DIVA-SEC: UN SISTEMA SICURO DI VIDEO ON-DEMAND*

M. Baltatu[1], C. Basile[2], A. Lioy[1], F. Maino[1], A. Mazzeo[2], N. Mazzocca[2], D. Mazzocchi[1]

DiVA-SEC è un'architettura per il Video On-Demand (VoD) che consente di garantire diversi livelli di sicurezza attraverso l'utilizzo di certificati a chiave pubblica X.509v3. Gli attori coinvolti nell'architettura DiVA-SEC (il Video Client, il Video Server ed i Film Server) sono dotati di un certificato a chiave pubblica per autenticare in modo forte tutte le parti coinvolte in una transazione e garantire integrità, riservatezza e non ripudio nel protocollo di comunicazione del sistema. Attraverso un token di autorizzazione, il Video Server controlla l'accesso dei Video Client ad uno specifico Film Server, scelto in funzione del carico complessivo del sistema. In questo modo è possibile utilizzare l'architettura DiVA-SEC negli scenari più disparati di distribuzione di Video On-Demand: dalla distribuzione di filmati di pubblico dominio, ai diversi scenari commerciali in cui sia necessario controllare l'accesso al Film Server o garantire la riservatezza della lista dei film distribuiti, del film richiesto al Video Server e della trasmissione del film stesso. L'architettura DiVA-SEC si propone quindi come un sistema di Video On-Demand che risponde ai requisiti di sicurezza richiesti dalle applicazioni commerciali, offrendo svariate tipologie di servizio da utilizzare nei diversi ambiti applicativi. Il meccanismo di sicurezza è integrato con l'architettura generale di DiVA e non influisce in maniera apprezzabile sulle prestazioni delle varie componenti del sistema e sulle sue caratteristiche di bilanciamento del carico.

1. INTRODUZIONE

DiVA (Distribuited Video Architecture) [1] è un sistema sperimentale di Video on Demand (VoD) sviluppato presso l'Università Federico II di Napoli e basato su un'architettura client/server per la raccolta e la distribuzione in tempo reale di sorgenti video da un database remoto. Sono presenti uno o più client che richiedono ad un server remoto la trasmissione di un filmato o di una qualsiasi sequenza video e/o audio; il sistema, quindi, presenta quelle caratteristiche di

* Il lavoro descritto in questo articolo è stato svolto col supporto del Ministero dell'Università e della Ricerca Scientifica (MURST) nell'ambito del progetto in cofinanziamento MOSAICO (Metodologie e Strumenti di Progetto di Sistemi ad Alte Prestazioni per Applicazioni Distribuite).

[1] Politecnico di Torino, Dipartimento di Automatica e Informatica
[2] II Università degli Studi di Napoli, Dipartimento di Ingegneria dell'Informazione

interattività e dinamicità tipiche delle moderne applicazioni multimediali distribuite (DMA).

Nei sistemi VoD in generale si assume che il computer usato dall'utente non abbia memoria a sufficienza per contenere l'intera sequenza audio-video, di conseguenza risulta necessario avere un'entità che si occupi della trasmissione dei frame relativi alla sequenza richiesta. E' inoltre ipotizzabile la presenza di più richieste relative ad utenti distinti: il sistema deve quindi potere scalare le richieste. Si assume inoltre che le sequenze audio-video disponibili siano distribuite in rete su macchine differenti, ognuna delle quali gestisca una propria base dati locale dei filmati ed il tutto deve essere trasparente per l'utente finale. Si richiede, naturalmente, un livello

L'intrinseca complessità per lo sviluppo di applicazioni DMA richiama la necessità di metodologie dell'ingegneria del software pratiche ed efficaci per la modellistica, il progetto e l'implementazione. E' ormai chiaro che lo sviluppo di complesse applicazioni DMA trae beneficio dall'adozione delle metodologie OO e dalle tecnologie per la distribuzione degli oggetti, in quanto esse offrono potenziali vantaggi in termini di produttività, riusabilità, integrazione di sistemi legacy, interoperabilità tra piattaforme software/hardware eterogenee, e semplicità di manutenzione. In questo lavoro è stato utilizzato il linguaggio UML per l'analisi e la progettazione del sistema DiVA; inoltre per l'implementazione si è fatto uso della piattaforma CORBA come middleware per la distribuzione degli oggetti dell'applicazione.

