocalTec [23]).

Lo sviluppo delle tecnologie *VoIP* era tuttavia ancora agli albori, mancando 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 e alla successiva standardizzazione, avventuta nel 1999, di quello che é attualmente lo *standard de facto* per le comunicazioni *VoIP*, cioé *SIP (Session Initiation Protocol)*.

L'utilizzo e le evoluzioni di queste tecnologie sono stati fin da allora in continua espansione, particolarmente nell'anno 2003 con la prima release di Skype e nel 2004 col proliferare dei provider *VoIP* commerciali [2].

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.

Utilizzo efficiente della banda

Il primo miglioramento significativo che caratterizza il *VoIP* rispetto alle classiche linee *PSTN*, é un uso piú efficiente del mezzo trasmissivo.

In queste ultime infatti, la quantitá di banda a disposizione per ogni utente é una risorsa fisica, non modificabile salvo interventi di potenziamento sulle infrastrutture. 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 **flessibilitá**:

- **Possibilitá di veicolare molteplici chiamate attraverso uno stesso canale fisico**, senza la necessitá di dover modificare le infrastrutture per aggiungere ulteriori linee.

- **Indipendendenza dalla posizione fisica dell'utente**: é sufficiente un canale di accesso a Internet, o a qualsiasi rete interna basata su *IP*, per dare la possibilitá a un utente di registrarsi presso un provider, effettuare chiamate e segnalare la propria presenza per essere raggiunto.

- **Estensione dei servizi disponibili**: possono essere implementate *feature* aggiuntive come video chiamate, scambio file, lavagne condivise, etc...

Un'altra serie di vantaggi é 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.

La tecnologia *VoIP* infatti permette di:

- **Veicolare il traffico voce attraverso le linee dati** . É evidente il vantaggio economico che puó trarne un'azienda, che in questa maniera non é piú costretta ad affrontare costi di gestione separati, ma puó unificare la trasmissione dati e quella vocale, traendo anche ulteriori vantaggi dalla flessibilitá che offrono le tecnologie *VoIP*.

- **Costi per gli utenti basati sul reale consumo**, piuttosto che sul tempo effettivo di durata della chiamata, si traducono in una diminuzione dei costi. Infatti, il costo di una telefonata considerando la quantitá complessiva di informazione trasferita, piuttosto che la durata di questa, é generalmente molto inferiore.

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

- **Allocazione dimanica delle risorse**. A differenza delle linee *PSTN*, che richiedono collegamenti fisici dedicati per ogni linea telefonica, veicolare il traffico attraverso un'unica linea dati permette di applicare

politiche di distribuzione del traffico, che garantiscono un'utilizzo piú efficiente della banda a disposizione.

Protocolli per il VoIP

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

Vi é tuttavia una serie di protocolli comunemente usati.

UDP: User Datagram Protocol

Standardizzato nel 1980, é uno tra i protocolli fondamentali della rete Internet. Attraverso di esso é possibile scambiare messaggi, chiamati *datagram*, sfruttando il sottostante protocollo *IP* [32].

Ogni canale di comunicazione tra *host* é determinato dalla coppia indirizzo ip e porta.

La mancanza di procedure di *handshake* o di sincronizzazione tra host, denota la natura semplice del protocollo UDP, che non garantisce la trasmissione affidabile e ordinata dei datagram, e non protegge dai duplicati.

L'integritá del singolo datagram é verificabile attraverso un valore di *checksum* a 16 bit, incluso nell'header, e generato con la stessa procedura descritta dal protocollo *TCP* [33] .

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

Un altra sua caratteristica fondamentale é la possibilitá di effetture comunicazioni *broadcast* e *multicast*, particolarmente utile nel caso di conferenze multimediali tra piú *host*.

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*.

É 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.[37]

Si noti che RTP non fornisce meccanismi per garantire la *QoS (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*, per monitorare la *QoS* e trasmettere informazioni che riguardano i partecipanti a una sessione in corso.

SDP: Session Description Protocol

É un protocollo per la descrizione di sessioni multimediali.

SDP gestisce l'annuncio, l'invito e altri metodi di inizializzazione di una sessione multimediale. [20]

Non si occupa del trasporto dei dati, ma permette agli *end-point* di negoziare parametri di sessione, come il tipo di trasmissione, formati, codec e tutta una serie di proprietá 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*, *SRTP (Real-time Streaming Protocol)* e *SIP*, sia come formato a sé stante per la descrizione di sessioni *multicast*.

SRTP: Secure Real-time Transport Protocol

SRTP é un profilo per *RTP*, che garantisce le proprietá di **confidenzialitá**, **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.*[8]

SRTP puó 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 sequenzialitá/sincronizzazione é garantita implicitamente facendo affidamento al campo *seqnum* del protocollo *RTP*.

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

- **Confidenzialitá** del payload del traffico RTP e RTCP.
- **Integritá** dell'intero pacchetto.

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

Altri obbiettivi funzionali del protocollo sono:

- **Supporto per nuovi algoritmi crittografici**: il framework deve garantire la possibilitá di poter utilizzare in futuro nuovi algoritmi, qualora ne vengano scoperti di piú 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 paccheti** 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 proprietá assicurano che *SRTP* é 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** puó fornire materiale crittografico per garantire confidenzialitá e integritá per entrambi i canali *SRTP/SRTCP*. Questo obbiettivo é raggiunto utilizzando *session-key* diverse per i rispettivi canali, derivate in maniera sicura dalla *master-key*.
- **Il refresh periodico della session-key**, limita la quantitá di traffico cifrato con la stessa chiave, rendendo piú difficile il processo di crittanalisi.
- **Salting keys** sono usate per proteggere il protocollo da attacchi di tipo *pre-computation*[27] e *time-memory tradeoff*[10].

Si noti che il protocollo non si occupa della fase di negoziazione delle chiave. Per risolvere questa problematica *SRTP* fa tipicamente affidamento a protocolli di *key-agreement* come *ZRTP* o *MIKEY(Multimedia Internet Keying)*[7].

SIP: Session Initiation Protocol

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

É costruito per essere indipendente dal sottostante livello di trasporto, supportando quindi comunicazioni sia *UDP* che *TCP*, ed é 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.

Inoltre fornisce servizi di autenticazione degli utenti e di autorizzazione per l'utilizzo dei servizi.

Implementa policy per l'instradamento delle chiamate e offre funzionalitá 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 volontá del chiamato di prendere parte nella comunicazione.
- **User capabilities**: determinare il tipo di media e i parametri per stabilire la comunicazione.

- **Session setup** : *ringing*, stabilire i parametri di sessione per enrambi gli *endpoint*.

- **Session management**: trasferimento e terminazione delle sessione, modifica dei parametri e invocazione dei servizi.

SIP non é un sistema di comunicazione integrato vericalmente, ma piuttosto un componente da utilizzare insieme ad altri protocolli *IETF*, 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), **integritá** e **cifratura**.

Il protocollo funziona sia con IPv4 che IPv6.

Definizioni

Di seguito vengono fornite alcune definizioni che descrivono le entitá coinvolte nel procollo:

- **Address-of-Record**: Un indirizzo di record é composto da uno o piú *SIP-URI* che puntano a un dominio con un servizio di locazione, attraverso il quale si é 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* puó essere considerato l'indirizzo pubblico di un utente.

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

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

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

- **Call Stateful**: un proxy é *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* é sempre *transaction stateful*, ma non é detto che valga il contrario.

- **Dialog**: un dialogo é 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*. É identificato da un identificativo di chiamata, un tag locale e uno remoto.

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

- **Location Service**: É 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 piú indirizzi. Il *binding* puó 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 entitá SIP come parte del protocollo. Si dividono In *Request* e *Response*.

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

- **Proxy Server**: é un'entitá 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 é 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**: é 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**: é 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 é definita da un insieme di chiamanti, riceventi, e dagli stream di dati che viaggiano tra loro."[20] Se viene usato il protocollo *SDP*, una sessione é definita dalla concatenazione del *SDP user name, il session id, network type, address type* e gli *address element* nel campo sorgente.
- **SIP Transaction**: una transazione si verifica tra un client e un server, e comprende tutti i messaggi a partire dalla prima *SIP Request* del client fino all'ultima *SIP Response* del server.
- **Stateful Proxy**: é un'entita logica che conserva lo stato delle transazioni tra un client e un server.
- **Stateless Proxy**: é un'entitá logica che non mantiene lo stato della transazione tra client e server. Si limita a inoltrare ogni *SIP Request* verso il server e ogni *SIP Response* verso il client.
- **User Agent Client (UAC)**: é l'entitá logica che crea una nuova *SIP Request*, e usa la macchina a stati del client per inviarla. Il ruolo di *UAC*

dura solamente per il tempo necessario a completare la transazione. In altre parole, se un'entitá invia una *SIP Request*, agisce come *UAC*. Se successivamente riceve una *SIP Request*, allora assumer il ruolo di *UAS*.

- **User Agent Server(UAS)**: é l'entitá logica che genera un *SIP Response* a una *SIP Request*. Il response accetta, rifiuta o ridirezione la request. Come per l'*UAC*, il ruolo di server dura solo fino al completamento della transazione.

- **User Agent (UA)**: é l'entitá logica che puó agire sia da *UAC* che da *UAS*.

Messaggi SIP

SIP é un protocollo testuale, derivato da *HTTP*, 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*, é 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 piú risposte provvisorie (classe 1XX) ma al piú una risposta definitiva.

Le classi definite sono le seguenti:

- **100-199**: indica una risposta informativa e provvisoria (es: quando invitiamo un utente ad una conversazione, una risposta di tipo 180 indica che questo é stato avvisato ma non ha ancora accettato la conversazione).

- **200-299**: indica che la richiesta é stata eseguita con successo.

- **300-399**: indica che la richiesta é 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 lo 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 piú *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 puó capitare che trascorra un discreto intervallo di tempo, che puó durare anche svariati secondi, prima che questo riceva risposta. Quando e se lo *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 giá stata portata a termine, ovvero non permette di interrompere una sessione giá 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 piú 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* é 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 piú rintracciabile. Di conseguenza un *UA* genera tale richiesta anche per rinnovare una precedente registrazione; in tal caso puó anche aggiornare la lista dei contatti.

Quality of Service

Sono stati espressi in precedenza tutti i principali vantaggi e novitá, 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 **interattivitá**. Il passaggio dalle classiche linee dedicate *PSTN*, alle reti dati con tecnologia *IP*, introduce degli *overhead* nella comunicazione.

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

- **One-way delay (latenza)**: misura il tempo che impiega un pacchetto a percorrere la strada tra due *end-point*. Al crescere di questo valore, si ha un degrado della qualitá della conversazione, in termini di interattivitá. Le linee guida fornite da *ITU-T* [6] indicano una latenza di al piú di 150 ms per ottenere una qualitá audio soddisfacente.

- **Packet delay variation (Jitter)**: misura la variazione nel tempo del *one-way delay*, ignorando i pacchetti persi lungo il percorso [15]. 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 qualitá della trasmissione.

- **Packet Loss rate**: misura la percentuale di pacchetti persi. Alti valori di questo indice, provocano un degrado della conversazione in termini di qualitá della voce. Per garantire una certa qualitá 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 piú 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 piú 64 Kbps in trasmissione.

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

Per quanto riguarda il tempi di latenza, non c'é molto che si possa fare, in quanto sono fortemente dipendenti dalla linea usata per la trasmissione. Ridurre il consumo di banda, puó aiutare a non sovraccaricare il canale.

La prima scelta opportuna é 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 qualitá 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 rimpiazzandi con 0.
- **Waveform substitution**: i frame mancanti vengono rimpiazzati da porzioni di traffico vocale disponibili, che vengono replicate. Questa tecnica é molto popolare poiché di facile implementazione.
- **Model-based Method**: é una delle tecniche piú 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*.

Wireless e Terminal Mobility

Standard IEEE 802.11

Lo standard si occupa di definire specifiche, per un livello *MAC (Medium Access Control)* e diversi livelli fisici (*PHY*), per la connettivitá di stazioni (*STA*) fisse, portatili e mobili, all'interno della rete locale (*local area network*). Descrive inoltre delle prassi regolamentative per standardizzare l'accesso a una o piú bande di frequenza in un contesto di comunicazione locale.[3] Piú nello specifico lo standard:

- **Definisce le funzioni e i servizi richiesti**, a un device che soddisfa lo standard, per operare nel contesto di reti *ad-hoc* o infrastrutturate, e definisce gli aspetti relativi alla mobilitá del *STA* all'interno di queste reti.

- **Definisce le procedure MAC** di supporto ai servizi di trasmissione asincrona *MSDU (MAC Service Data Unit).*

- **Definisce svariate tecniche di signalling fisico e funzioni di interfaccia**, che sono controllate dal IEEE 802.11 MAC.

- **Permette operazioni a un device 802.11**, all'interno della *WLAN (Wireless Local Area Network)*, che puó coesistere in sovrapposizione con altre *WLAN*.

- **Descrive i requisiti e le procedure per garantire la confidenzialitá** dei dati trasferiti attraverso il mezzo condiviso *WM* (Wireless Medium), e descrive la procedura di autenticazione.
- **Definisce meccanismi per la selezione dinamica delle frequenze** *(DFS - Dynamic Frequency Selection)* e il **controllo della potenza trasmissiva** (*TPC - Transmit Power Control*), che puó essere usato per soddisfare la regolamentazione delle operazioni sulle frequenze dei 5 GHz.
- **Definisce le procedure MAC di supporto alle applicazioni LAN con esigenze di QoS**, incluse il trasporto voce, audio e video.

802.11e

Lo standard 802.11e introduce il concetto di classi/tipologie di traffico; vengono ridefinite le esistenti *DCF (Distributed Coordination Function)* e *PCF (Point Coordination Function)*, ridenominate rispettivamente *EDCA (Enhanced Distributed Channel Access)* e *HCCA (HCF (Hybrid Coordinator Function) Controlled Channel Access)*.

- **EDCA**: viene mantenuta l'impostazione di base della *DCF*: *CSMA/CA*, binary exponential backoff e *RTS/CTS* opzionale. In aggiunta viene introdotto il concetto di classe di traffico; in base alla tipologia di traffico trasmessa da un terminale, saranno scelti opportuni valori di *AIFS (arbritation interframe space)* e di *CWmin* e *CWmax* per la *contention window*. L'obiettivo é quello di permettere un'attesa inferiore per accedere al canale, da parte di tipologie di traffico *real-time*.

  Per limitare l'intervallo di tempo, per il quale é possibile occupare il canale in seguito ad una contesa vinta, viene introdotto il *TXOP (Trasmission Opportunity)*; questo valore, che varia a seconda della tipologia di traffico, stabilisce il tempo massimo di occupazione.

- **HCCA**: viene mantenuta l'impostazione di base della *PCF*, aggiungendo ad essa i concetti di classi di traffico e *TXOP*.

Sono inoltre introdotte funzionalitá di:

- **Admission Control**: permette all'*AP* di impedire l'accesso al canale ad un terminale con una determinata tipologia di traffico. Un *AP* tipicamente ha una bassa capacitá in termini di gestione di applicazioni *real-time*. Nel caso del *VoIP* non é possibile supportare piú di 10 sessioni contemporaneamente, pena un degrado netto della qualitá del servizio. Una undicesima sessione *VoIP* causerebbe un disservizio, senza poter essere servita correttamente; in tali casi l'*Admission Control* permette all'*AP* di proteggere le attuali sessioni da questo fenomeno, rifiutando l'undicesima sessione.

- **QosNoAck**: questo parametro, specificabile per ogni frame che un terminale desidera spedire all'*AP*, indica di non generare un *ACK* di risposta in seguito alla corretta ricezione. Questo serve ad evitare tentativi potenzialmente inutili di ritrasmissione, da parte del terminale, di dati *real-time*.

- **DLS (Direct Link Setup)**: permette ed abilita la comunicazione diretta tra terminali all'interno dello stesso *BSS*.

Come evidenziato anche in [30], l'utilizzo della *DCF*, o meglio ancora della *EDCA*, non fornisce garanzie ad una applicazione *real-time* sui tempi di accesso al canale. Sebbene le modifche introdotte migliorino sicuramente la situazione originale, priorizzando tipologie di traffico *real-time*, non si tratta di garanzie deterministiche bensí statistiche.

In sintesi non é garantito che ad ogni contesa venga priorizzato il traffico *real-time (VoIP)*, ma solo che mediamente questo risulti priorizzato. Puó tuttora capitare infatti, che una applicazione a bassa prioritá (es: mail), si trovi ad avere un valore di *backoff timer* inferiore a quello di una ad alta prioritá, permettendo alla prima di vincere la contesa.

L'unica soluzione a tali problematiche viene fornita dall'utilizzo della *PCF*, o meglio ancora *HCCA*; questa modalitá tuttavia é definita come opzionale dallo standard ed é scarsamente diffusa e supportata dai dispositivi hardware attualmente in commercio.

Le problematiche di Sicurezza

Confidenzialitá

Il termine *confidenzialitá* si riferisce all'occultamento di informazioni o risorse. Meccanismi di controllo d'accesso, sono di supporto alla confidenzialitá. Uno di questi meccanismi é la *crittografia* che si occupa di cifrare dati per renderli incomprensibili.

Una chiave di cifratura, controlla l'accesso ai dati non cifrati, ma rappresenta essa stessa un altro dato da proteggere. [11]

La necessitá di mantenere informazioni segrete, nata dall'uso dei computer nell'ambito dell'industria e di strutture militari o governative, é attualmente un'esigenza che coinvolge sempre di piú anche gli utenti comuni.

La diffusione di massa dei personal computer, ha portato anche i normali cittadini ad avere l'esigenza di mantere alcuni dati confideziali, 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 un semplice *sniffing passivo*.

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 nascondere il significato. Il termine *crittanalisi* invece, indica lo studio di come forzare sistemi crittografici che sono

definiti come:

Theorem 0.0.1. *Un sistema crittografico é una 5-tupla (*$\mathbf{E}, \mathbf{D}, \mathbf{M}, \mathbf{K}, \mathbf{C}$*), dove* $\mathbf{M}$ *é un insieme di messaggi in chiaro,* $\mathbf{K}$ *é l'insieme delle chiavi,* $\mathbf{C}$ *é l'insieme dei messaggi cifrati,* $\mathbf{E} : \mathbf{M} \times \mathbf{K} \rightarrow \mathbf{C}$ *é l'insieme delle funzioni di cifratura e* $\mathbf{D} : \mathbf{C} \times \mathbf{K} \rightarrow \mathbf{M}$ *l'insieme delle funzioni di decifratura.*

Lo scopo della crittografia é 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 $\mathbf{D}$ e $\mathbf{E}$).

Gli attacchi possibili sono i seguenti:

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

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

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

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

Con il termine *cifrario perfetto* si vuole sottolineare che, per scoprire una chiave, non esiste un metodo piú 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 per un calcolatore farne una visita esaustiva.

Algoritmi di Cifratura Simmetrici

I cifrari classici (chiamati anche a *chiave-singola* o *simmetrici*) sono quei sistemi dove la chiave usata per cifrare é la stessa usata per decifrare.

Piú formalmente:

Theorem 0.0.2. *Un cifrario é simmetrico se:*
$\forall E_k \in C \ e \ \forall k \in K, \exists D_k \in D \mid D_k = E_k^{-1}$

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

Sono generalmente piú efficienti dei cifrari asimmetrici. Un cifrario simmetrico non puó garantire la proprietá di non ripudiabilitá di un messeggio.

DES (Data Encryption Standard) é ufficialmente considerato non piú sicuro dal 1999. Al suo posto, il National Institute of Standadds and Technology ha proposto, nel 2001, l'algoritmo noto come *AES (Advanced Encryption Standard)*[29].

Algoritmi di Cifratura Asimmetrici

Nel 1972 Diffie ed Hellman [1] proposero un nuovo modello di crittografia, che utilizzasse due chiavi, una per cifrare e una per decifrare il messaggio.

La chiave per cifrare é **pubblica**: per trasmettere un messaggio in maniera sicura, lo si cifra con la chiave pubblica del destinatario.

La chiave per decifrare é **privata**. Deve essere tenuta segreta, e serve per decriptare i messaggi associati alla rispettiva chiave pubblica.

Poiché una delle due chiavi é 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 infattibile** derivare la chiave privata, conoscendo quella pubblica.

- deve essere **computazionalmente infattibile** scoprire la chiave privata effettuando un *chosen plain-text attack*.

RSA[34] soddisfa queste richieste garantendo secretezza ed autenticazione.

Un caso pratico : descrizione dell'algoritmo RSA

RSA trova le sue fondamenta nelle proprietá dei numeri primi. Di seguito vengono presentate alcune proprietá per introdurre l'algoritmo:

Siano:

$$(Z)_n = 0, 1, 3, \ldots, n-1 \qquad \text{numeri interi modulo n} \tag{1}$$

$$(Z)_n^* = \qquad \text{numeri interi primi con n} \tag{2}$$

$$GCD(m, n) = \qquad \text{massimo comun divisore tra n e m} \tag{3}$$

Siano p e q due numeri primi allora:

$$\Phi(p) = p - 1 \tag{4}$$

$$GCD(p) = p - 1 \implies \Phi(n, m) = \Phi(n)\Phi(m) \tag{5}$$

$$\Phi(pq) = \Phi(p)\Phi(q) \tag{6}$$

Il processo di generazione delle chiavi é il seguente:

1: Scelgo due numeri primi grandi p e q
2: calcolo $n = p * q$
3: scelgo e tale per cui $GCD(e, \Phi(n)) = 1$
4: calcolo d tale per cui $d * e \mod \Phi(n) = 1$
5: chiave privata $= (e, n)$

6: chiave pubblica = (d, n)

La sicurezza di RSA contro attacchi di tipo *brute force* é garantita utilizzando uno spazio delle chiavi di dimensione adeguata.

Gli attacchi matematici che si possono portare al cifrario sono due :

- fattorizzo n e ottengo p e q
- calcolo $\Phi(n)$, senza fattorizzare n , e poi calcolo $d = e^{-1}(\mod \Phi(n))$

Fortunatamente entrambi gli attacchi sono caratterizzati da un costo computazionale pari alla fattorizzazione di n che, se scelto sufficientemente grande, rende l'attacco infattibile.

Autenticazione

Con il termine *autenticazione* si intende il processo di *binding* tra un'identitá e un soggetto. In altre parole é il processo attraverso il quale si é 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 proprietá. 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.

L'*HMAC* é una tecnica usata per autenticare i messaggi trasmessi tra due *end-point*, mentre *challenge/response* é una tecnica utilizzata per autenticare tra loro un utente e un sistema remoto.

HMAC

Avere un metodo per verificare l'integrita di un'informazione trasmessa, o memorizzata su un dispositivo non affidabile, é una necessitá nel mondo dei computer e delle comunicazioni. I meccanismi che forniscono tali controlli di integritá, 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 [25] é un meccanismo per l'autenticazione di messaggi che fa uso di funzioni di hash crittografiche. Puó essere utilizzato insieme a una qualsiasi funzione iterativa come *MD5* o *SHA1*.

La sicurezza di *HMAC* dipende dalle proprietá delle sottostanti funzioni di hash.

Si noti che da quando *MD5* é considerato non piú sicuro, poiché risulta vulnerabile ad un'attacco basato sulle collisioni (*collision search attack*) [17], il suo uso é deprecato.

Il meccanismo *HMAC* é basato sul documento [26], dove l'algoritmo viene presentato e ne viene proposta 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 é 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 che le funzioni di hash siano facilmente sostituibili, qualora se ne ne scoprano di piú veloci e sicure.

Definizione di HMAC

Sia **H** una funzione di hash crittografica e **K** una chiave segreta.

Si assume che **H** calcoli l'hash dei dati iterando una funzione di compressione base su blocchi di dati.

Chiamiamo **B** la lunghezza in byte di questi blocchi ed **L** la lunghezza in byte dell'hash di output (L = 16 per MD5, L=20 per SHA-1). La chiave deve essere lunga al massimo quanto la dimensione di un blocco **B**. Le applicazioni che usano una chiave con lunghezza maggiore di **B** , devono preliminarmente calcolare l'hash della chiave, e successivamente usare tale hash di **L** byte come chiave dell'algoritmo *HMAC*.

Si definiscano poi due stringhe fisse, diverse tra loro, come segue:

$$\text{ipad} = \text{il byte 0x36 ripetuto B volte} \tag{7}$$

$$\text{opad} = \text{il byte 0x5C ripetuto B volte} \tag{8}$$

Per calcolare la funzione HMAC sui dati di input :

$$H(K \oplus \text{opad}, H(K \oplus ipad, text)) \tag{9}$$

L'algoritmo di *HMAC* é il seguente:

1: hash_1 = **B** $\|$ $(0x00 \times (B - len(K)))$;
2: hash_2 = hash_1 $\oplus$ ipad ;
3: hash_3 = text $\|$ hash_2;
4: hash_4 = H (hash_3);
5: hash_5 = hash_1 $\oplus$ opad;
6: hash_6 = hash_4 $\|$ hash_5;
7: hash_result = H (hash_6);

Scelta della chiave

La chiave da usare con *HMAC* puó essere di qualsiasi lunghezza. In ogni caso, é 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, usando un generatore corretto di sequenze pseudo-casuali inizializzato con un seme casuale, e deve essere cambiata periodicamente.

Challenge-Response

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

Un sistema di questo tipo infatti soffre di un problema fondamentale: la password puó essere utilizzata per successive autenticazione. Se un attaccante sta in qualche modo sniffando il traffico, puó carpire il segreto e utilizzarlo per autenticarsi in seguito. Con il *challenge-response* invece, la chiave trasmessa, basata su un qualche segreto condiviso, cambia ad ogni successiva autenticazione, garantendo che anche se viene scoperta non puó essere usata per sessioni successive a quella in corso.

Il meccanismo é definito come segue:

Theorem 0.0.3. *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* **f** . *Un meccanismo di autenticazione é* challenge-response *se il sistema* **S** *invia un messaggio casuale* **m** *(challenge) al sistema* **U**, *che risponde con il risultato della funzione* $r = f(\mathbf{m})$ *(response).* **S** *é in grado di validare* r *calcolando autonomamente il risultato della funzione.*

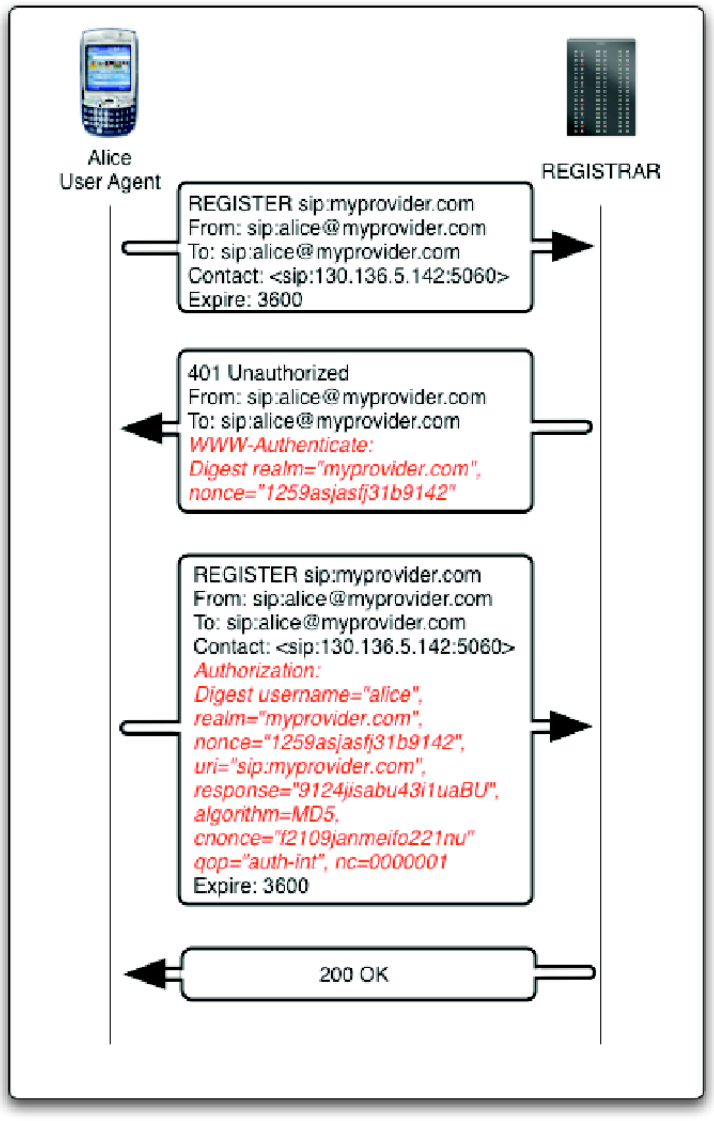
Autenticazione SIP/HTTP Digest

L'*HTTP Digest Authentication* [18] é basato su una verifica crittografica di una password in chiaro, condivisa tra client e server.

É 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 puó verificare l'hash della

password, mentre un attacco *MiTM*, una volta intercettato l'hash, non potr 'aa 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*.



Come mostrato in figura, l'autenticazione digest di una registrazione *SIP* é una procedura formata da 4 fasi (*four-way handshake*):

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

- **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**: é il *real name*: 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)**: puó essere "auth" o "auth-int", o puó non esserci del tutto. Influenza il modo in cui il *Response* deve essere calcolato, come sará spiegato in seguito.
  - **Nonce**: rappresenta il *Challenge* generato dal server ed é utilizzato dal client per il calcolo del response.

    Sará 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'attualitá del *nonce*, che sará inserito negli header *Authorization* delle richieste successive.
  - **Opaque**: é un campo utilizzato solo dal server, in cui puó inserire ció che vuole per scopi sconosciuti al client, il quale dovrá semplicemente ricopiarlo nei messaggi di richiesta successivi. Spesso viene utilizzato per memorizzare informazioni di stato relative alla sessione di autenticazione.
  - **Algorithm**: puó 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*, il cui calcolo includerá 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* é 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 é assente, si assume sia "MD5".

- **Stale**: se *true*, indica al client che la sua precedente richiesta é stata rifiutata perché il *nonce* era scaduto, ma il *Response* era stato calcolato con la giusta password. In questo caso il client userá 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 conterrá 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 qualitá 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 é utilizzato come input nel calcolo del response, permettendo al server di rilevare dei *replay-attack*
  - **Client nonce (cnonce)**: questo parametro deve essere presente se é stato specificato un parametro qop. É il nonce generato dall'UAC e utilizzato nel calcolo e nella validazione del response. Il suo scopo é di proteggere la password contro attacchi di tipo *chosen plaintext*. Un attacco di questo tipo potrebbe essere lanciato da un'entitá 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 puó includere un header Authentication-Info, il cui scopo principale, nel contesto *SIP*, é 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 qualitá 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 qualitá di protezione (qop) é impostata ad "auth" o non é presente, il termine A2 del server response authentication viene calcolato come:

$$\text{A2} = '':'' \| URI \tag{10}$$

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

$$A2 = '':'' \| URI \| '':'' \| H(body) \tag{11}$$

Descriveremo di seguito la procedura utilizzata dal client per calcolare il response. Siano:

$$KD(secret, data) = H(secret \| '':'' \| data) \tag{12}$$

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

Quando la qualitá della protezione (qop) é impostata ad "auth" o "auth-int", il response é calcolato come:

$$response = KD(H(A1), nonce) \| '':'' \| nc \| '':'' \| cnonce \| '':'' \| qop \| '':'' \| H(A2)) \tag{13}$$

Quando invece il parametro qop é assente:

$$response = KD(H(A1), nonce) \| '':'' \| H(A2)) \tag{14}$$

Se il parametro del campo "algorithm" é "MD5" o non é specificato nessun algoritmo, il termine A1 é calcolato come:

$$A1 = username \| '':'' \| realm \| '':'' \| passwd \tag{15}$$

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

$$A1 = H(username \,\|'':''\|\, realm \,\|'':''\|\, passwd \,\|'':''\|\, nonce \,\|'':''\|\, cnonce \tag{16}$$

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 é "auth" o non é specificata, il termine A2 si calcola come:

$$A2 = method \,\|'':''\|\, URI \tag{17}$$

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

$$A2 = method \,\|'':''\|\, URI \,\|'':''\|\, H(body) \tag{18}$$

Considerazioni sulla sicurezza dell'autenticazione HTTP Digest

L'autenticazione *HTTP Digest* soffre di alcune debolezze in termini di sicurezza, nonostante rimanga la tecnica di autenticazione piú utilizzata dal protocollo SIP. La debolezza crittografica dell'algoritmo MD5 é stata ampiamente studiata e documentata nel corso degli anni. La vulnerabilitá 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 [9] [28][21][40]). 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[24].