DiVA-SEC, che è l'estensione sicura di DiVA, è stata sviluppata nel corso del progetto MURST MOSAICO, ed integrata con l'infrastruttura di certificazione ICE-TEL [2], un progetto dell'Unione Europea (programma Telematics for Research) per sviluppare un'infrastruttura di certificazione a chiave pubblica sperimentale di dimensione europea.

Gli attori dell'architettura DiVA-SEC sono dotati di un certificato a chiave pubblica X.509v3 utilizzato per garantire, laddove necessario, autenticazione forte, integrità, riservatezza e non-ripudio. Tali certificati vengono utilizzati per:

- garantire l'identità degli attori coinvolti nell'architettura DiVA-SEC attraverso

- garantire la riservatezza e l'integrità dei messaggi previsti dal protocollo DiVA-SEC, attraverso l'uso di firme digitali e di opportuni meccanismi di cifratura
- garantire il non-ripudio dei messaggi scambiati tra gli attori del sistema, a supporto di eventuali funzioni di pagamento del servizio richiesto

- distribuire un token per il controllo dell'accesso ai filmati distribuiti attraverso il sistema DiVA-SEC

Queste funzionalità permettono di prevedere una varietà di scenari di utilizzo dell'architettura DiVA-SEC che sembrano coprire un ampio campo di applicazioni:

- distribuzione di filmati di pubblico dominio, con la possibilità di autenticare reciprocamente l'utente ed i dispositivi di distribuzione dei filmati
- distribuzione di filmati a pagamento con possibilità di garantire la riservatezza della lista di film distribuita dal sistema, la riservatezza del film richiesto ed il non-ripudio della richiesta di accesso ad un filmato
- distribuzione di filmati con le caratteristiche del punto precedente più la possibilità di cifrare il filmato distribuito

L'architettura DiVA-SEC si propone quindi come un sistema di Video On-Demand che risponde ai requisiti di sicurezza richiesti dalle applicazioni commerciali a cui tipicamente un servizio di VoD è rivolto, offrendo diverse tipologie di servizio da utilizzare in differenti contesti applicativi. Il meccanismo di sicurezza è integrato con l'architettura generale di DiVA e non ne compromette le prestazioni complessive e le caratteristiche di bilanciamento del carico sulle varie componenti del sistema.

Nel seguito viene data una breve descrizione dell'architettura del sistema DiVA e, partendo dall'analisi degli scenari di utilizzo di un sistema sicuro di VoD, viene presentata l'implementazione della sicurezza in DiVA-SEC. Infine vengono tratteggiate le linee lungo le quali verrà sviluppato il lavoro futuro.

2. L'ARCHITETTURA DEL SISTEMA DIVA

L'architettura generale di DiVA prevede requisiti di gestione e configurazione del sistema, oltre a funzionalità di trasmissione dei dati di tipo continuo tra client e server.

I dati scambiati fra i vari componenti sono fondamentalmente di due tipi:

- il primo flusso di dati è relativo alle informazioni necessarie per consentire l'identificazione di un utente, la gestione del direttorio dei film e delle risorse rappresentate dal server sul quale è fisicamente memorizzata l'informazione richiesta;
- il secondo è rappresentato dalla sequenza di fotogrammi (*frame*) e dalla sequenza audio del filmato richiesto, una volta identificato il server che lo contiene

Relativamente alla trasmissione dei frame, considerato che essa necessita di una rigida sincronizzazione, si è utilizzata la libreria CMT (*Continuous Media Toolkit*), sviluppata dal *Berkeley Multimedia Research Group* (BMRC) dell'Università della California a Berkeley. CMT costituisce una piattaforma orientata allo sviluppo di applicazioni per la gestione di flussi di informazioni continui nel tempo del tipo audio e/o video (*streaming*). Tale libreria mette a disposizione del programmatore un insieme di oggetti per la trasmissione, la decodifica e la visualizzazione di dati multimediali codificati in diversi formati. Questi oggetti sono utilizzabili da script Tcl/Tk mediante un'estensione del linguaggio di script. La libreria CMT consente di avere un diretto controllo dei sistemi di rete garantendo elevati livelli di prestazioni. Una soluzione alternativa sarebbe quella di utilizzare la libreria Java JMF (*Java Media Framework*).