Allo stato attuale non vi é piú alcuna buona ragione per continuare ad usare *MD5*, considerando che esistono algoritmi piú robusti come *SHA-1* e *SHA-2*.

Gli unici campi del messaggio *SIP* di cui viene garantita l'integritá sono il *Digest-URI* e opzionalmente il corpo del messaggio SIP, selezionan-

do il meccanismo di protezione “auth-int”. Un attacco *MiTM* potrebbe alterare facilmente il contenuto del campi Contact nell'header del messaggio REGISTER. Verranno analizzate in seguito le modifiche introdotte dall'architettura *ABPS*, per rafforzare il meccanismo di autenticazione.

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'identitá e una chiave.
- **Generazione** delle chiavi.
- **Mantenimento** delle chiavi.
- **Revoca** delle chiavi.

Il problema della distribuzione delle chiavi, rappresenta uno dei punti piú delicati per garantire una reale confidenzialitá 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 é la chiave stessa che deve essere condivisa. In un sistema asimmetrico, 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 puó 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 entitá coinvolte

possono scambiare dati tra loro, ma una terza entitá non deve essere in grado di derivare la chiave analizzando i dati scambiati.

- Le due entitá possono decidere di dare fiducia a un terza parte coinvolta.
- Il sistema di cifratura e i protocolli usati devono essere noti. L'unica informazione che deve rimanere segreta é la chiave crittografica.

Non esistono soluzioni standard per questo problema. La scelta di un protocollo rispetto ad un altro é 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 verrá descritta in seguito.

Protocollo Diffie-Hellman

É un protocollo crittografico per lo scambio di chiavi. Nato nel 1976 da un idea di Whitfield Diffie e Martin Hellman[16], sulla base dei precedenti studi condotti sulla crittografia asimmetrica da parte di Ralph Merkle.

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

Descrizione dell'algoritmo

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

$$Y = \alpha^x \bmod q, \qquad 1 \leq X \leq q-1 \tag{19}$$

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

$$X = \log_\alpha Y \bmod q \qquad 1 \leq Y \leq q-1 \tag{20}$$

Il calcolo di Y a partire da X é 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 é 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 reso pubblico il valore Y_i calcolato come segue:

$$Y_i = \alpha^{X_i} \bmod q \tag{21}$$

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

$$K_{ij} = \alpha^{X_i X_j} \mod q \tag{22}$$

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

$$K_{ij} = Y_j^{X_i} \bmod q = (\alpha^{X_j})^{X_i} = \alpha^{X_j X_i} = \alpha^{X_j X_i} \bmod q \tag{23}$$

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

$$K_{ij} = Y_i^{X_j} \bmod q \tag{24}$$

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

$$K_{ij} = Y_i^{(\log_\alpha Y_j)} \bmod q \tag{25}$$

Come visto in precedenza, questo calcolo non é 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 proprietá di *perfect forward secrecy.*

Theorem 0.0.4. ***Perfect forward secrecy (PFS)*** *é una proprietá associata a 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 verrá utilizzata per cifrare una singola conversazione.*

Un tale sistema gode della proprietá 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 é 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 puó avviare due sessioni Diffie-Hellman distinte con le altre due entitá, e intercettare in questa maniera il traffico.

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'identitá, non si possono applicare meccanismi di firma digitale.

Il sistemi di crittografia asimmetrica invece, riescono a garantire proprietá piú robuste come la *firma digitale*. Un utente puó apporre la firma a un messaggio cifrandolo con la propria chiave privata. Il destinatario, avendo a disposizione la chiave pubblica puó calcolare la funzione inversa verificando che il documento non sia stato alterato. Nella pratica comune non é il documento ad essere firmato, ma un hash di questo generato opportunamente. Un documento firmato possiede la proprietá aggiuntiva di *non ripudiabilitá*, che garantisce che il messaggio é autentico e puó essere stato generato solo conoscendo la chiave privata.

Si consideri la seguente notazione:

$$X \to Y : \{Z\}k \tag{26}$$

dove l'entitá X manda all'entitá 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 entitá esterna , considerata fidata da entrambi.

Si puó introdurre la nozione di certificato come:

Theorem 0.0.5. *Un* ***certificato*** C *é un token che permette di effettuare il binding tra un'identitá (es.Alice) e 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:*

$$C_{alice} = \{e_{Alice} \parallel Alice \parallel T\}d_{Cathy} \tag{27}$$

Le due tipologie piú diffuse di certificati sono:

- **Certificati X.509v3** Per validare un certificato, l'utente deve essere in possesso della chiave pubblica dell'entitá che ha emesso il certificato (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 é valida cosí come la chiave pubblica. L'utente controlla infine il periodo di validitá 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 é possibile. Abbiamo quindi a che fare con una molteplicitá 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 é ampiamente usato per garantire confidenzialitá 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 unitá chiamate *packets*. Una singola chiave pubblica, puó essere firmata da piú 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 é indicato nel certificato, quelli PGP indicano all'interno del certificato stesso il livello di fiducia, ma questo puó avere valori diversi a seconda del firmatario. Inoltre é 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 proprietá crittografiche robuste di tali sistemi, l'utilizzo pratico deve essere effettuato con le dovute precauzioni e pone piú di qualche problema.

Oltre alle difficoltá legate all'esistenza di differenti protocolli e differenti *CA (Certification Authority)* incompatibili tra loro, Bruce Schneier nell'articolo [13] pone i seguenti problemi:

- **Di chi ci fidiamo e perché?**

  C'é un rischio correlato ad un uso impreciso del termine *trust*. Un *CA* é spesso definito "trust". Nella letteratura crittografica, l'unico significato di questa parola é 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'autoritá di garantire questo? Chi l'ha resa fidata?

- **Chi sta usando la mia chiave?**

  Su che supporto é 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 proprietá di non repudiabilitá. In certi stati (es.Utah e Washington) le leggi prescrivono che, se la chiave pubblica di un utente é stata firmata da un'autoritá di certificazione, questo é legalmente responsabile per l'utilizzo che ne viene fatto.

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

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

- **Il CA é un authority?**

  Ammesso che una *CA* abbia un'autoritá nel rilasciare certificati, ma ha anche qualche autoritá sul contenuto di questi? Per esempio un certificato *SSL* contiene il nome *DNS* del server. Esistono autoritá qualificate per l'assegnazione di nome *DNS*, ma nessuna *CA* presente nella lista dei browser piú comuni ha un'autoritá 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 autoritá per i contenuti certificati, hanno sviluppato una struttura di certificazione divisa in due parti: una *RA* (Registration Authority) gestita dall'organizzazione che ha autoritá sui contenuti, in comunicazione attraverso un canale sicuro con la *CA* che emette i certificati. Il modello *RA-CA* é senza dubbio meno sicuro, in quanto permette ad un entitá priva di autoritá sul contenuto (la *CA*), di rilasciare un certificato con quei contenuti.

- **Come ha fatto il CA a identificare il proprietario del certificato?**

  Un'autoritá di certificazione dovrebbe identificare un soggetto prima di rilasciargli un certificato. Come puó farlo?

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

  É evidente che i certificati offrano un supporto importante per le pratiche di sicurezza. Essi tuttavia non sono una panacea a tutti i mali e, soprattutto, non rappresentano una funzionalitá che basta aggiungere a un sistema per renderlo sicuro, ma devono essere usati con le dovute attenzioni affinché possano essere davvero efficaci.

  Alcune questioni sulle pratiche adottate nell'utilizzo dei certificati debbono quindi essere prese in considerazione, in particolare per quan-

to riguarda i tempi di validitá, i metodi di revoca e, soprattutto, l'interazione con l'utente.

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

  Prima di affidarsi ai certificati per garantire la sicurezza di un sistema, vanno valutate con attenzione le reali esigenze che devono essere soddisfatte. L'eventuale adozione di questa pratica, non deve farsi influenzare dall'apparente semplicitá con cui questi meccanismi possano essere utilizzati, ma deve affidarsi ad un'analisi piú ampia sul grado di sicurezza che si vuole ottenere.

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

Tecniche di attacco

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

- **Man In The Middle (MITM)**

  Questa tecnica é 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. É uno degli attacchi piú pericolosi. Qualsiasi protocollo di scambio di chiavi, che non si avvale di una qualche forma di autenticazione tra le parti coinvolte, é suscettibile a questo tipo di attacco. Il protocollo Diffie-Hellman, nella sua forma originale é 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 é adeguatamente sicuro, un attaccante in possesso di un messaggio,(ad esempio un pacchetto in transito per la registrazione a un servizio), puó 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 entitá, che vogliono accordarsi sui parametri per stabilire una comunicazione sicura, non sono protetti poiché non é ancora avvenuta una reciproca autenticazione. Un'attacante, sfruttando un *MiTM*, puó rimuovere dalla lista degli algoritmi supportati quelli piú sicuri, lasciando quelli piú deboli.

  Il risultato di questo attacco é che, pur supportando meccanismi sicuri di autenticazione, le due entitá coinvolte si accorderanno per utilizzare un metodo vulnerabile.

ZRTP

ZRTP[31] é un protocollo per lo scambio di chiavi crittografiche, che utilizza il protocollo Diffie-Hellman durante la fase setup della chiamata attraverso un *media path*. É trasportato attraverso la stessa porta usata da un flusso multimediale *RTP*, precedentemente stabilito attraverso un protocollo di signalling come *SIP*. Il protocollo permette di generare un segreto condi-

viso tra gli end-point, che puó 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 proprietá 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 continuitá sulla chiave. Viene memorizzato parte del materiale crittografico, per poter essere utilizzato in sessioni successive, che sará mischiato col valore della chiave condivisa DH nella successiva chiamata. Le proprietá di continuitá sulla chiave, sono analoghe a quelle del protocollo *SSH*[41].

Tutto questo é realizzato senza fare affidamento su *PKI*, certificati, *trust model*, *CA*, e altre complessitá 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*. Puó essere usato scoprendo che un client lo supporta, senza una precedente indicazione del protocollo di signalling. Questo fornisce capacitá *best effort* a *SRTP*. Riduce inoltre la complessitá dell'implementazione, minimizzando la dipendenza dal livello di signalling e quello multimediale. Tuttavia, se il supporto a *ZRTP* viene indicato nella fase di signalling, attraverso l'attributo *zrtp-hash* nell'header *SDP*, allora il protocollo gode di alcune proprietá

aggiuntive. Inviando un hash del messaggio *ZRTP Hello*, il protocollo fornisce un binding tra il canale di signalling e quello multimediale. Se questo viene fatto attraverso un canale di *signalling* che fornisce protezione ent-to-end sull'integritá, allora lo scambio di chiavi é protetto automaticamente da un attacco *MiTM*. É concepito per sessioni multimediali unicast. Per le conferenze multimediali, ogni coppia partecipante alla sessione deve effettuare una negoziazione ZRTP separata.

Descrizione del protocollo

Il protocollo puó essere suddiviso in 3 fasi:

- **Discovery**

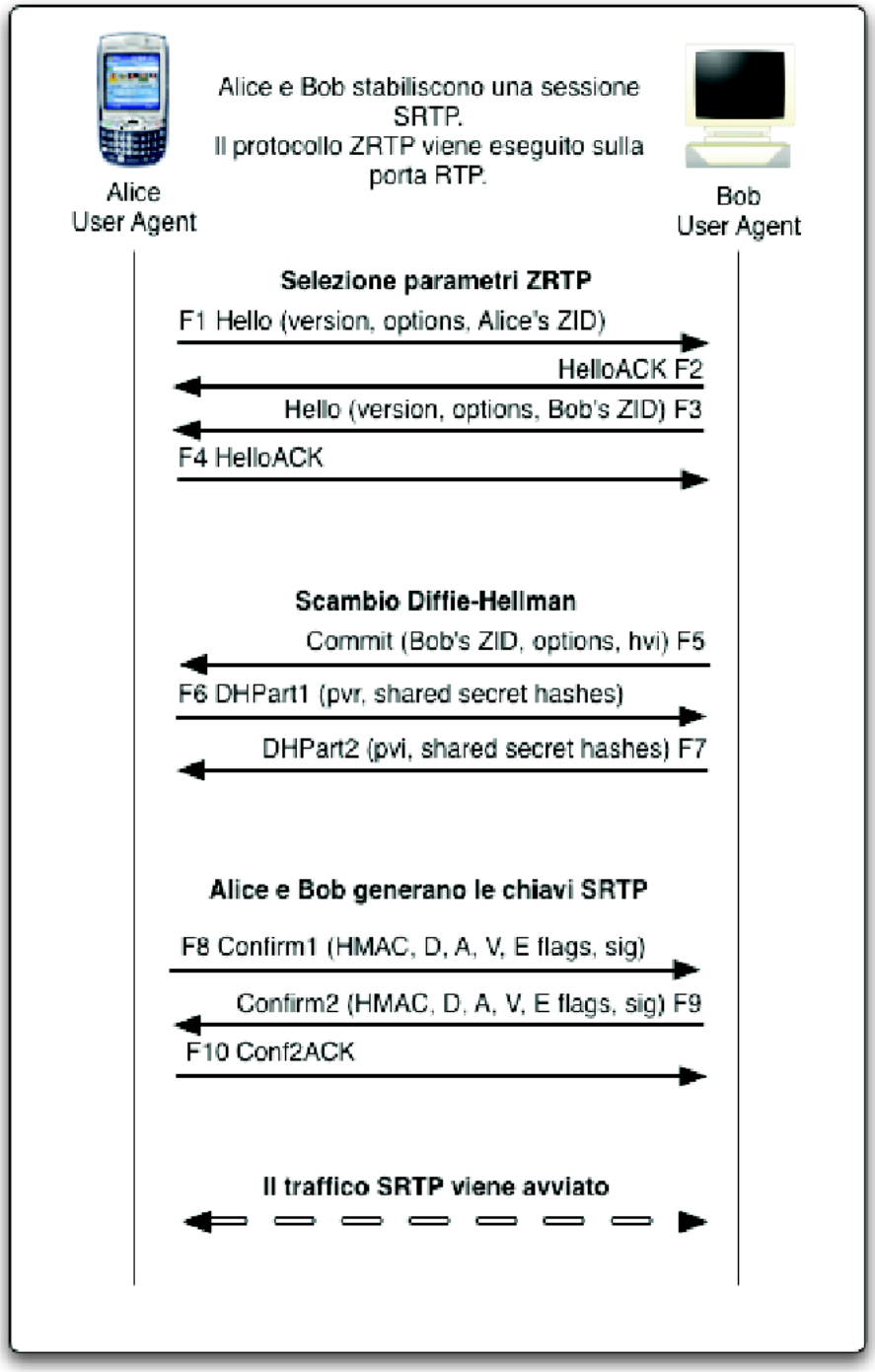
  Durante questa fase gli end-point ZRTP si scambiano informazioni sugli algoritmi supportati e le opzioni. Le informazioni sono trasportate attraverso gli **Hello message**.

  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**
  - **ciphe type**
  - **authentication method and tag lenght**
  - **key agreement type**
  - **SAS algotithms supported**
  - **ZID**

  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 piú robusto.


Alice e Bob stabiliscono una sessione SRTP.
Il protocollo ZRTP viene eseguito sulla porta RTP.
Alice
User Agent
Bob
User Agent
Selezione parametri ZRTP
F1 Hello (version, options, Alice's ZID)
HelloACK F2
Hello (version, options, Bob's ZID) F3
F4 HelloACK
Scambio Diffie-Hellman
Commit (Bob's ZID, options, hvi) F5
F6 DHPart1 (pvr, shared secret hashes)
DHPart2 (pvi, shared secret hashes) F7
Alice e Bob generano le chiavi SRTP
F8 Confirm1 (HMAC, D, A, V, E flags, sig)
Confirm2 (HMAC, D, A, V, E flags, sig) F9
F10 Conf2ACK
Il traffico SRTP viene avviato

- **Commit Contention**

  Dopo che entrambe le parti hanno ricevuto dei messaggi di hello compatibili, un messaggio *Commit* puó essere inviato per inziare lo scambio di chiavi. L'end-point che invia il *commit* viene chiamato *initiator*, la sua controparte é chiamata *responder*.

  In caso entrambi inviano 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 s1, 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 é usato come indice di questa cache.

  I messaggi DHPart1 e DHPart2 contengono una lista di hash di questi segreti, che permettono alle due entitá coinvolte un confronto per capure quali segreti usare per il calcolo della chiave di sessione. Se non é 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) puó essere definito dall'UA VoIP come chiave disponibile out-of-band. In alcuni casi puó essere fornito dal livello di signalling. 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 s1, s2 e s3 sono calcolati in maniera tale che possano essere tutti utilizzati in seguito per calcolare s0.

  Il valore di **s1** puó essere il valore rs1 o rs2 dell'*initiator*, oppure *null*. La scelta viene effettuata nella seguente maniera:

- se il valore rs1 dell'initiator corrisponde al valore rs1 o rs2 del responder, allora

  $s1 = rs1$

- se e solo se il confronto precedente fallisce, e se il valore rs2 dell'intiator corrisponde al valore rs1 o rs2 del responder, allora :

  $s1 = rs2$

- altrimenti non sono disponibili chiavi precondivise e:

  $s1 = null$

Il segreto condiviso **s2** 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 **s3** che eventualmente puó 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 piú significativi.

Il calcolo é il seguente:

1: rs1IDr = MAC(rs1, Responder)
2: rs2IDr = MAC(rs2, Responder)
3: auxsecretIDr = MAC(auxsecret, Responder's H3)
4: pbxsecretIDr = MAC(pbxsecret, Responder)
5: rs1IDi = MAC(rs1, Initiator)
6: rs2IDi = MAC(rs2, Initiator)
7: auxsecretIDi = MAC(auxsecret, Initiator's H3)
8: pbxsecretIDi = MAC(pbxsecret, Initiator)

Il responder invia rs1IDr, rs2IDr, auxsecretIDr, e pbxsecretIDr nel messaggio DHPart1 . L'initiatior invia rs1IDi, rs2IDi, auxsecretIDi, e pbxsecretIDi nel messaggio DHPart2. Il respondere usa i valori calcolati localmente per effettuare il confronto.

- **Key-Agreement**

ZRTP mette a disposizione tre modalitá per scambiare le chiavi:

- Diffie-Hellman mode
- Pre-shared mode
- Multistream mode

Verrá analizzato solo lo scambio Diffie-Hellman, in quanto l'unico rilevante al fine dell'esposizione. La versione classica di questo algoritmo é stata presentata in precedenza, quindi ora ci soffermeremo solo nell'analisi delle specificitá 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 é :

$$pvi = g^{svi} \bmod p \tag{28}$$

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

$$hvi = hash(\text{initiator's DHPart2 message} \parallel \text{responder's Hello message}) \tag{29}$$

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 puó 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 invdovinare il valore SAS. Questo fatto implica che il SAS puó essere relativamente corto: un valore a 16 bit, lascerebbe all'attaccante una possibilitá 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 cosí:

$$pvr = g^{svr} \bmod p \tag{30}$$

in maniera analoga a quanto spiegato per l'initiator.

Dopo avere 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 iniettare 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 é andata a buon fine, il responder puó calcolare il proprio valore per l'hash commitment usando la chiave pubblica pvi ricevuto 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 é in corso un attacco MITM e lo scambio di chiavi va terminato.

Il responder puó quindi calcolare il risultato Diffie-Hellman:

$$DHResult = pvi^{svr} \bmod p \tag{31}$$

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 puó quindi inviare DHPart2 e calcolare il risultato Diffie-Hellman:

$$DHResult = pvr^{svi} \bmod p \tag{32}$$

ABPS: Alway Best Packet Switching

Lo scenario é quello di un device mobile, equipaggiato con interfacce wireless eterogenee *(NIC - Network interface cards)*, in grado di sfruttare completamente il vantaggio offerto dalle interfacce multiple a disposizione. Le attuali tecnologie infatti, permettono al device di usare selettivamente solo una *NIC*. In alternativa, il modella *ABPS* estende le capacitá di un device mobile, con piú interfacce di rete, fornendogli la possibilitá di instradare ciascun *datagram*, attraverso l'interfaccia piú opportuna, permettendo in questa maniera un uso simultaneo di tutte le *NIC* disponibili.

Le applicazioni possono quindi creare policy per il *load balancing* e per il *recovery* della connessione, offrendo un supporto effettivo per il servizi multimediali su device mobili.

L'architettura *ABPS*[39] presentata in seguito, fa uso di un server proxy fisso che svolge il ruolo di *relay*, con lo scopo di superare limiti di connettivitá dovuti alla presenza di *firewall* e *NAT*.

Su ogni nodo mobile viene installato un proxy client *SIP* e *RTP*, che mantiene un tunnel *multi-path* con il server proxy, attraverso tutte le interfacce disponibili. Il canale logico cosí stabilito consente, attraverso un'estensione dei protocolli *SIP* e *RTP/RTCP*, che ogni pacchetto possa essere identificato anche quando cambia l'indirizzo IP di provenienza. Questo assicura che le policy di *load-balancing* e di *recovery* possano reagire tempestivamente ad errori di trasmissione o riconfigurazioni della rete, poiché non incorrono nel-

l'overhead introdotto dal classico meccanismo *message/response*, tipico del protocollo *SIP*.

Inoltre, il modello *ABPS* é progettato in modo che applicazioni basate su *SIP* non debbano subire alcuna modifica per utilizzare questa architettura.

Anche il supporto alla mobilitá é garantito utilizzando l'attuale protocollo *IPv4*, permettendo in tal modo una piena compatibilitá con le infrastrutture esistenti.

Lo scenario

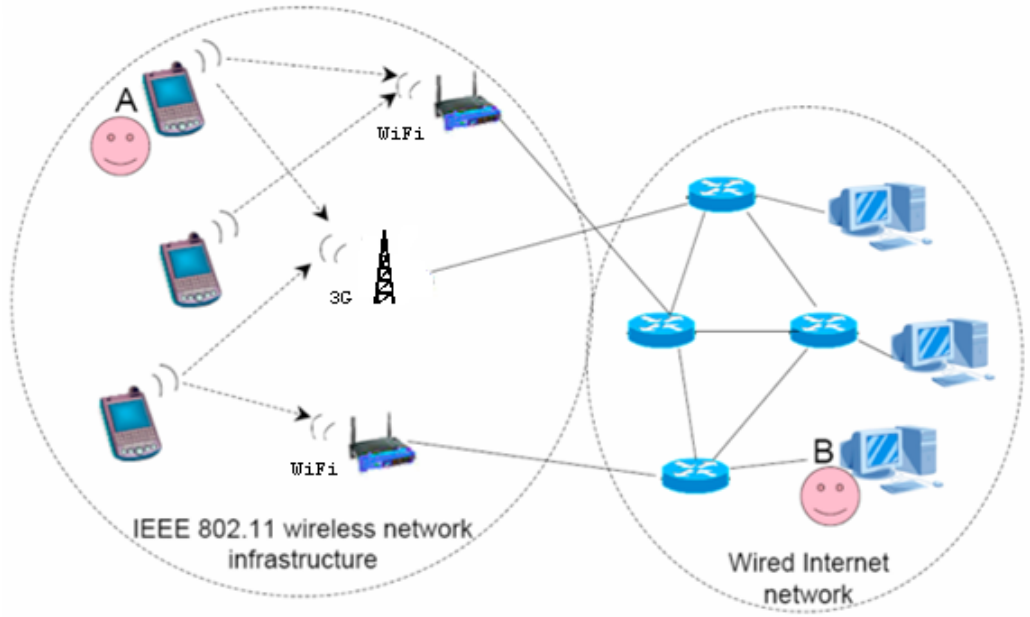
Lo scenario delle telecomunicazioni mobili si sta attualmente evolvendo verso tre direzioni principali:

- creazione di nuove tecnologie wireless a banda larga dedicate a differenti scenari (es Mobile WiMAX, ZigBee)

- la continua espansione della copertura territoriale di tecnologie Wifi (IEEE 802.11a/b/h/n) offerte sia dai provider (es.Boing [4]) che da organizzazioni basate sulla cooperazione fra gli utenti (es.Fonera [5])

- la definizione, da parte del consorzio *3GPP2*, delle specifiche per l'architettura *IMS* per reti 3G, che supporta tutte le applicazioni distribuite basate su IP e promuove lo sviluppo di servizi multimediali basati sui protocolli SIP e RTP.

Un device mobile equipaggiato con *NIC* wireless eterogenne viene chiamato *multi-homed Mobile Node* (MN).



Sfortunatamente, malgrado i numerosi sforzi compiuti a riguardo, esistono ancora una serie di difficoltá di natura tecnica che impediscono di garantire agli utenti mobili un pieno accesso ai servizi multimediali, in particolare per quando riguarda applicazioni *SIP* e *RTP* che necessino di tempi di latenza ridotti per garantire l'interattivitá.

Oltre agli stringenti requisiti di *QoS* per le applicazioni *VoIP*, esposti in precedenza, i sistemi mobili soffrono di alcune difficoltá aggiuntive:

- **Hand-off** : questo termine indica la procedura di selezione e associazione di un client wireless verso un AP, dopo che la connessione con la rete precedente non é piú disponibile a causa di una mancata copertura o di interferenze temporanee nella trasmissione.

- **Costi di servizio e copertura** : sebbene le tecnologie *3G* offrano una copertura decisamente piú affidabile sulla lunga distanza, il loro costo di utilizzo é tendenzialmente maggiore di tecnologie che operano sulla media distanza come *IEEE 802.11*. Queste ultime tuttavia presentano piú difficoltá per garantire connettivitá agli utenti mobili.

- **Firewall e NAT**: la presenza di firewall e l'incremento di sistemi di NAT, nati per sopperire alla crescente diminuzione degli indirizzi IP disponibili, hanno creato numerosi problemi di connettivitá. I NAT in particolare, per essere superati richiedono l'introduzione di tecnologie di supporto come *STUN*[36] e *ICE*[12]. In situazioni di host coperti da NAT simmetrico, é necessaria la presenza di un server d'appoggio pubblicamente raggiungibile per sopperire ai problemi di connettivitá.

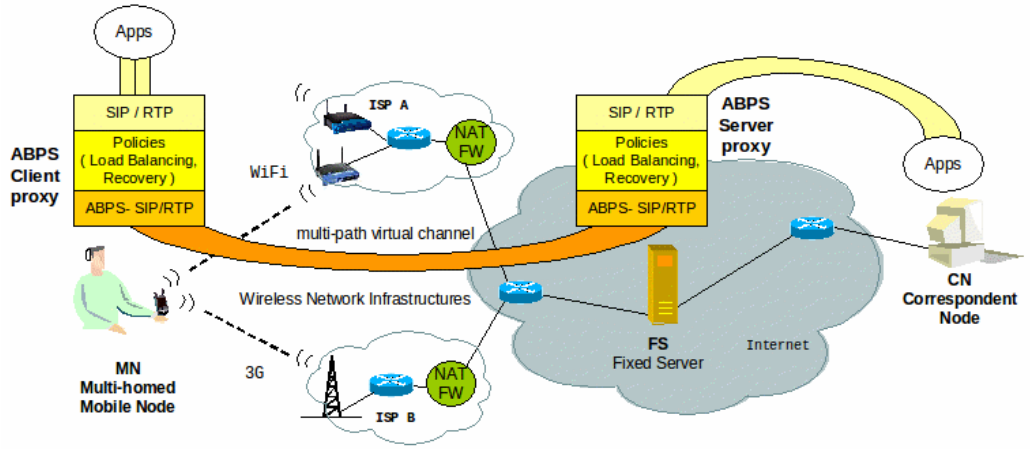
- **Always Best Connected - ABC** : la maggiorparte dei sistemi di gestione della mobilitá seguono questo approccio[19], che suggerisce di utilizzare un canale trasmissivo alla volta, fino a che le prestazioni di questo non degradano eccessivamente. Quando il canale utilizzato non é piú considerato affidabile, si procede alla riassociazione con un altro AP, qualora ce ne sia uno piú performante disponibile.

La soluzione ABPS prende in considerazione tutti questi aspetti, fornendo soluzioni originali e innovative, che riescono a soddisfare i requisiti QoS richiesti per servizi multimediali interattivi su dispositivi mobili, senza richiedere modifiche ai protocolli di livello trasporto esistenti.

Robust Wireless Multi-Path Channel (RWMPC)

RWMPC é un meccanismo *cross-layer*, che segue il modello ABPS per fornire un'effettiva interattivitá e una bassa percentuale di pacchetti persi, soddisfando requisiti QoS per il VoIP. É concepito per essere utilizzato da un host mobile, equipaggiato con almeno due interfacce di rete wireless.

Sfruttando *RWMPC*, l'applicazione puó stabilire e mantenere link wireless con diversi *access-point* garantendo l'accesso a due reti wireless, potenzialmente amministrate da diverse organizzazioni. Permette inoltre di selezionare, per ogni datagram UDP trasmesso, l'interfaccia migliore attraverso cui instradarlo.



Monitor

É il componente dell'architettura *ABPS* che si occupa di monitorare e configurare le interfacce disponibili sul dispositivo.

É un'applicazione separata che configura dinamicamente le interfacce e le regole di routing.

Comunica con l'*ULP* per informarlo sulle interfacce disponibili, che sono state configurate e possono essere utilizzate per l'inoltro dei datagram.

Gli notifica inoltre quando una *NIC* viene disabilitata a causa di un errore di connessione o del degradarsi della qualitá del segnale.

Transmission Error Detector - TED

Come spiegato in precedenza, la percentuale di pacchetti persi é un fattore discriminante per il rispetto dei requisiti *QoS* nei sistemi *VoIP*. L'utilizzo di un canale di comunicazione wireless, comporta un aumento intrinseco dei pacchetti persi, dovuto all'instabilitá di questo tipo trasmissione rispetto ai classici collegamenti cablati. Le cause principali del degrado della qualitá della trasmissione in un canale wireless sono:

- **Interferenze Ambientali**: le onde prodotte da un device wireless si propagano lungo le tre dimensioni dello spazio, seguendo le leggi della

fisica. Gli ostacoli che incontrano lungo il percorso, a seconda della forma e del materiale di cui sono costituiti, producono fenomeni di assorbimento o riflessioni che disurbano la trasmissione.

- **Interferenze Trasmissive**: altre sorgenti trasmissive presenti nelle vicinanze del dispositivo mobile possono interferire con la comunicazione. (es.forni microonde, dispositivi controllo a distanza come cancelli automatici, allarmi auto, etc)

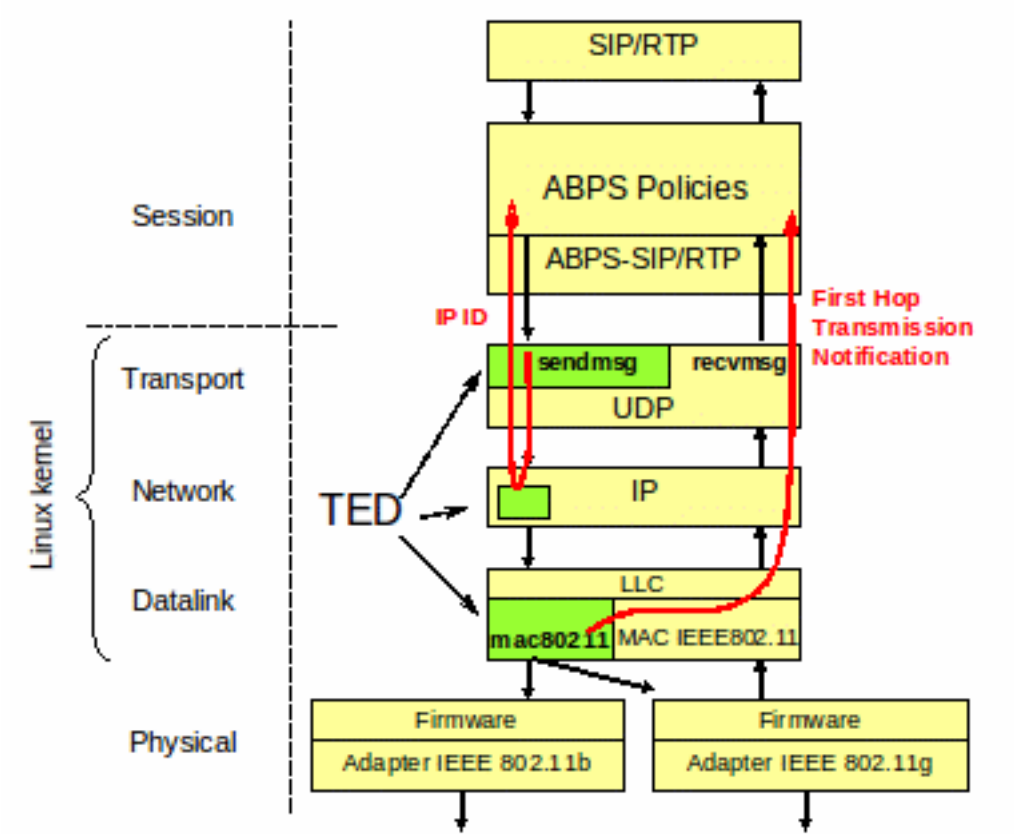
- **Mezzo Condiviso**: i vari terminali wireless condividono lo stesso mezzo trasmissivo, cioé l'aria. Devono quindi essere applicate politiche di *multiplexing*, per alternare e regolamentare l'accesso a tale mezzo. Una concentrazione di terminali in uno spazio ristretto, comporta una diminuzione della banda disponibile. Un normale *AP* sopporta circa dieci client che effettuano trasmissioni *VoIP* contemporaneamente.