Per i fini del caso di studio, la parte relativa allo *streaming* dei dati tra client e server, si riconduce, dal punto di vista delle specifiche, ad utilizzare un sistema di trasporto secondo una libreria che garantisce tempi di trasmissione prefissati su particolari sistemi di rete (per esempio ATM). La maggiore complessità delle specifiche funzionali risulta pertanto nella parte relativa alla gestione degli accessi e delle richieste da parte di un utente.

Un utente DiVA dispone di un processo VideoClient per sottoporre le proprie richieste. Il VideoClient indirizza tali richieste al VideoServer, che è il cuore del sistema: quest'ultimo controlla se l'utente può accedere al sistema, ne esamina la richiesta ed eventualmente la indirizza ad uno dei FilmServer, il quale si occupa della trasmissione dei frame della sequenza richiesta verso il VideoClient.

L'utente nella configurazione minima del sistema può richiedere:

- la lista dei film disponibili,
- la trasmissione di una particolare clip (audio e/o video).

Le funzionalità di gestione previste sono:

- aggiornamento della lista dei film disponibili;
- gestione dell'aggiunta di un nuovo FilmServer;
- l'attivazione automatica dei processi FilmServer.

L'architettura completa del sistema DiVA è schematizzata in Figura 1.

L'interfaccia grafica (scritta in Tcl/Tk) del VideoClient richiama l'oggetto CORBA *VC* (VideoClient) a cui propaga le richieste dell'utente (1); tali richieste sono *VS* (VideoServer) che viene raggiunto tramite l'ORB (sequenza di messaggi 2.1, 2.2).

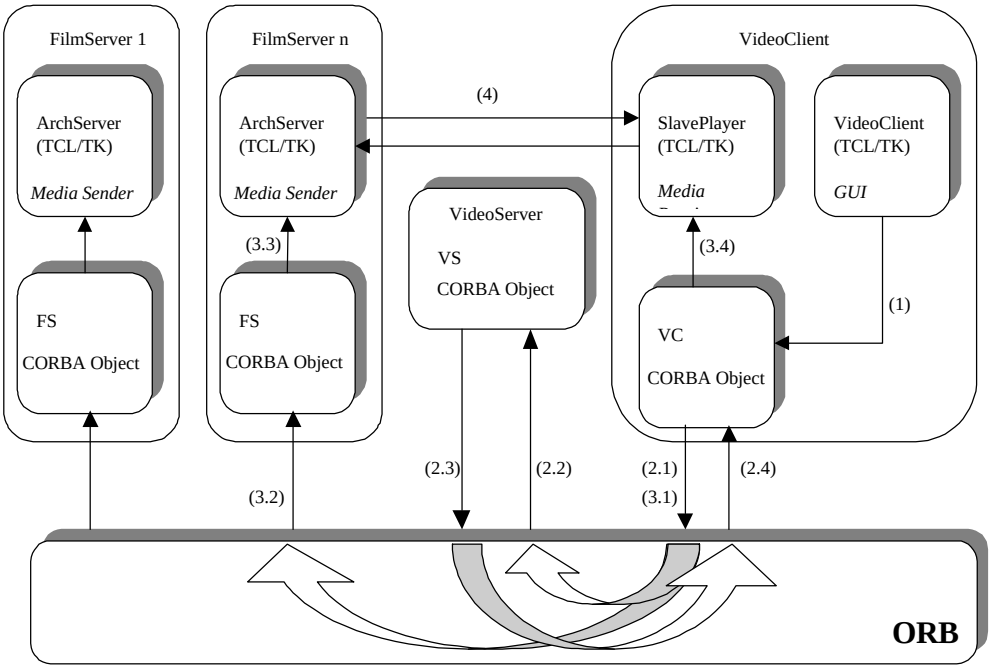

Figura 1: architettura del sistema DiVA

Il CORBA *VS* valuta le richieste e, passando per l'ORB, comunica l'esito delle stesse al CORBA *VC* (2.3, 2.4).

Nel caso di esito positivo il CORBA *VC* contatta il CORBA *FS* (l'interfaccia CORBA del FilmServer) per attivare il processo Tcl/Tk *ArchiveServer* che si occuperà della trasmissione dei frame audio e/o video al VideoClient (3.1, 3.2, 3.3).

In seguito il CORBA *VC* attiva, sulle macchine dove desidera che sia visualizzato il filmato, i processi Tcl/Tk *SlavePlayer* dedicati alla ricezione ed alla visualizzazione dei frame (3.4).

A questo punto inizia la trasmissione del filmato sul canale creato tra l'*ArchiveServer* e lo *SlavePlayer* (4).

3. POLITICHE DI SICUREZZA E SCENARI DI UTILIZZO

Per poter valutare le possibili politiche di sicurezza da introdurre nel sistema, saranno analizzate innanzitutto le interazioni fra le varie componenti del sistema.

Analisi delle interazioni fra gli attori del sistema

Le operazioni fondamentali che il VideoClient effettua, attraverso il componente CORBA VC, sono:

- la richiesta al VS della lista dei filmati (*getFilmList*)
- la richiesta al VS del permesso (*token*) per la trasmissione di un particolare filmato (*getFilmInf*)
- la richiesta al FS di trasmissione di quel filmato (*upArchServer*), mediante il *token* ottenuto dal VS
- la richiesta di attivazione, eventualmente su più macchine, ai rispettivi SP dei processi Tcl/Tk *SlavePlayer* per la visualizzazione del filmato (*upSlavePlayer*).

Il FS, a seguito dell'invocazione del metodo *upArchServer*, attiva il processo Tcl/Tk *ArchiveServer* per la trasmissione dei frame del filmato.

L'interazione tra VideoServer e FilmServer (affidata alle interfacce VS2FS e FS2VS) essenzialmente contempla la gestione del direttorio dei filmati e lo scambio di informazioni per effettuare un bilanciamento del carico sui FilmServer.

Scenari di Utilizzo

Le possibili politiche di sicurezza in DiVA in riferimento alle possibili applicazioni pratiche del sistema possono essere individuate in:

1. Nessuna sicurezza
2. Autenticazione tra VideoServer e FilmServer
3. Autenticazione tra VideoServer, FilmServer e VideoClient
4. Autenticazione tra VideoServer, FilmServer e VideoClient; riservatezza della FilmList, del Token e della richiesta di visualizzazione (parametro filmId nel metodo *getFilmInf* del VideoServer)
5. Autenticazione tra VideoServer, FilmServer e VideoClient; riservatezza della FilmList, del Token, della richiesta di visualizzazione e nella trasmissione dei frame del filmato
6. Autenticazione tra VideoServer, FilmServer e VideoClient; riservatezza della FilmList, del Token, della richiesta di visualizzazione e nella trasmissione dei frame del filmato; sicurezza nella transazione per l'addebito all'utente dell'importo della trasmissione del filmato.

Lo scenario (1) è quello in cui il concetto di sicurezza non viene affatto inserito nell'architettura. Questo può avere senso in particolari ambiti applicativi quale, ad

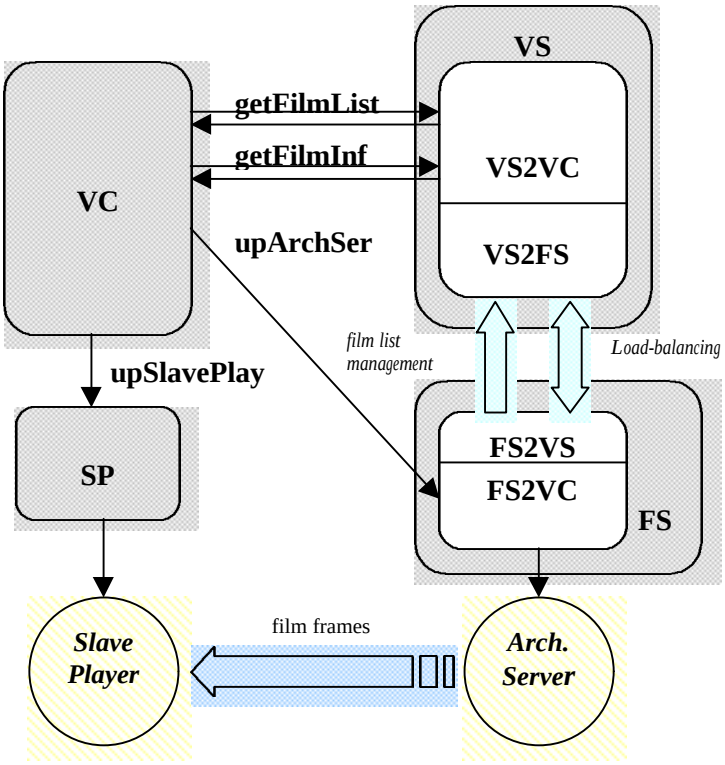

Figura 2: messaggi scambiati nel sistema DiVA-SEC

esempio, la trasmissione di filmati di pubblico dominio.