La ritrasmissione di un datagram andato perduto non é sempre possibile in un sistema interattivo. Se la notifica di mancata ricezione di un pacchetto, arrivasse dall'host destinatario, i tempi di latenza tra i due end-point impedirebbero comunque che il datagram reinviato, possa giungere a destinazione entro i 150 ms necessari.

TED é un meccanismo *cross-layer*, che sfrutta gli *ACK* ricevuti a livello *MAC*, per notificare all'applicazione informazioni sulla perdita di un pacchetto durante il primo *hop*, cioé nel percorso tra l'host mobile e l'*AP*.

Nel protocollo *IEEE 802.11*, il destinatario di una frame a livello *MAC*, spedisce al mittente una notifica di avvenuta ricezione. Quando il frame trasmesso viene perduto, oppure l'ack non viene ricevuto dal mittente, quest'ultimo continua a ritrasmetterlo fintanto che non viene superato un numero limite di tentativi. Dopo questo limite, il livello MAC scarta il frame senza ulteriori notifiche.

Il componente *TED* si occupa di monitorare l'avvenuta trasmissione di un frame, comunicando questa informazione all'applicazione *ULP*.



UDP Load Balancer - ULP

Questo componente dell'architettura *ABPS*, riceve dall'applicazione i pacchetti UDP, li incapsula all'interno di datagram UDP, e li trasmette all'endpoint destinatario. Usa un socket per ogni interfaccia wireless attiva, della cui configurazione si é occupato il *Monitor*.

Riceve 3 tipi di notifiche:

- **Notifiche dal Monitor**: ULB viene informato sulla disponibilitá di una nuova interfaccia o sulla disabilitazione di un'interfaccia precedentemente disponibile. Si comporta di conseguenza, bindando un nuovo socket o chiudendone uno associato ad un'interfaccia non piú diponibile, e considerando perduti tutti i frame trasmessi attraverso di essa che

non hanno ricevuto un notifica First-hop Trasmission Notification dal TED.

- **Notifiche ICMP**: informano l'*ULP* sulla perdita di frame lungo il percorso end-to-end.
- **Notifiche dal TED**: segnalano l'avvenuta trasmissione o perdita di un frame lungo il primo hop.

Basandosi su questi tre tipi di informazioni, l'*ULB* decide se un datagram *UDP* perso debba essere ristrasmesso usando un'altra interfaccia, oppure debba essere scartato definitivamente. Inoltre, monitorando le notifiche ricevute dal *TED*, puó selezionare l'interfaccia da usare per trasmettere il frame successivo e sviluppare politiche di *load-balancing*.

L'architettura ABPS

L'architettura *ABPS-SIP/RTP* proposta, é stata progettata seguendo tre linee guida fondamentali:

- **Servizi esterni per la continuitá della comunicazione**: l'architettura proposta richiede la presenza di un servizio di ancoraggio esterno, chiamato server *ABPS*. É un servizio esterno, indipendente dalla rete d'accesso, che viene ospitato da un server fisso (*FS*), raggiungibile pubblicamente attraverso un indirizzo IP statico. Il server deve essere posizionato al di fuori di qualsiasi firewall o sistema di *NAT*. Il servizio di ancoraggio é costituito da un proxy *SIP* e *RTP*, opportunamente esteso, che agisce da *relay* intermediario nella comunicazione tra un *MN (Mobile Node)* e il destinatario corrispondente *CN (Correspondent Node)*, in maniera tale da garantire la comunicazione senza soluzioni di continuitá.

  Nello specifico il server nasconde la posizione corrente di *MN*, riscrivendo i messaggi di livello sessione(*SIP*). Dal punto di vista del *CN*,

il server *ABPS* appare essere il *MN*. Ogni *MN* é equipaggiato con un *proxy SIP/RTP* opportunamente esteso(chiamato *ABPS client*), che mantiene un canale di comunicazione *multi-path* con il server ABPS, in maniera tale da poter superare la presenza di *NAT* e *firewall*.

- **Uso simultaneo di tutte le NIC**: l'achitettura ABPS prevede che le applicazioni del *MN* possano usare contemporaneamente tutte le interfacce disponibili, differenziando la scelta di quale NIC usare per ogni datagram trasmesso, e introducendo la possibilitá di scegliere quella piú opportuna a seconda delle caratteristiche del *datagram* stesso.

  Questa scelta permette al client e al server *ABPS* di collaborare, per implementare policy di *load-balancing* e di recupero della connessione, al fine di massimizzare la banda disponibile e minimizzare sia la percentuale di pacchetti persi che il costo economico.

- **Signalling short-cuts**: client e server *ABPS* comunicano tra loro utlizzando estensioni dei protocolli *SIP* ed *RTP*, che utilizzano *short-cut* dei normali messaggi di *signalling*. L'intento é quello di ridurre il ritardo prima che una comunicazione possa continuare, dopo che é avvenuto un cambiamento nei parametri relativi ad essa. Queste estensioni definiscono un *User Identifier (UID)* e un *RTP flow identifier (FID)*, che identificano in maniera univoca rispettivavente l'utente del mobile node (*UID*) e ogni flusso *RTP* tra il *MN* e il *CN*. queste esetensioni introducono, all'interno dei pacchetti *SIP* e *RTP*, i campi necessari per idenficare univocamente e in sicureza il mittente di un determinato messaggio, anche qualora cambi l'indirizzo IP sorgente ad esso associato. In questa maniera il client *ABPS* puó cambiare l'interfaccia attraverso cui inoltrare un determinato datagram, senza usare un messaggio SIP preliminare per informare il server dell'avvenuto cambio di indirizzo IP. É l'estensione a livello applicativo del principio *ABPS*.

Per utilizzare il servizio di continuitá offerto dall'architettura *ABPS*, un *MN* deve seguire i seguenti passi:

- **Configurazione off-line**: una configurazione preliminare effettuata attraverso una registrazione off-line, permette al *MNU (Mobile Node User)* e al *MN* di sfruttare la continuitá di servizio, ottenendo sia l'*UI (User Indentifier)* che la chiave pre-condivisa con l'*ABPS server*.

- **Configurazione del contesto di sicurezza**: quando il *MN* viene avviato e configura le sue *NIC*, il client costruisce un contesto temporaneo di sicurezza col server *ABPS*, sfruttando la chiave pre-condivisa. In questa maniera viene generata una chiave di sessione condivisa, senza che sia richiesta la trasmissione della chiave pre-condivisa attraverso la rete, preservando in tale maniera i requisiti di sicurezza. Dopo questa fase, il *MN* e il *FS* possiedono tutte le informazioni necessarie per costruire e autenticare i messaggi del protocollo *ABPS-SIP/RTP*. In particolare, il server *ABPS* é in grado di identificare messaggi *SIP* e *RTP* di un client, anche dopo che questo ha cambiato indirizzo IP di tutte le *NIC*.

- **Setup/Update de canale Multi-path**: il client comunica al server *ABPS* le *NIC* attive sul *MN*, attraverso un messaggio *REGISTER ABPS-SIP*, che include tutti gli indirizzi IP delle interfacce attive. Quando il *MN* cambia la configurazione di un'interfaccia, il client *ABPS* informa il server attraverso un messaggio *UPDATE ABPS-SIP*. Tutti i messaggi *ABPS-SIP/RTP* includono un numero di sequenza, utile per scartare i messaggi giunti in ritardo o in disordine.

- **Setup del flusso RTP tra MN e CN**: per iniziare una trasmissione *RTP* con un *CN*, l'applicazione sul MN apre una porta UDP per ricevere i messaggi RTP e inoltra, attraverso il client *ABPS*, un messaggio *INVITE* al server con destinazione il *CN*. Il messaggio include un header *SDP* attraverso cui specifica il numero di porta. Il server *ABPS* provvederá ad inoltrare il messaggio verso il *CN*. Sia client che server, prima di trasmettere un messaggio all'entitá specificata, eseguono le seguenti tre operazioni:

  - aprono una porta udp per ricevere i messaggi *RTP* dell'entitá con cui sono in comunicazione.
  - inseriscono nel messaggio *SIP* il proprio indirizzo IP nel campo mittente.
  - inseriscono nel messaggio *SIP* il numero di porta che é stato aperto.

  Dal punto di vista del *CN*, il messaggio INVITE sembra provenire dal server *ABPS*. I successivi messaggi (sia i *SIP Response* che i pacchetti *RTP*) verrano inviati dal *CN* verso il server *ABPS*. Il server *ABPS* inoltra i messaggi verso il client, e quest ultimo verso l'applicazione sul *MN*. Dopo questa fase di configurazion, il flusso dati *RTP* tra *MN* e *CN* segue tre percorsi *end-to-end* bidirezionali e consecutivi:

  - il percorso tra *MN* e il client *ABPS* usando il protocollo *RTP*.
  - il percorso tra client e server ABPS usando le estensioni dei protocolli *SIP* e *RTP*.
  - il percorso tra il server *ABPS* e il *CN* usando *RTP*.

- **Policy QoE (Quality of Experience) per il flusso RTP**: i flusso dati *RTP* tra client e server *ABPS* é sotto il controllo di un sottolivello di policy, gestito da entrambe le entitá. I sottolivelli di policy usano le estensioni *ABPS* di *SIP* e *RTP*, implementando meccanismi di *QoS* come:

  - selezione della *NIC* da utilizzare per la trasmissione, sulla base del pacchetto da inviare.
  - rivelazione dei pacchetti persi.
  - ritrasmissione tempestiva.
  - load-balancing.

  Differenti insiemi di meccanismi di *QoS*, danno vita a differenti canali *multi-path* virtuali che soddisfano diversi requisiti di *QoS*, ad esempio:

  - interattivitá (rivelazione pacchetti persi/ritrasmissione tempestiva)
  - consumo di banda (selezione *NIC packet-based/load-balancing*)

  Il sottolivello che si occupa delle policy, implementa questi canali virtuali *multi-path* con *QoS* rafforzata, e rappresenta ognuno di essi, all'applicazione sul *MN*, attraverso una coppia di porte *SIP* e *RTP*. In questa maniera ogni applicazione puó scegliere il canale virtuale desiderato semplicemente scegliendo la porta di uscita desiderata, consentendo al *MN* di interagire con l'architettura *ABPS* senza dover subire modifiche.

- **Scorciatoie per l'aggiornamento dei canali Multi-path**: dopo la fase di configurazione, il flusso dati *RTP* attraverso il canale *multi-path* é sotto il controllo del sottolivello delle policy. I messaggi *SIP* gestiscono il flusso *RTP* assumendo il fatto che gli *end-point* coinvolti mantengano i loro indirizzi IP. Nel nostro contesto possono accadere due eventi che invalidano la precedente assunzione:

  - il *MN* cambia l'indirizzo IP di una *NIC*, ad esempio in seguito ad un avvenuto *hand-off*.
  - il sottolivello di policy decide di trasmettere un messaggio *RTP* attraverso una diversa *NIC*.

  In entrambi i casi, l'approccio *SIP* convenzionale richiederebbe lo scambio di un messagio *REINVITE* per rinegoziare i parametri della comunicazione. Questo scambio request/response introduce un ritardo aggiuntivo, proporzionale al *RTT (Round Trip Time)* tra i due *end-point*, prima che possa essere spedito il successivo pacchetto *RTP*. É probabile che in questa maniera il datagram raggiunga la destinazione troppo tardi, violando i requisiti QoS di interattivitá. Questo meccanismo inoltre riduce l'effettivitá di ogni policy di ritrasmissione. L'approccio *ABPS*

invece, mira a rimuovere ogni ritardo addizzionale causato dai messaggi *REINVITE*. L'estensione del protocollo *RTP*, include nei pacchetti stessi le informazioni necessarie per continuare la comunicazione, anche dopo che il mittente ha cambiato indirizzo IP. Per soddisfare questa esigenza, vengono inclusi nei *datagram RTP* i seguenti campi:

- **FID (Flow Identifier)**:identifica univocamente il flusso *RTP* tra *MN* e *CN*, al quale il pacchetto appartiene.
- **Firma Digitale**: per garantire la proprietá di autenticazione ai datagram, evitando contraffazioni.

Inoltre, dopo che il mittente é stato identificato, il server *ABPS* analizza l'indirizzo IP sorgente espresso attraverso il protocollo IP e la porta *UDP* del mittente. In questa maniera, ogni pacchetto *RTP* trasporta sia i dati che un insieme ristretto di informazioni di *signalling* , permettendo l'instradamento attraverso una qualsiasi delle *NIC* del *MN*. In aggiunta a ció, il *datagram ABPS-RTP* puó includere la lista completa di tutti gli indirizzi IP delle *NIC* disponibili sul dispositivo.

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 proprietá di autenticazione, integritá e, opzionalmente, confidenzialitá 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, poiché le applicazioni multimediali scambiano un numero elevato di piccoli *datagram RTP*, é essenziale limitare sia la dimensione dei dati trasmessi, che il costo computazionale per i processi di cifratura e decifratura.

Estensioni ABPS al protocollo SIP

ABPS-SIP é un'estensione del protocollo SIP che permettte di stabilire un canale di comunicazione sicuro *SIP*, tra le due estremitá 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[14], la chiave pre-condivisa é 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:

- **UID - User identifier** : 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'integritá.

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

Estensioni ABPS al protocollo RTP

ABPS-RTP é 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 proprietá di autenticazione, integritá e confidenzialitá al flusso dati *RTP*, prescindendo dall'indirizzo IP del mittente. Il destinatario di un messaggio é in grado di distingure

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)[8]. STRP* aggiunge un header con un codice di autenticazione, calcolato usando una chiave di cifratura associata a un singolo flusso *RTP*.

Prima di poter configurare un flusso *SRTP* tra due end-point, é 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* é motivata dalle sue prestazioni, oltre che dalle forti garanzie che offre in termini di sicurezza.