Sempre nell'ambito della trasmissione di filmati di pubblico dominio lo scenario (2) è sicuramente più robusto del precedente in quanto, garantendo l'autenticazione delle componenti del sistema, protegge lo stesso da eventuali attacchi atti a disturbarne il funzionamento.

Nello scenario (3) si introduce l'autenticazione del VideoClient. Il sistema, quindi, può diventare selettivo verso gli utenti nel permettere a questi la visione dei filmati. Tuttavia, in questo scenario, il sistema si rivela alquanto indifeso verso eventuali attacchi atti a rubare informazioni riservate, quali la lista dei filmati (FilmList) ed il Token (in quale costituisce il permesso ceduto dal VideoServer ad un utente per la visione di un certo filmato).

Con lo scenario (4) si risolve il problema della sicurezza per le informazioni che si scambiano VideoServer, FilmServer e VideoClient. Tuttavia per un utilizzo commerciale del sistema di VoD è necessario che nessun altro, eccetto l'utente autorizzato, possa visionare il filmato.

: richiede la lista dei filmati disponibili su un VideoServer. Si possono individuare due soluzioni. La prima, a massima sicurezza, garantisce riservatezza della comunicazione, autenticazione forte del richiedente, non ripudio della richiesta della FilmList e della risposta. Posto:

X = "getFilmList "+ *nonce* + E(K-VC,hash("getFilmList",*nonce*)) + cert(VC)

allora il VC genererà una chiave simmetrica KS ed invierà al VS il seguente messaggio (richiesta firmata e cifrata, seguita dalla chiave usata per la cifratura):

$$E (KS, X) + E (K\text{-}VS^{-1}, KS)$$

a cui il VS risponderà con la lista dei film disponibili (firmata e cifrata):

E (KS, *filmList* + *nonce* + E(K-VS,hash(*filmList*+*nonce*)))

L'utilizzo della stringa pseudo-casuale *nonce*, da utilizzare "*no more than once*", consente al VideoServer di individuare attacchi di tipo replay e di non rispondere a messaggi che contengano nonce già utilizzati. L'implementazione dovrà essere tale da tenere traccia dei nonce già utilizzati da ciascun VideoClient.

Per rendere più efficiente il procedimento di firma della risposta, il VideoServer potrebbe utilizzare, invece che una firma a chiave pubblica, una firma basata su *keyed-MAC* (ad esempio HMAC [3]) sfruttando la chiave KS. Il messaggio di risposta del video server quindi diventerebbe:

E (KS, *filmList* + *nonce* + HMAC(KS,*filmList*+*nonce*))

Questa soluzione non garantirebbe contro il ripudio della FilmList da parte del VideoServer, ma ridurrebbe il tempo di generazione della risposta. Poiché la FilmList contiene anche informazioni sui prezzi dei vari filmati é importante, in questo caso, garantire il non ripudio del messaggio e si è quindi preferito utilizzare la firma a chiave pubblica.

Si noti che la cifratura del certificato del VideoClient si rende necessaria per garantire l'anonimato del richiedente, che potrebbe voler mantenere riservata la richiesta inoltrata presso un VideoServer. Nel caso che non sia richiesta la riservatezza delle comunicazioni tra VideoClient e VideoServer si possono usare i medesimi tipi di messaggi eliminando solamente la cifratura simmetrica.

Nell'implementazione attuale, per limitare la complessità, l'interfaccia non prevede *nonce* ed il VideoClient, nel caso in cui non intenda crittografare i dati, passa al VideoServer un puntatore a NULL al posto della chiave di sessione.

Il Video Server gestisce localmente il processo di autorizzazione all'accesso ai filmati attraverso i file di configurazione indicati in Figura 3:

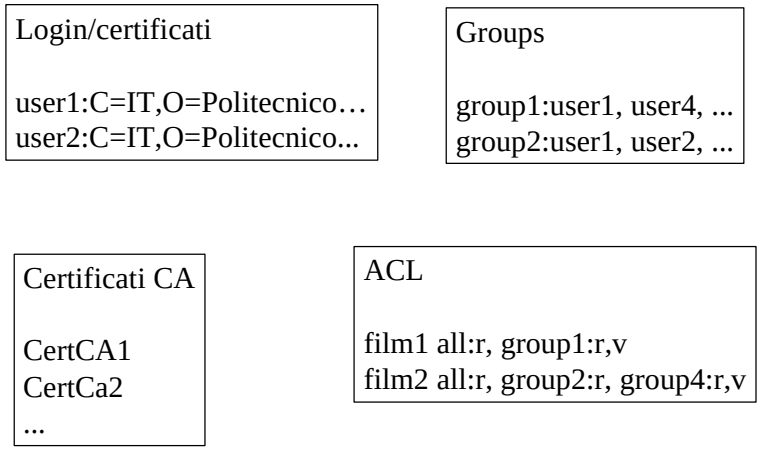

Figura 3: Access Control List sul VideoServer

- il file login/certificati mappa ciascun certificato su un utente definito localmente al VideoServer
- il file groups consente di raggruppare degli utenti aventi caratteristiche comuni in uno o più gruppi definiti localmente. Tutti gli utenti appartengono almeno al gruppo *all*
- l'Access Control List, consente di definire per ciascun gruppo o utente le modalità di accesso a ciascun filmato: visualizzazione del filmato (attributo *v*), oppure accesso in lettura alle informazioni collegate al filmato tramite attributo *r* (descrizione, costo, ...).
- la tabella *login/certificati* individua i certificati in maniera univoca attraverso il *Distinguished Name* del certificato e, qualora questo non fosse univoco attraverso l'attributo *Unique Key Identifier* del certificato con cui è stato firmato; questo consente di evitare di aggiornare la tabella allo scadere dei certificati degli utenti.

Per verificare il certificato presentato dal Video Client, il Video Server conserva nella tabella *Certificati CA* i certificati delle CA considerate fidate.

Va notato che il concetto di login è definito localmente al Video Server ed introdotto solo per rendere più semplice la scrittura delle ACL. Il FilmServer è al di fuori dello *scope* di tale concetto ed identifica l'utente solo tramite il certificato presentato autorizzandolo a compiere le operazioni indicate nel token autorizzante.

Metodo getFilmInf: richiede l'autorizzazione (*token*) per la visione di un particolare filmato. Posto:

$$X = filmID + nonce + E(K\text{-}VC, hash(filmID+nonce)) + cert(VC)$$

allora il VC genererà una chiave KS ed invierà al VS il seguente messaggio (richiesta firmata e cifrata, seguita dalla chiave usata per la cifratura):

$$E(KS, X) + E(K\text{-}VS^{-1}, KS)$$

a cui il VS risponderà con il token (cifrato):

$$E(KS, token + cert(VFS) + cert(AFS))$$

Il Video Server, dopo aver autenticato il Video Client, verifica tramite le ACL i diritti di accesso al filmato identificato da *filmID*, quindi sceglie il FilmServer in base al carico corrente e costruisce il *token* che consente al Video Client la richiesta di visualizzazione del filmato su tale Film Server. Oltre al token, il Video Server invia i certificati dei Film Server audio (AFS) e video (VFS) in modo che il Video Client possa disporne nelle comunicazioni successive per cifrare i messaggi diretti ai Film Server.

Il token è una struttura contenente i seguenti campi:

- **Token.Info**
  - indirizzo IP del Film Server per il video (VFS)
  - indirizzo IP del Film Server per l'audio (AFS)
  - E (URLvideo, $K\text{-}VFS^{-1}$)
  - E (URLaudio, $K\text{-}AFS^{-1}$)
  - Time Limit
  - IDutente
  - numero di porta per il video
  - numero di porta per l'audio
  - E (KTX, $K\text{-}VC^{-1}$)
  - E (KTX, $K\text{-}AFS^{-1}$)
  - E (KTX, $K\text{-}VFS^{-1}$)
- **Token.Signature:**
  - E (K-VS, hash(token.info))

Il Film Server verifica la firma apposta sul token per controllare che sia stato effettivamente rilasciato dal Video Server. Si noti che il token viene utilizzato per trasportare in modo riservato informazioni tra il VideoServer ed il FilmServer attraverso il VideoClient.

I campi URLvideo e URLaudio sono cifrati con la chiave pubblica dei FilmServer (eventualmente coincidenti) presso i quali ottenere il filmato, per non consentire al VideoClient di osservarne il contenuto, che potrebbe essere utilizzato per capire la configurazione del filesystem dei FilmServer.