Sicurezza e Autenticazione

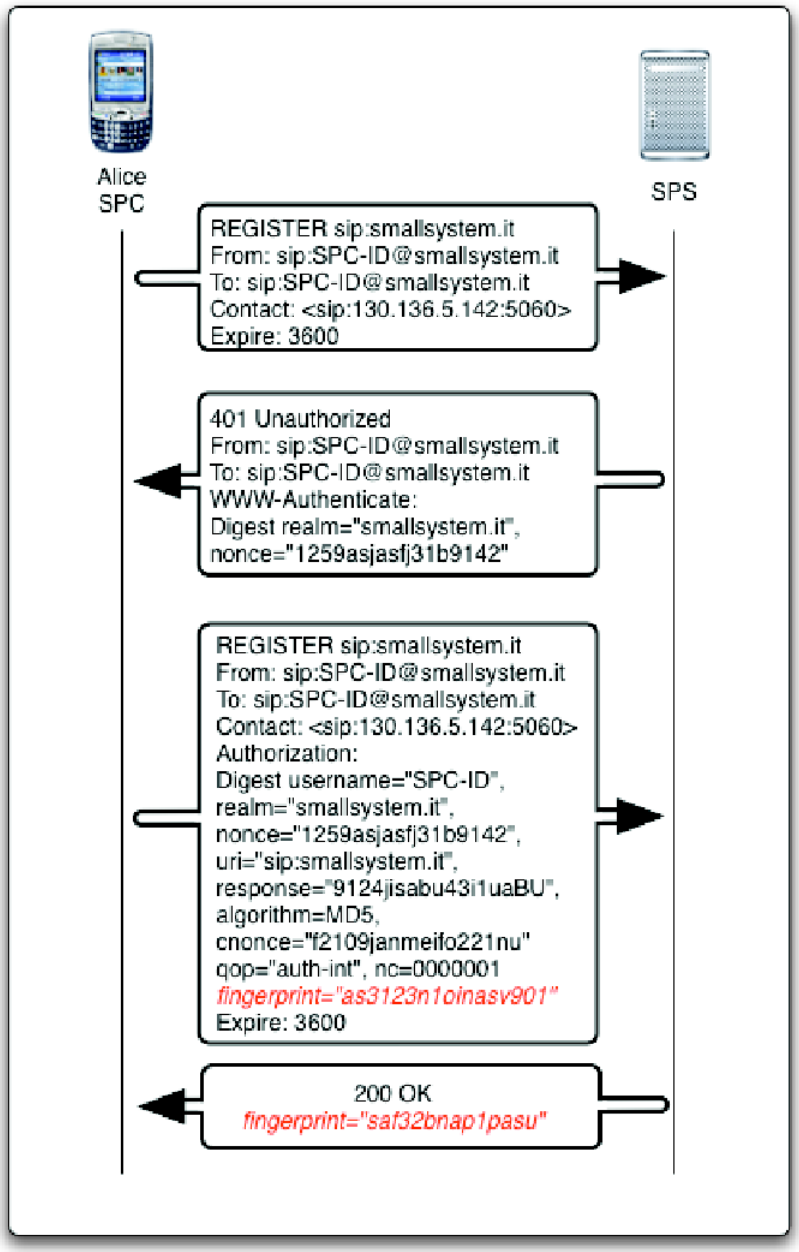
Oltre alle estensioni ai protocolli *SIP* e *RTP* descritte in precedenza, l'architettura ABPS fornisce alcune innovazioni a supporto della sicurezza. In particolare viene esteso il meccanismo di *challenge-response* descritto in precedenza, e viene utilizzato il protocollo *ZRTP* per lo scambio di chiavi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

Autenticazione Challenge-Response

L'autenticazione challenge/response progettata per *ABPS*, e richiesta dal server proxy *ABPS*, é un'estensione dell'autenticazione *SIP/HTTP Digest* descritta in precedenza, con opportuni cambiamenti che garantiscono l'autenticazione mutuale e l'integritá 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 suo *fingerprint* calcolato con l'algoritmo *SHA-1*, che ne garantisce l'autenticitá.



Di seguito viene esposta nel dettaglio la procedura di autenticazione

ABPS.

- **Fase 1 - Richiesta Iniziale**: il *SIP User Agent Client (UAC)* invia una richiesta (e.g. *REGISTER*), per la quale é necessaria l'autenticazione, al server (es. il *SIP Registrar*). (Identico a *SIP/HTTP Digest*)
- **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*)
  - calcola l'hash del messaggio inviato attraverso l'algoritmo *SHA-1* e lo salva in memoria. Verrá utilizzato nella fase 4:

$$\text{hash\_message\_challenge\_server } = SHA-1(message\_challenge) \tag{33}$$

- **Fase 3 - Response**:
  - il client calcola l'hash del messaggio ricevuto contenente il challenge, e lo salva per utilizzarlo in seguito:

$$\text{hash\_message\_challenge\_client} = SHA-1(message\_challenge) \tag{34}$$

  - 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

$$\text{hash\_message\_response\_client} = SHA-1(message\_response) \tag{35}$$

– calcola una chiave temporanea, basata sulla *master_key* e su alcune credenziali del messaggio response:

$$\text{masq\_master\_key} = SHA{-}1(''20 :'' \| \; master\_key \; \|'':'' \| \; cnonce \; \|'':'' \| \; nonce) \tag{36}$$

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

$$\text{fingerprint} = HMAC{-}SHA{-}1(message\_response, masq\_master\_key) \tag{37}$$

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

- **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

$$\text{masq\_master\_key} = SHA{-}1(''20 :'' \| \; master\_key \; \|'':'' \; \| \; cnonce \; \|'':'' \| \; nonce) \tag{38}$$

  – Calcola il *fingerprint* del messaggio ricevuto contenente il response:

$$\text{fingerprint} = HMAC{-}SHA{-}1(message\_response, masq\_master\_key) \tag{39}$$

- 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 é andata a buon fine, crea un messaggio di acknowledgement.
- Calcola la chiave di sessione,utile per l'autenticazione dei messaggi:

$$\text{session\_key\_ia} = HMAC - SHA - 1((hash\_message\_response\_client \| ":" \| hash\_message\_challenge\_client \| 0x00), masq\_master\_key) \quad (40)$$

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

$$\text{fingerprint} = HMAC - SHA - 1(message\_200\_OK, session\_key\_ia) \quad (41)$$

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

- **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):

$$\text{session\_key\_ia} = HMAC - SHA - 1((hash\_message\_response\_client \| ":" \| hash\_message\_challenge\_client \| 0x00), masq\_master\_key) \quad (42)$$

– Calcola il *fingerprint* del messaggio ricevuto con la chiave di sessione appena calcolata:

$$\text{fingerprint} = HMAC{-}SHA{-}1(message\_200\_OK, session\_key\_ia) \tag{43}$$

– 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 puó 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 realtá il response non dovrebbe essere mai calcolato usando direttamente la precondivisa tra client e server *ABPS*, ma piuttosto una chiave derivata da questa, utilizzando una *Key Derivation Function*, in conformitá alle direttive espresse in [14].

Sicurezza del traffico RTP

Premessa

L'architettura *ABPS*, come spiegato in precedenza, permette di realizzare una 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 continuitá 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, é 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 a uno scenario in cui un client cambi punto di ancoraggio alla rete.

Va inoltre considerato il fatto che l'architettura *ABPS* é 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 é all'oscuro dell'esistenza del protocollo *ABPS* e comunica con il primo come se esso fosse il proxy server *ABPS*, cioé 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 é, 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 modalitá Diffie-Hellman.

Per garantire che la negoziazione delle chiavi avvenga in modalitá 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:

- Genera un hash del messaggio *ZRTP Hello* che verrá spedito in seguito.

- 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 é protetto da manomissioni grazie al fingerprint allegato, calcolato grazie alla chiave di sessione precedentemente negoziata durante l'autenticazione *challenge-response ABPS*.

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

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

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

Symbian: il sistema operativo Nokia

Symbian OS, é uno dei sistemi operativi Nokia per cellulari e smartphone, nato come figlio del sistema *EPOC* di Psion.

La piattaforma Symbian é stata creata grazie alla fusione e integrazione tra diverse tecnologie, in particolare il *core* del sistema chiamato *S60* e parti delle interfacce utente UIQ e MOAP(S).

Il progetto é stato realizzato in particolare grazie al contributo di societá come Nokia, NTT DoCoMo, Sony Ericsson e Symbian Ltd.

Nel 2008, la societá Symbian Software Limited fu acquistata da Nokia e successivamente trasformata in un'organizzazione non-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, é stato rilasciato completamente sotto licenza *EPL* solo nel febbraio 2010.

Caratteristiche Tecniche

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

Il kernel attualmento in uso é conosciuto sotto il nome di *EKA2 (EPOC Kernel Architecture 2)*.

Le sue 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

Una delle caratteristiche piú importanti di *EKA2* é la capacitá di eseguire autonomamente diversi *signalling stack* completi. Le versioni precedenti del sistema operativo necessitavano di una CPU dedicata per svolgere tale compito.

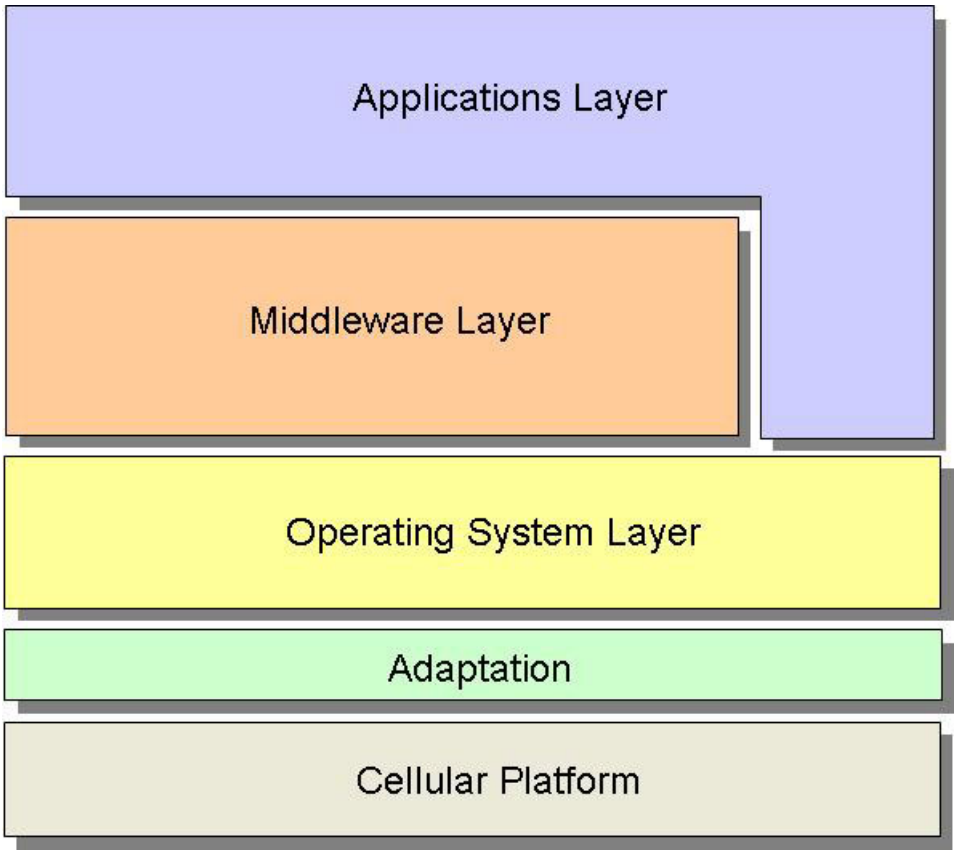
Questi *stack* sono componenti particolarmente complessi, difficilmente riscrivibili per essere supportati nativamente da Symbian OS. EKA2 risolve questa problematica introducendo il concetto di *personality layer* per emulare le primitive di base di altri sistemi operativi, permettendo l'utilizzo di *signalling stack* esistenti, pressoché senza modifiche.

L'architettura della piattaforma Symbian é suddivisa in diversi livelli:

- **Applications Layer**: é composto da specifiche *UI (User Interface)* e componenti dell'*engine* delle applicazioni. Utilizza i servizi forniti dal livello middleware e dal sistema operativo.
- **Middleware Layer**:

  fornisce servizi al livelllo applicativo come: servizi multimediali, networking e servizi di locazione.

- **Operating System Layer**: fornisce tutti i servizi di alto livello del sistema operativo attraverso una collezione completa di *technology domain* come quello di comunicazioni, networking, grafica, multimedia, etc ... Include inoltre una serie di servizi di basso livello come framework, librerie e utility, che trasformano le astrazioni hardware e i meccanismi dell'OS in un'interfaccia programmabile.
- **Adaption Layer**: integra il software generico con la piattaforma del telefono cellulare. Viene implementato dai creatori degli specifici device attraverso, anche se un'implementazione di riferimento viene fornita con la piattaforma stessa.
- **Cellular Platform**: é il livello hardware specifico dei device e del software che fornisce i servizi richiesti dalla piattaforma Symbian.



Si noti come ogni livello possa essere composto da pacchetti appartenenti a qualsiasi *technology domain*.

I *Technology domain* sono gruppi di pacchetti, ognuno di quali é una collezione di componenti. Ogni dominio segue una propria *roadmap* per lo sviluppo, curata da manager appartenenti alla Symbian Foundation.

Ogni pacchetto appartiene esattamente a un dominio, a seconda delle funzioni di cui si occupa. Attraverso questo tipo di suddivisione dello sviluppo, la Symbian Foundation cerca di incoraggiare la formazione di una forte comunitá di sviluppo intorno alle singole aree di interesse, generando discussioni e rivisitazioni intorno al software prodotto.

Ogni pacchetto é mantenuto da una persona appartenente a un'organizzazione membro della Symbian Foundation, che ha il compito di esaminare e organizzare i contributi al codice provenienti dalla comunitá.

Licenza

L'attuale licenza sotto cui é rilasciato Symbian OS é EPL (Eclipse Public License). Questa licenza é approvata sia da OSI (Open Source Initiative) che da FSF (Free Software Foundation)

Progettazione e Sviluppo

Strumenti e Librerie di Supporto allo Sviluppo

Libzrtp

É la libreria che implementa le funzioni di *ZRTP*, sviluppata dall'autore del protocollo stesso, Philip Zimmermann, e utilizzata all'interno dello ZPhone Project (http://zfoneproject.com/).

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

- Linux
- Windows
- Symbian OS

Uno degli scopi di questa tesi é stato quello di integrare questa libreria all'interno di PJSIP, per permettere a quest'ultima di acquisire le funzioni di gestione del protocollo ZRTP.

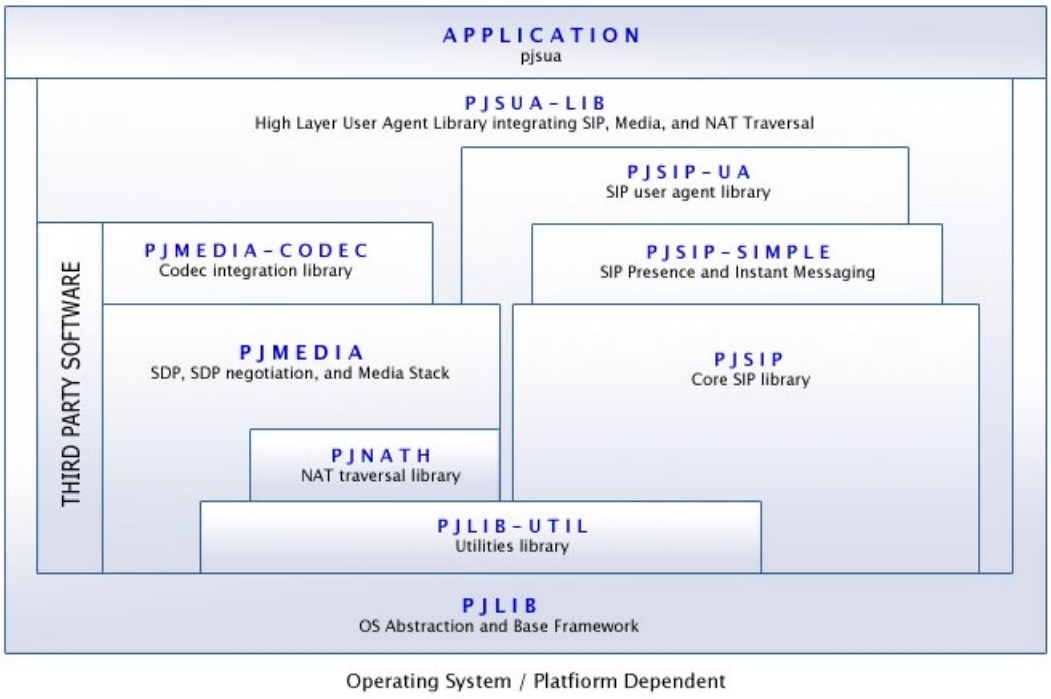
Pjsip

PJSIP é 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 é rilasciato sotto licenza GPL 2.0.

- **Elevate prestazioni**: é 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 piú potente.
- **Dimensioni Contenute/Scalabilitá**: é in grado di scalare verso il basso per essere utilizzata con device a basse prestazioni e ridotte risorse di memoria, cosí come sfruttare le potenzialitá offerte da server multiprocessore. Il tutto usando lo stesso stack SIP.
- **Portabilitá**: funziona su architetture a 32 bit, 64bit, big endian e little endian. É portabile su praticamente ogni sistema operativo esistente.
- **Documentazione Esaustiva**: é dotata di una documentazione esaustiva, sia generata a partire dai sorgenti, che corredata di articoli scritti a mano con spiegazioni articolate sull'architettura.



La libreria PJSIP ha molte caratteristiche interessanti. Un'esposizione completa e dettagliata di queste, esula dallo scopo della tesi. Di seguito verranno accennate solo alcune delle *feature* ritenute piú interessanti, in particolar modo quelle che hanno avuto rilevanza in relazione al lavoro implementativo.

Memory Pool Management

Viene introdotto il concetto di *Memory Pool*, che rappresenta un gestore centralizzato per la memoria dinamica utilizzata dall'applicazione. Le *Memory Pool* permettono operazioni di allocazione dinamica della memoria equiparabili alla classica funzione *malloc* (C) o all'operatore *new* (C++). L'utilizzo dei classici operatori infatti, non é particolarmente consigliabile all'interno di applicazioni *real-time*, poiché costituiscono un collo di bottiglia per le prestazioni e causano problemi di frammentazione.

Le caratteristiche principali del sistema di *Memory Pool* di PJSIP sono:

- a differenza di altre implementazioni, permette l'allocazione di blocchi di memoria di dimensioni variabili.

- é molto efficiente: non solo l'allocazione ha un costo computazionale pari a $O(1)$, ma é anche un'operazione molto semplice, che richiede solo alcune operazioni di aritmetica dei puntatori e non necessita dell'acquisizione di nessun *mutex*.

- efficienza nell'utilizzo della memoria: non viene tenuta traccia dei singoli blocchi allocati dall'applicazione, evitando l'introduzione di *overhead*.

- prevenzione dei *memory leak*: ogni *Memory Pool* possiede delle funzioni di *Garbage collection*. Al momento della distruzione di una pool, tutti i blocchi allocati vengono automaticamente rilasciati.

- i *memory leak* sono facilmente tracciabili, poiché ogni pool possiede un nome e l'applicazione puó ispezionare quali pool sono attualmente attive nel sistema.

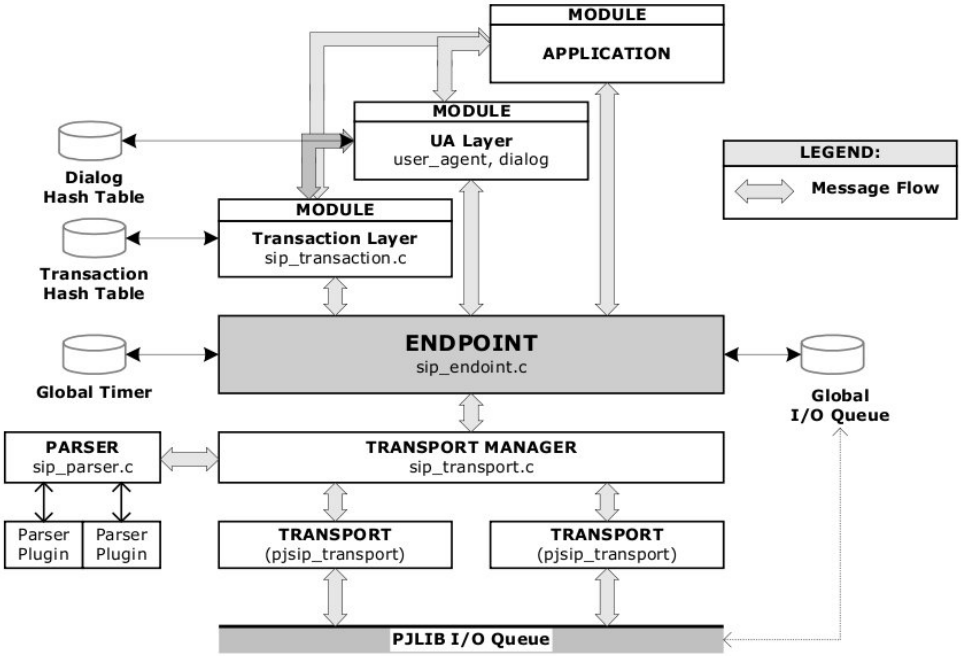
- l'allocazione della memoria in una pool non é *thread-safe*: si assume che ogni pool sia gestita da qualche oggetto di livello piú alto, che ha il compito di assicurare che le operazioni avvengano in sicurezza. In questa maniera le operazioni di de/allocazione possono essere efficienti evitando operazioni di *locking* non necessarie.

- il comportamento di default prevede che le API della pool, come l'operatore *new* del C++, sollevino un'eccezione di tipo PJ_NO_MEMORY_EXCEPTION qualora l'allocazione di un blocco di memoria fallisca. Questa caratteristica (eventualmente disattivabile), permette che la gestione di errori di allocazione della memoria possa essere effettuata da componenti di piú alto livello presenti nell'applicazione.

Tipicamente ogni modulo utilizza una proprio *Memory Pool* e, quando questo ha terminato il proprio compito e viene rimosso dal sistema, la elimina rilasciando con essa tutte le risorse precedentemente allocate.

Architettura Modulare

L'architettura della libreria PJSIP é altamente modulare e strutturata a livelli, sia per quanto riguarda la parte che implementa le funzioni dello *stack SIP*, sia per la parte relativa ai moduli deputati al trasporto dei messaggi.



Il *core* della libreria é rappresentato dal *SIP endpoint* (sip_endpoint.c). Alcuni dei compiti di cui si occupa sono:

- gestisce una *pool factory* e alloca le *pool* associate ai vari moduli *SIP*.
- si occupa della temporizzazione, e schedula eventi che notificherá ai relativi moduli *SIP* interessati.
- gestisce le varie istanze dei moduli di trasporto e controlla il parsing dei messaggi e la stampa.
- gestisce i moduli PJSIP, che sono la struttura primaria attraverso cui estendere la libreria per fornire nuove funzionalitá di parsing e trasporto.
- riceve messaggi dal transport manager e li ridistribuisce attraverso i moduli che implementano i livelli dello stack *SIP*.

La libreria mette a disposizione un'implementazione completa di un classico *stack SIP*, suddivisa in livelli che vengono identificati ciascuno da un valore che ne indica la prioritá. Un messaggio viene passato a tutti i moduli registrati presso il *SIP endpoint*: se si tratta di un messaggio in arrivo questo attraversa i moduli a partire da quello con prioritá maggiore, viceversa per quanto riguarda i messaggi in uscita.

Di seguito i livelli standard in ordinde di prioritá:

- **PJSIP_MOD_PRIORITY_TRANSPORT_LAYER**:
  rappresenta il livello di prioritá dei moduli di trasporto. É il livello piú basso che ha a che fare direttamente con il livello di rete e implementa la trasmissione dati vera e propria (UDP, TCP,...).

- **PJSIP_MOD_PRIORITY_TSX_LAYER**:
  é il livello usato dai moduli relativi alle transazioni, che gestiscono tutti i messaggi in entrata o uscita che appartengono a una determinata transazione *SIP*.

- **PJSIP_MOD_PRIORITY_UA_PROXY_LAYER**:
  é il livello usato da un *UA* o da un proxy. Gestisce tutti i messaggi appartenenti ai *SIP Dialog* esistenti.

- **PJSIP_MOD_PRIORITY_DIALOG_USAGE**:
  é il livello dedicato all'applicazione che avvia un determinato *SIP Dialog*. Gestisce tutti i messaggi indirizzati o provenienti da un Dialog appartenente a una determinata sessione.

- **PJSIP_MOD_PRIORITY_APPLICATION**:
  é il livello dedicato all'applicazione vera e propria, che vuole utilizzare tutti i livelli sottostanti per creare un vero e proprio client *VoIP*.

Lo stack *SIP* é personalizzabile, creando un nuovo modulo che implementi le funzioni di interfaccia definite nel file sip_module.h. Il modulo é posizionabile all'interno della struttura a livelli selezionando adeguatamente il livello di prioritá desiderato.

Network Tools

Come strumenti di supporto allo sviluppo e al debug, ci si é affidati ai seguenti tool opensource:

- **Wireshark e Tcpdump**: software open-source per lo sniffing, utilizzati per l'analisi e il debug dei protocolli di rete utilizzati.
- **Netcat**: utility che permette di leggere e scrivere su socket TCP e UDP in maniera similare al celebre comando unix cat. In presenza di messaggi di protocollo sempre diversi l'uno dall'altro a causa della presenza di valori pseudo-casuali, é stato molto utile per riprodurre situazioni di errore.

Implementazione del Sottosistema di Autenticazione e Cifratura

Introduzione

Il lavoro implementativo ha riguardato in particolare il client Symbian e la libreria PJSIP con cui esso é sviluppato.

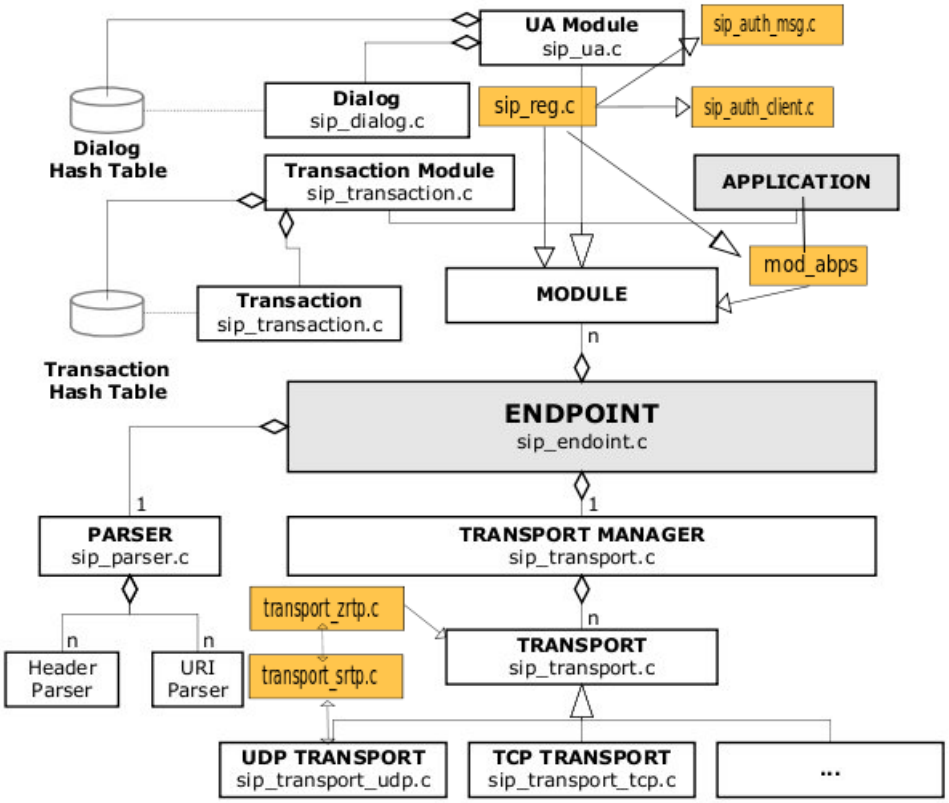
Questa libreria, come descritto in precedenza, ha una composizione modulare che permette estensioni e personalizzazioni della stessa, nel rispetto della propria struttura. L'implementazione per il sistema operativo Symbian dell'architettura *ABPS* tuttavia, non ha una struttura completamente definita,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

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

Si noti che al momento della stesura di questo elaborato, PJSIP non offre un supporto al protocollo ZRTP. Vi é tuttavia la societá KHAMSA SA (http://www.khamsa.ch) che é attualmente impegnata nell'estensione della libreria per offrire le funzionalitá richieste dal protocollo ZRTP. Successive versioni del client potranno quindi avvantaggiarsi di un'implementazione piú organica del protocollo, integrata all'interno di PJSIP stessa.

Autenticazione e Cifratura ABPS all'interno di PJSIP

Questa sezione si occupa di descrivere i moduli principali di PJSIP, coinvolti nel sottosistema di autenticazione e cifratura dell'architettura ABPS. Nella figura che segue sono evidenziati le componenti della libreria che svolgono tale compito:



Client ABPS in Symbian

Come accennato in precedenza, allo stato attuale non esiste alcuna implementazione completa del client *ABPS* per Symbian OS.

Manca ad esempio la parte che si occupa della gestione del *multi-homing*; il modulo di autenticazione era parzialmente implementato, ma soffriva di numerosi problemi che verranno trattati nell'apposita sezione.

Symbian OS stesso presenta molte differenze dai sistemi operativi piú comuni come Linux e Windows, in particolare per quanto riguarda la gestione della rete e il supporto per i thread.

Inoltre, il codice sorgente é stato rilasciato da poco e sono scarsi gli strumenti di supporto allo sviluppo: l'SDK Nokia (System Developement Kit), funziona solo in ambiente Windows cosí come il simulatore, che é afflitto da performance scarsissime e offre il supporto per una sola interfaccia di rete.

Nello sviluppo per il sistema operativo Symbian pertanto, si sono adottate alcune scelte tecniche diverse rispetto alle implementazioni esistenti.

In particolare é stato deciso di accorpare il client *SIP* con il proxy client *ABPS*.

Lo User Agent ABPS é implementato all'interno del file ua.cpp

Modulo di Registrazione

Il modulo di registrazione é quella componente della libreria PJSIP, implementata nel file sip_reg.c, che si occupa del processo di registrazione di un client verso un server *SIP*.

Al momento dell'attivazione di un determinato account presente nel client, questo modulo é deputato alla creazione dei messaggi REGISTER associati, e alla gestione dei *SIP Response* relativi. In particolare, quando il server risponde con un errore di tipo 401/407, richiedendo un'autenticazione di tipo challenge-response, questo modulo si appoggia a quello di autenticazione (sip_auth.c) per costruire il corretto *response* da inserire all'interno della successiva REGISTER.

Se la registrazione effettuata con successo, é avvenuta verso un proxy server *ABPS*, allora viene caricato anche il modulo relativo (mod_abps), che si occuperá della firma dei messaggi *SIP/ABPS*.

Modulo di Autenticazione

Le funzioni di supporto all'autenticazione del client si trovano nel modulo sip_auth_client.c, che é stato opportunamente esteso per implementare, oltre alla classica autenticazione *HTTP/Digest* descritta in precedenza e disponibile in PJSIP, anche quella relativa al sistema *ABPS*.

I messaggi tra client e server proxy, scambiati successivamente ad un'autenticazione avvenuta con successo, dovranno essere opportunamente modificati per aggiungere ai pacchetti *SIP* i campi relativi al protocollo *SIP/APBS*.

Queste estensioni sono implementate all'interno del file sip_auth_msg.c e contengono i campi descritti in precedenza:

- **Proxyclientid**: identifica univocamente il client all'interno del sistema *ABPS*.
- **Seqnum**: numero di sequenza relativo ai pacchetti del protocollo *SIP/ABPS*.
- **Fingerprint**: hash del messaggio calcolato in funzione della chiave di sessione condivisa.

Il modulo ABPS

Questo modulo viene caricato successivamente all'avvenuta registrazione verso un proxy server *ABPS*. Il client a questo punto é in possesso della chiave condivisa (session_key_ia), per poter firmare i messaggi.

Questo modulo, che ha prioritá di livello applicazione (PJSIP_MOD_PRIORITY_APPLICATION), intercetta ogni messaggio *SIP* diretto verso il proxy server *ABPS* o ricevuto da quest'ultimo.

Se si tratta di un messaggio diretto al proxy server allora vengono aggiunti i campi relativi al protocollo *SIP/ABPS*, tra cui il fingerprint associato al messaggio stesso.

Se il messaggio proviene dal proxy, il modulo si occupa di accertarne l'autenticitá attraverso il calcolo del fingerprint e, qualora questo superi il controllo, di inoltrarlo verso l'applicazione User Agent.

Costruzione dell'INVITE

L'header *SDP* all'interno del messaggio INVITE, deve essere esteso per includere lo Zrtp Hello Hash, ovvero l'hash del primo messaggio di Hello relativo al protocollo ZRTP, che verrá avviato in seguito, preliminarmente alla costruzione del canale sicuro SRTP.