Il campo Time Limit serve per garantire che il token possa essere utilizzato una sola volta dal Video Client (*one-time token*). Quando il Film Server ottiene il token dal Video Client controlla se il Time Limit è maggiore dell'ora corrente. In caso negativo non accetta la richiesta, altrimenti controlla se il token è contenuto nella lista dei token già utilizzati. Se il token non è stato ancora utilizzato accetta la richiesta del Video Client ed inserisce il token in questa lista. Va notato che, in ogni caso, la dimensione della lista dei token utilizzati è limitata poiché un token può essere rimosso non appena l'ora corrente eccede il Time Limit riportato sul token stesso.

La trasmissione del token al FilmServer propaga al VideoClient ed al VideoServer una chiave di trasmissione (KTX) che verrà passata ai processi Tcl *Slave Player* ed *Archive Server* per autenticarsi reciprocamente. Un protocollo a sfida firmata

tramite keyed-MAC tra VideoClient e VideoServer consente l'autenticazione tra i processi. La chiave di trasmissione potrebbe anche essere utilizzata per la crittografia dei frame, qualora si volesse garantire la riservatezza della trasmissione del filmato. La crittografia della trasmissione del filmato però presenta forti ripercussioni sulla qualità del servizio stesso: si sta valutando quali possano essere le soluzioni che consentano di garantire un adeguato livello di riservatezza senza compromettere le prestazioni dell'architettura di VoD basata, a livello trasporto, su Cyclic UDP [4].

Interazione VC – FS2VC

Metodo upArchServer: richiede la trasmissione del filmato di cui ha ottenuto il token. Posto:

$$X = token + E(K\text{-}VC, hash(token)) + cert(VC)$$

il VideoClient invia il token firmato e cifrato:

$$E(KS, X) + E(K\text{-}FS^{-1}, KS)$$

Il Film Server deve verificare che il campo IDutente riportato nel token, corrisponda a quello del certificato con cui è stato firmato il token stesso. Ciò consente di evitare che il token rilasciato ad un utente possa essere utilizzato da un altro utente. Questo campo può essere implementato con hash(cert(VC)), quindi il FS non deve far altro che calcolarsi hash(cert(VC)) dal cert(VC) ricevuto e confrontarlo con il campo IDutente.

Si noti che in questo caso il token stesso, che contiene un campo TimeLimit unico per ciascuna richiesta, impedisce attacchi di tipo replay. Inoltre il Film Server può verificare la firma del Video Server nel campo *token.signature* per accertarsi della validità del token. Infine il Film Server, per verificare che il token sia stato generato per essere consumato su quel FilmServer, verifica la corrispondenza del proprio indirizzo IP con quello indicato nel token.

Interazione VC – SP

Metodo upSlavePlayer: richiede la visualizzazione del filmato di cui dispone il token. Si noti che, in generale, lo SlavePlayer è un processo che risiede sulla stessa macchina sul quale si trova il VideoClient e quindi la chiave di trasmissione può essere comunicata direttamente a tale processo senza fare ricorso ad un ulteriore imbustamento. Però si potrebbe pensare ad un sistema completamente distribuito in cui lo SlavePlayer risiede su un'altra macchina ed ha un proprio certificato. In questo caso l'autenticazione tra VideoClient e SlavePlayer utilizzerebbe il seguente messaggio:

$$E(KS, token + E(K\text{-}VC, hash(token) + cert(VC) + KTX) + E(K\text{-}SP^{-1}, KS)$$

Lo Slave Player, in maniera analoga al Film Server, verifica l'identità del Video Client controllando il campo *IDutente* nel token.

Il Video Client deve avere una lista di tutti i certificati degli Slave Player sui quali può attivare la visualizzazione del filmato. Inoltre esso comunica allo Slave Player la chiave di trasmissione KTX ricevuta dal Video Server.

Interazione FS – VS2FS

Metodo addFilmInf: inserisce un nuovo filmato nel direttorio globale dei filmati disponibili. Posto:

$$X = \text{filmInf} + \text{nonce} + E(K\text{-FS}, \text{hash}(\text{filmInf},\text{nonce})) + \text{cert}(FS)$$

il Film Server invia il seguente messaggio:

$$E(KS, X) + E(K\text{-}VS^{-1}, KS)$$

Il Video Server deve autenticare il Film Server controllando i suoi dati nel certificato; quindi il Video Server avrà una tabella con tutti i dati dei Film Server.

Implementazione

Il sistema DiVA-SEC è stato implementato su sistemi Windows utilizzando Orbix [5] per le funzionalità CORBA e le librerie OpenSSL [6] per l'implementazione delle funzionalità crittografiche e la gestione dei certificati.