Come spiegato in precedenza, questa operazione serve ad evitare attacchi di tipo *MiTM*, realizzando uno scambio di chiavi in modalitá Diffie-Helmann, autenticato grazie all'avvenuto scambio dell'hash. Quest'ultimo infatti permette, ad entrambe le parti coinvolte nel protocollo, di accertare l'autenticitá dei primi messaggi ZRTP scambiati, essendo entrambi a conoscenza del valore hash di questi, prima che inizi il vero e proprio scambio di chiavi.

L'integritá dello Zrtp Hello Hash é garantita in quanto lo scambio avviene attraverso messaggi del protocollo *SIP/ABPS*, che sono a loro volta autenticati grazie alla chiave di sessione condivisa (session_key_ia) scambiata in precedenza.

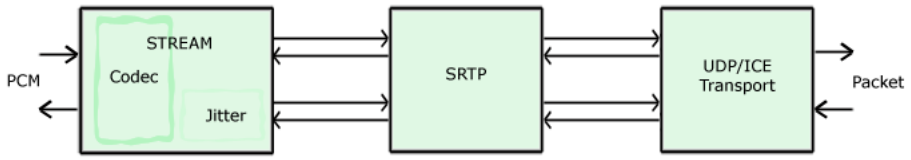
Il file sdp.c, che contiene le funzioni di supporto al protocollo *SDP*, é stato esteso per permettere l'aggiunta del campo zrtp-hash.

Transport SRTP

La trasmissione dei messaggi in PJSIP, avviene attraverso l'utilizzo di moduli denominati *Transport*, ognuno dei quali si occupa della gestione di un determinato protocollo, come ad esempio *UDP* (transport_udp.c) e *TCP* (transport_tcp.c).

I *Transport* sono moduli indipendenti l'uno dall'altro, ma spesso cooperano tra loro attraverso chiamate in sequenza che replicano la gerarchia dei protocolli.

Un esempio concreto é il *Transport SRTP* (transport_srtp.c), che si occupa di cifrare i datagram *RTP*, costruiti dal livello multimediale, e di trasmettere i pacchetti *SRTP* creati attraverso il sottostante *Transport UDP*.



Transport ZRTP

L'integrazione del protocollo ZRTP all'interno di PJSIP é stata solo parzialmente realizzata.

Il lavoro fatto finora comprende:

- integrazione della libreria libzrtp, che permetta la compilazione e il linking di questa all'interno di PJSIP, come libreria esterna.
- inizializzazione della libreria libzrtp e configurazione delle strutture dati associate.
- generazione dell'hash dei messaggi Zrtp Hello e interazione col modulo *SDP* per l'inclusione di questi nell'INVITE.

Il vero é proprio scambio di chiavi ZRTP in modalitá Diffie-Helmann non é stato ancora implementato.

Lo sviluppo di questa parte del sottosistema di sicurezza sará oggetto di successivi lavori di tesi, e avrá come oggetto la scrittura di un nuovo *Transport* di supporto a questo protocollo, che si appoggerá al modulo transport_srtp.c, in maniera analoga a quanto quest'ultimo fa con il modulo transport_udp.c .

Il Software di partenza

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

- il client *VoIP* sviluppato per piattaforme Symbian
- il server proxy *ABPS* sviluppato per piattaforme Linux

Entrambi questi software soffrivano di alcuni problemi. La prima parte dello sviluppo ha quindi riguardato la correzione dei bug e il refactoring di alcune parti del sistema.

Il client VoIP per Symbian

Il client *VoIP*, sviluppato in precedenza da altri tesisti, per quanto offrisse tutte le funzioni necessarie a gestire la procedura di autenticazione *ABPS*, era afflitto da numerosi bug e problemi architetturali che di fatto ne impedivano il corretto funzionamento.

La grossa parte delle funzioni in questione si trova all'interno dei moduli sip_reg.c e sip_auth.c, che si occupano rispettivamente delle procedure di registrazione e di autenticazione del client.

Nello specifico il client soffriva dei seguenti problemi:

- **Riconoscimento del challenge ABPS**: non riusciva a riconoscere correttamente i challenge provenienti dal server proxy *ABPS* e di conseguenza rispondeva usando le procedure del classico sistema di autenticazione *HTTP/Digest*, provocando il fallimento della procedura di autenticazione.

- **Distinzione delle procedure di autenticazione**:

  l'architettura *ABPS* necessita di due registrazioni, una da effettuare verso il server proxy basata sulla chiave precondivisa, e l'altra verso un server *SIP* esterno (es. Ekiga), basata sulla coppia nome utente e password, fornite dal provider stesso all'utente al momento della

registrazione al servizio. Una distinzione esplicita tra queste due registrazione era assente nell'implementazione originale e confondeva il client che, anche dopo essersi autenticato con successo presso il proxy server *ABPS*, non riusciva a rispondere correttamente al challenge ricevuto dal server *SIP* esterno.

- **Dipendenza degli account esterni alla registrazione APBS**:

  l'architettura *ABPS* prevede che il server proxy agisca sia da tramite per il traffico *SIP* che quello *RTP*. Questo rappresenta, come illustrato in precedenza il primo punto di ancoraggio alla rete. Qualsiasi comunicazione che proviene dal client deve necessariamente passare dal proxy server, che provvederá all'inoltro dei messaggi ricevuti solo dopo un'autenticazione avvenuta con successo.

  Nell'implementazione originale, i due processi di autenticazione partivano in contemporanea ed erano all'oscuro l'uno dell'altro. Questo fatto si traduceva in tre aspetti negativi per il funzionamento:

  - **Una mancata distinzione concettuale**: se si intende utilizzare server esterni, attraverso il servizio *ABPS*, questa possibilitá é subordinata ad un'autenticazione prelimanare al server proxy stesso.
  - **Overhead nel traffico**: la mancata autenticazione al proxy server *ABPS* implica che tutti i messaggi ricevuti dal client vengano scartati. Era del tutto inutile e controproducente tentare un'autenticazione a un servizio esterno prima dell'avvenuta autenticazione, soprattutto in un contesto dove la quantitá di traffico trasmessa é un fattore importante sia in termini di consumo di banda che di costi del servizio.
  - **Errori nel proxy server**: in certe situazioni il proxy server non gestiva correttamente i due tentativi di autenticazione contemporanei. Quest'ultimo, probabilmente a causa di qualche bug nell'implementazione delle policy di sicurezza, risultava confuso dal

fatto che dallo stesso indirizzo IP sorgente provenissero richieste legittime e non legittime, comportando un fallimento dell'autenticazione ABPS anche laddove la risposta al challenge avvenisse correttamente.

I primi due problemi, legati sostanzialmente a bug presenti nell'implementazione, sono stati corretti.

Al terzo problema si é provveduto introducendo nel client il concetto di "dipendenza" dall'account *ABPS*: qualsiasi altro account presente nel client, che si vuole sfruttare per effettuare operazioni legate al protocollo *SIP*, deve prima attendere che la procedura di autenticazione al sistema *ABPS* sia terminata con successo. Naturalmente il client puó conservare i dati relativi agli account esterni ed eseguire le registrazioni ai rispettivi servizi, a patto che gli account in questione non siano marchiati come dipendenti da *ABPS*. In tal caso le procedure *SIP* possono avvenire in autonomia, ma senza passare attraverso il server proxy, e quindi non potendo avvantaggiarsi dei servizi di supporto alla mobilitá offerti da questo.

Il Server Proxy ABPS

Il server proxy *ABPS* godeva di una stabilitá 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.

Tuttavia le peculiaritá del client Symbian, legate in particolare ai tempi di risposta dell'applicazione, hanno permesso di individuare alcuni bug presenti nel proxy, permettendo la correzione di questi e il conseguente miglioramento del software stesso. Si noti a tal proposito che il test del client é avvenuto attraverso il simulatore, che é molto lento. Questa condizione, se da un lato ha dilatato enormemente i tempi di sviluppo, ha altresí permesso di evidenziare problematiche che, con l'utilizzo di software piú prestanti, probabilmente non sarebbero emerse.

In particolare:

- Bug nel calcolo dei tempi di expire: il proxy server soffriva di un bug che provocava un valore troppo ristretto dei tempi di expire della registrazione. Il relativo salvataggio della registration table, che tiene traccia degli avvenuti tentativi di autenticazione, riusciti e falliti, provocava un fallimento della procedura di registrazione del client Symbian, spesso troppo lento per riuscire a completarla con successo entro i limiti temporali stabiliti.

- Bug nel calcolo del fingerprint: dopo aver ricevuto un messaggio dal client, il proxy server lo memorizza all'interno di apposite strutture dati fornite dalla libreria libosip. Il calcolo del fingerprint avveniva non sul messaggio originale, bensí sulla rappresentazione di questo in formato caratteri, ricavata in seguito a partire dai dati memorizzati all'interno delle strutture stesse. Succedeva in taluni casi che, per ragioni interne al server stesso, i dati venissero memorizzati in maniera leggermente diversa rispetto al messaggio originale provocando il fallimento del confronto tra il fingerprint calcolato su questi, e quello ricevuto nel messaggio originale del client.

Entrambi questi problemi sono stati risolti: il primo correggendo il calcolo dei tempi di expire, il secondo introducendo un riferimento al messaggio originale ricevuto dal server, sui cui effettuare il calcolo del fingerprint.

I Firewall della rete Unibo

I test dell'applicazione sono stati condotti utilizzando un proxy *ABPS*, collocato su un server interno alla rete della Facoltá di Informatica dell'Universitá di Bologna.

Durante lo sviluppo ci si é accorti che talvolta alcuni pacchetti venivano intercettati dai firewall della rete Unibo che, riconoscendo il protocollo *SIP*, modificavano il campo Contact del datagram in transito, per ragioni probabilmente legate all'amministrazione della rete stessa.

La modifica del pacchetto comportava necessariamente lo scarto di questo da parte del proxy server *ABPS*, a causa di un fallito confronto tra il fingerprint ricevuto, cioé quello calcolato dal client sul pacchetto originale, e quello calcolato dal proxy sul datagram col campo Contact modificato.

Questa circostanza ci ha portato difronte a problematiche inaspettate, che esulavano dallo scopo della tesi stessa. É stato pertanto deciso di sorvolare temporaneamente il problema, modificando il campo indicante il protocollo *SIP* nei datagram in transito, in maniera tale da "ingannare" i firewall Unibo che in questa maniera non modificano piú i pacchetti.

Il problema rimane particolarmente delicato, e necessitá di soluzioni piú eleganti da stabilirsi in seguito ad un'analisi approfondita della problematica, che sará probabilmente presa in considerazione da tesi successive in materia.

Conclusioni

In questo documento sono state analizzate le principali tecnologie e protocolli di supporto alle trasmissioni *VoIP*, con particolare enfasi per quelli che rigurdano il contesto di sicurezza di tali applicazioni, in particolare in relazione alle proprietá di autenticazione e confidenzialitá.

É stata presentata l'architettura *ABPS* che, sfruttando le tecnologie descritte in precedenza, é in grado di garantire a un device multi-homed in movimento, la possibilitá di effettuare traffico *VoIP*, sfruttando al meglio le potenzialitá offerte dalle interfacce multiple a disposizione, che gli consentono di rientrare nei limiti richiesti per garantire la *QoS* necessaria ad applicazioni multimediali interattive.

Il lavoro implementativo ha consentito di completare lo sviluppo del sottosistema di autenticazione nel client per Symbian OS, che ora é in grado di autenticarsi con successo sia presso i proxy server *ABPS*, che presso un generico provider *VoIP* come Ekiga.

Lo sviluppo dell'applicazione ha inoltre permesso di rafforzare l'implementazione del proxy server *ABPS*, su sistema operativo Linux, correggendo diversi bug e migliorando la gestione del processo di autenticazione, in particolare per quanto riguarda il calcolo del fingerprint sui pacchetti.

É stato possibile inoltre individuare alcune problematiche fino ad ora sconosciute, come la modifica dei pacchetti in transito da parte dei firewall Unibo. Tali problemi, anche in assenza di un'analisi approfondita che sará eventualmente oggetto di successivi lavori di tesi, sono comunque stati temporaneamente risolti attraverso un'oscuramento del protocollo, ottenuto mo-

dificando adeguatamente sia il client che il proxy server.

La libreria libzrtp é stata integrata con successo all'interno di PJSIP, e sono state implementate le fasi preliminari allo scambio di chiavi ZRTP, come la generazione dello Zrtp Hello Hash e lo scambio di questo con il proxy attraverso l'header SDP allegato al messaggio di INVITE. L'implementazione vera e propria dello scambio di chiavi in modalitá Diffie-Hellman non é stata ancora completata.

Ringraziamenti

Colgo l'occasione per ringraziare tutti i professori della Facoltá di Informatica, che mi hanno permesso attraverso i loro insegnamenti, di conseguire le competenze necessarie a svolgere questo compito. In particolare il Dott. Vittorio Ghini, il cui aiuto é stato fondamentale sia per quanto riguarda l'analisi e la comprensione delle tecnologie legate al panorama *VoIP*, che per quanto concerne lo sviluppo vero e proprio dell'applicazione in cui é stato parte attiva.

Ringrazio inoltre gli altri tesisti con cui ho collaborato, in particolar modo Stefano Rizzo che aveva iniziato lo sviluppo del client *VoIP ABPS* per Symbian OS, e Lorenzo Paoloni con il quale ho collaborato per l'implementazione di quanto esposto in questa tesi.

Bibliografia

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

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

[3] Ieee 802.11: Wireless lan medium access control (mac) and physical layer (phy) specifications. 2007.

[4] Boing wireless. 2008.

[5] Fonera. 2008.

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

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

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

[9] A. Bosselaers B.D. Boer. Collisions for the compression function of md5. *Advances in Cryptology, Proceedings of EUROCRYPT*, 1994.

[10] A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. 2000.

[11] Matt Bishop. Introduction to computer security. 2008.

[12] G. Camarillo and J. Rosenberg. The Alternative Network Address Types (ANAT) Semantics for the Session Description Protocol (SDP) Grouping Framework. RFC 4091, Internet Engineering Task Force, June 2005.

[13] C.Ellison and B.Schneier. Ten risks of pki: What you're not being told about public key infrastructure. *Computer Security Journal - Volume XVI, Number 1*, 2000.

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

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

[16] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory, vol. IT-22*, Nov. 1976.

[17] H. Dobbertin. The status of md5 after a recent attack. *RSA Labs' CryptoBytes, Vol. 2 No. 2*, Summer 1996.

[18] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, Internet Engineering Task Force, June 1999.

[19] E. Gustafsson and A. Jonsson. Always best connected. *IEEE Comm. Mag., vol. 10, no. 1*, 2003.

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

[21] X. Lai J Liang. Improved collision attack on hash function md5. *Journal of Computer Science and Technology*, 2007.

[22] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775, Internet Engineering Task Force, June 2004.

[23] Tom. Keating. Internet phone release 4. *Computer Telephony Interaction Magazine*.

[24] V. Klima. Tunnels in hash functions: Md5 collisions within a minute. *IACR Eprint Server*, 2006.

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

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

[27] D. McGrew and S. Fluhrer. Attacks on encryption of redundant plaintext and implications on internet security. 2000.

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

[29] National Institute of Standard and Technology. Advance encryption standard (aes). *FIPS Pub 197*, nov 2001.

[30] W. Zhuang P. Wang, H. Jiang. Capacity improvement and analysis for voice/data traffic over wlans. *IEEE Transaction on Wireless Communications, Vol. 6, No. 4*, April 2007.

[31] A. Johnston P. Zimmermann and Ed.Avaya. Zrtp: Media path key agreement for unicast secure rtp. *Internet-Draft*, 2010.

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

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

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

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

[36] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489, Internet Engineering Task Force, March 2003.

[37] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, Internet Engineering Task Force, July 2003.

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

[39] Giorgia Lodi Fabio Panzieri Antonio Messina Vittorio Ghini, Luigi Enrico Tomaselli. Always best packet switching for sip-based mobile multimedia services. *Dept. of Computer Science, University of Bologna, Italy.*

[40] H. Yu X. Wang. How to break md5 and other hash functions. *EUROCRYPT, vol. 3494 of Lecture Notes in Computer Science*, 2005.

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