Il materiale crittografico (chiave privata, chiave pubblica e certificato) degli attori coinvolti nel sistema DiVA-SEC è memorizzato in formato PKCS#12 [7]. Questo consente di utilizzare un normale browser Web di ultima generazione come interfaccia verso l'infrastruttura di certificazione. L'utente può quindi registrarsi presso una Certification Authority riconosciuta da chi gestisce il sistema di VoD ed ottenere un certificato a chiave pubblica. Il browser Web consente di esportare la chiave privata ed il certificato a chiave pubblica in un file in formato PKCS#12 che verrà direttamente utilizzato dal VideoClient. Lo stesso approccio può essere utilizzato da chi gestisce il VideoServer ed i FilmServer.

L'aggiunta della sicurezza nei messaggi scambiati nel sistema DiVA-SEC non ha modificato le prestazioni complessive del sistema e le prove effettuate non hanno mostrato rallentamenti rilevabili nel processo di autorizzazione all'accesso ad un filmato. Riteniamo invece che la cifratura dello stream audio/video possa comportare un notevole degrado della qualità del servizio e, per questo motivo, stiamo valutando quali siano le soluzioni che meglio si prestano a garantire riservatezza ed integrità per la trasmissione dell'intero film.

5. CONCLUSIONI E LAVORO FUTURO

L'architettura DiVA-SEC consente di distribuire Video On-Demand garantendo quei requisiti indispensabili ad una applicazione commerciale:

- autenticazione forte dell'utente, del VideoServer e del FilmServer
- integrità e riservatezza in fase di negoziazione della distribuzione di un filmato
- non-ripudio della richiesta di un film
- gestione di un meccanismo robusto per l'autorizzazione all'accesso ad un pool di FilmServer
- autenticazione del VideoClient durante il download del filmato

L'associazione tra le chiavi pubbliche e l'identità dei componenti il sistema DiVA-SEC è garantita tramite certificati X.509v3, distribuiti agli utenti attraverso browser Web standard ed usabili anche per garantire la sicurezza in altre applicazioni (posta, accesso a siti Web, trasferimento file, terminale remoto, .).

Il lavoro futuro è orientato a garantire anche la riservatezza e l'integrità dello stream audio/video. Il canale trasmissivo, attualmente implementato tramite le librerie CMT, utilizza Cyclic UDP come protocollo di livello trasporto ed occorre garantire la riservatezza e l'integrità della trasmissione senza compromettere irrimediabilmente la qualità del servizio offerto. Una possibile direzione di ricerca prevede di sfruttare le caratteristiche di codifica differenziale dei frame trasmessi: ad esempio cifrando solo i frame principali (e non quelli che riportano le differenze rispetto ai frame precedenti). Si tratta di valutare se soluzioni di questo tipo siano praticabili, come influiscano sulla qualità del servizio offerto e se sia possibile quantificare il livello di riservatezza delle informazioni trasmesse attraverso uno stream cifrato parzialmente.

RINGRAZIAMENTI

Gli autori ringraziano Marius Marian per il supporto nello sviluppo e nell'implementazione delle idee esposte in questo articolo.

BIBLIOGRAFIA

1. L.Merone, S.Russo, C.Savy, "Object Oriented Development of Distributed Multimedia Applications: the DiVA system", Congresso AICA-98, Napoli, 18-20 Novembre 1998
2. D.W.Chadwick, A.J.Young, N.Kapidzic Cicovic, "Merging and Extending the PGP and PEM trust Models - The ICE-TEL Trust Model", The Internet Society Symposium on Network and Distributed System Security, San Diego (CA), February 1997

3. H.Krawczyk, M.Bellare, R.Canetti, "HMAC: keyed-hashing for message authentication", RFC-2104, 1997
4. B. C. Smith, "Cyclic-UDP: A Priority-Driven Best-Effort Protocol," Cornell University - Internal Report, May 1994
`http://www2.cs.cornell.edu/zeno/papers/Default.html#cpbp`
5. S. Baker, "CORBA Distributed Objects using Orbix", Addison-Wesley - ACM Press, 1997
6. OpenSSL project
`http://www.openssl.org/`
7. "PKCS #12 v1.0: Personal Information Exchange Syntax," RSA Laboratories' Public-Key Cryptography Standards (PKCS), Version 1.0, June 1999
`ftp://ftp.rsa.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf`