DESENVOLUPAMENT D'UN SERVEI CRIPTOGRÀFIC PER A LA PLATAFORMA JADE

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Jaume Bigas Vence i dirigit per Carles Garrigues Olivella.

Bellaterra, Juny de 2007

El firmant, Carles Garrigues Olivella, professor del Departament d'Enginyeria de la Informació i de les Comunicacions de la Universitat Autònoma de Barcelona

CERTIFICA:

Que la present memòria ha sigut realitzada sota la seva direcció per Jaume Bigas Vence

Bellaterra, Juny de 2007

Firmat: Carles Garrigues Olivella

Dedico aquest projecte als meus pares pel seu suport durant tots aquests anys.

Agraïments

Voldria donar les gràcies al meu director de projecte, Carles Garrigues, pels consells i l'orientació que m'ha donat al llarg del projecte.

A tots els professors del Senda pels seus consells i per compartir el seus coneixements en aquest àmbit.

A tots els companys projectistes per compartir les seves experiències en el desenvolupament del projecte. Especialment a Carlos Martínez per la paciència que ha tingut esperant part d'aquest projecte.

Als meus pares pel seu suport i per la confiança que han dipositat en mi durant tots aquests anys.

I a tots els meus amics per ser-ho i per animar-me en els moments més difícils.

Índex

Índex de figures

Capítol 1

Introducció

En els últims anys Internet s'ha anat estenent i ha anat entrant a la nostra vida quotidiana cada dia més. En aquest procés Internet s'ha convertit en una eina gairebé essencial en molts sectors com la comunicació, l'oci i el món dels negocis. En aquests sectors, Internet ha servit per a trencar la barrera física que suposa la distància, fent que persones que es troben molt lluny geogràficament es puguin comunicar fàcilment com si estiguessin a la mateixa sala.

En aquest ús d'Internet el model que s'ha utilitzat habitualment és el model client/servidor. Amb el pas del temps s'ha vist que aquest model centralitzat no és capaç de cobrir totes les nostres necessitats. Avui en dia, on cada cop hi ha més informació a Internet, aquesta no es troba centralitzada en un conjunt limitat de servidors en els quals l'usuari pot fàcilment connectar-se, molts cops sense ser conscient de que ho està fent, per a aconseguir la informació desitjada o per a fer una tasca en concret.

Cada cop més, hi ha grans quantitats de dades distribuïdes en diferents punts de la xarxa. Molts cops aquestes dades no poden abandonar el lloc on es troben, ja sigui per la seva gran dimensió, per limitacions legals (per exemple en dades mèdiques) o per altres raons. En aquests entorns ens pot interessar executar el nostre programa on es troben les dades ja que aquestes no poden viatjar fins on ens trobem nosaltres. En aquest entorn són molt útils els agents mòbils.

Els agents mòbils són processos que actuen en representació d'algú i presenten un conjunt de característiques. Els agents són autònoms, ja que tenen la capacitat de prendre decisions per a aconseguir l'objectiu que se'ls ha encomanat. També són capaços de reaccionar davant de les modificacions del seu entorn i, a més a més, tenen la capacitat d'iniciar accions que el canviïn.

Els agents també són capaços de comunicar-se intercanviant informació amb altres agents o usuaris. Finalment tenen la capacitat de moure's migrant a diferents entorns per a continuar la seva execució.

Per al seu funcionament, els agents mòbils necessiten una base sobre la que executar-se. Les plataformes, o agències, són entorns d'execució que ofereixen als agents els serveis necessaris perquè aquests es puguin executar, comunicar i migrar. Els serveis que ofereixi cada plataforma variarà segons la seva finalitat.

Els agents mòbils estan compostos per 3 components principals. El codi, les dades i l'itinerari. El codi dels agents mòbils indica quines són les accions que ha d'anar realitzant al llarg de la seva execució. El codi pot ser el mateix per a totes les plataformes per les que passa l'agent, potser particular per a cada una d'elles o una combinació d'aquestes dues opcions. Les dades que porta l'agent són dades que el codi de l'agent ha d'utilitzar al llarg de la seva execució. Les dades poden ser globals per a totes les plataformes i també poden incloure dades específiques per a plataformes concretes. Per acabar, l'itinerari marca les plataformes que anirà recorrent l'agent. Aquest itinerari pot ser implícit, que estigui localitzat dins del codi, o explícit, que es trobi en una estructura a part del codi.

En els agents mòbils la seguretat és un aspecte molt important, per la vulnerabilitat dels agents respecte a atacs de les plataformes i pel risc que suposa executar agents potencialment maliciosos. Justament la seguretat és un dels principals punts on els agents mòbils fluixegen. El fet que el codi es vagi movent per un conjunt de plataformes, de les quals no sempre ens podem fiar, complica la implementació de sistemes de seguretat.

En aquesta situació els agents poden ser víctimes d'atacs que tinguin per a objectiu alterar-ne el funcionament. Aquests atacs poden anar des de la modifi-

cació de les dades perquè es donin resultats erronis, la modificació del codi per a provocar un canvi en el seu comportament o poden anar dirigits a robar les dades que porta l'agent.

En aquest entorn el nostre projecte pretén implementar el protocol de protecció presentat a [ARO04], a través del qual els agents es puguin autoprotegir dels atacs externs, protegint el seu codi i les seves dades. D'altra banda, també es pretén permetre a les plataformes identificar quan un agent ha estat modificat per una tercera part per tal d'alterar el seu comportament o per a obtenir les dades que transportava.

A través d'aquestes implementacions la nostra intenció és fer que els agents que s'executin a la plataforma JADE no estiguin tan desprotegits davant d'aquests atacs externs.

1.1 Objectius

L'objectiu principal del nostre projecte és el desenvolupament d'un servei criptogràfic de JADE, per a que es puguin implementar agents mòbils autoprotegits. Aquest objectiu el podem dividir en diferents parts:

- Dotar les plataformes de parelles de claus asimètriques i protegir la clau privada, perquè ningú extern a la plataforma la pugui obtenir.
- Implementar els mecanismes d'identificació d'agents de [ARO04] per permetre la construcció d'agents autoprotegits.
- Implementar un conjunt de funcions criptogràfiques perquè els agents puguin fàcilment xifrar/desxifrar amb clau simètrica/asimètrica, signar dades, verificar signatures, calcular i comprovar hashos, demanar la generació de claus simètriques i asimètriques.
- Facilitar als agents funcions que utilitzen la clau privada de la plataforma sense que tinguin accés directa a la clau.

1.2 Continguts

Els continguts d'aquesta memòria estan disposats en els següents capítols:

- **Agents mòbils autoprotegits:** En aquest capítol introduirem el treball presentat a [ARO04] que permet que la seguretat dels agents recaigui en ells mateixos.
- **Anàlisi i planificació:** En aquest capítol veurem quins són exactament els requisits que ha d'assolir el nostre projecte per a poder arribar a acomplir els objectius. A part, també veurem la planificació corresponent a la realització del projecte.
- **Estudi de l'arquitectura dels serveis a JADE:** En aquest capítol veurem com ha d'estar compost un servei a JADE, per a poder implementar el nostre servei.
- **Disseny:** En aquest capítol veurem com assolir els requisits que hem vist que té el nostre projecte, tenint en compte l'arquitectura dels serveis de JADE.
- **Implementació i proves:** Aquest capítol consta de 2 parts. D'una banda, veurem els detalls de la implementació del disseny realitzat i també analitzarem si hi ha hagut algun error en la fase de disseny. D'altra banda, veurem quin ha estat el conjunt de proves realitzades per a comprovar el funcionament del nostre treball.
- **Conclusions:** En aquest últim capítol veurem el grau d'assoliment dels objectius del projecte i repassarem, breument, com els hem assolit. D'altra banda, també veurem les diferents línies de millora que queden obertes en aquest projecte.

Capítol 2

Agents mòbils autoprotegits

En aquest projecte ens proposem desenvolupar un servei criptogràfic per a les plataformes Jade que faci possible la implementació d'agents mòbils autoprotegits. La construcció d'agents mòbils autoprotegits es basa en el treball presentat per Joan Ametller en l'article [ARO04]. En aquest capítol descriurem els mecanismes proposats en aquest article, per a la implementació d'un mecanisme de seguretat en el entorn dels agent mòbils, on la seguretat és un dels principals problemes.

2.1 Introducció

La seguretat en els agents mòbils és un tema encara per resoldre, seguretat que s'ha de veure des de diferents punts de vista. Per una banda s'han de protegir les plataformes dels atacs d'agents maliciosos i d'altra banda s'ha de fer la protecció inversa, protegir els agents de plataformes malicioses que vulguin manipular indegudament les seves dades o codi. Finalment, també es necessiten mecanismes per a protegir les dades i el codi d'un agent davant dels atacs d'altres agents.

D'altra banda també ens trobem que un agent al llarg del seu recorregut pot recórrer plataformes que competeixin entre elles. En aquest punt, ens hem d'asse-

gurar que cada plataforma només pugui accedir a les dades que li corresponen, i no a les corresponents a d'altres plataformes.

Aquest article ens presenta un mecanisme que permet implementar agents autoprotegits, permetent que els agents puguin gestionar el seu itinerari protegit utilitzant la clau privada de la plataforma, de manera que cada plataforma només podrà accedir a les dades corresponents.

També es proposa representar el codi, que l'agent ha d'executar a cada plataforma, com a dades, per tal de poder-lo protegir de la mateixa manera que les dades pròpiament dites.

D'altra banda trobem que quan un agent arriba a una plataforma ha de desxifrar el codi i les dades pertinents utilitzant la clau privada de la plataforma. Per a fer aquesta operació es presenten dues opcions, que s'encarregui de l'operació la plataforma o que ho faci l'agent.

Perquè la plataforma faci aquesta operació, aquesta ha de conèixer com accedir a l'itinerari de l'agent per a desxifrar les dades corresponent. Això provoca que qualsevol modificació de l'agent en aquest aspecte, suposi una modificació en l'accés a la informació per part de la plataforma, amb l'afegit que no tots els agents han de tenir el mateix sistema per emmagatzemar les dades, fet que implicaria que la plataforma els hagués de conèixer tots i ser capaç d'operar amb ells.

D'altra banda, que l'agent s'ocupi d'aquesta tasca, tot i ser la millor opció per a la interoperabilitat, també presenta els seus problemes. El principal dels quals és que l'agent ha d'accedir a la clau privada de la plataforma, fet que podria posar en perill tot el mecanisme de seguretat.

2.2 Arquitectura

La solució que es presenta és una combinació de les dues descrites anteriorment, de tal manera que afecta a la plataforma i a l'agent, però mínimament a totes

dues parts. Per una banda, la plataforma s'amplia amb un servei criptogràfic. Aquest servei té per objectiu proporcionar funcions criptogràfiques que utilitzin la clau privada de la plataforma, perquè els agents la puguin utilitzar sense tenir-hi accés directe. D'altra banda, també hem de fer que la plataforma s'asseguri que l'agent no està intentant desxifrar unes dades que hagi robat d'un altre agent. Per aconseguir-ho, el servei haurà d'incorporar un mecanisme per a assegurar que unes dades només poden ser desxifrades per l'agent propietari.

Finalment, si aquest servei es crea com un *add-on* a la plataforma, aquest no ha d'afectar a la seva estructura interna.

D'altra banda, l'agent ha de portar una part de les dades xifrades, dades que només seran accessibles a la plataforma on s'hauran d'utilitzar. Per a poder protegir el codi i que aquest només sigui accessible quan arribi a la plataforma on ha d'executar-se, l'agent ha de portar el codi com a dades, perquè pugui ser xifrat.

Per a desxifrar el codi i les dades a utilitzar a cada plataforma, l'agent portarà un codi que no anirà xifrat, el codi de control. El codi de control serà l'encarregat de, a l'arribar a cada plataforma, demanar que es desxifrin les dades i el codi corresponent a aquella plataforma, perquè l'agent pugui fer les accions pertinents en aquella plataforma.

D'aquest manera els agents passaran a tenir una estructura (C,D), on C serà el codi de control encarregat de, a l'arribar a una plataforma, agafar la porció de dades (D) corresponent a aquella plataforma, demanar que es desxifri amb la clau privada de la plataforma i que s'executi el codi corresponent.

2.3 Mecanisme de protecció

També s'explica el mecanisme dissenyat perquè les plataformes es puguin assegurar que només l'agent propietari desxifri les dades, no permetent que un agent pugui robar dades xifrades d'un altre agent i, utilitzant la clau privada de la plataforma, en conegui el contingut.

2.3.1 Autenticació de l'agent

Per a fer possible l'autenticació de l'agent, es proposa que les dades xifrades estiguin compostes de les dades, pròpiament dites, i d'un identificador del codi de control, de manera que només el codi de control corresponent pugui desxifrar les dades. Per tant les dades a xifrar queden constituïdes en forma de parella, (D,A), on A és l'identificador del codi de control. D'aquesta manera quan l'agent demani desxifrar unes dades, la plataforma seguint l'algorisme de la figura 2.1 podrà validar l'agent.

També proposen fer servir el hash del codi de control de l'agent com l'identificador que acompanya les dades. Amb aquest mecanisme s'evita que un agent pugui robar les dades i desxifrar-les, ja que quan la plataforma comprovi el hash de l'agent que està demanant que li desxifrin les dades amb el hash que porten les dades, aquests no coincidiran.

2.3.2 Protecció de dades

D'altra banda, també es presenta un solució contra la modificació de les dades d'un agent per a provocar-ne un mal funcionament. Per a protegir-se d'aquest atac es proposa fer servir un mecanisme a través del qual l'agent pugui verificar les dades abans de fer-les servir, mitjançant una signatura. Aquestes dades, són a les que anteriorment ens hem referit com D, aniran xifrades amb la clau pública de la plataforma, juntament amb el hash del codi de control de l'agent.

Per a implementar aquest mecanisme el programador de l'agent hauria de crear un parell de clau asimètrica (Pa,Sa) i signar totes les dades amb la clau privada Sa abans de xifrar-les amb la clau pública de la plataforma. Amb conseqüència les dades a utilitzar a cada plataforma, a les que anteriorment ens hem referit com (D,A), tindrien la forma ((dj,sj),H(C)), tot això xifrat amb la clau de la plataforma. En aquesta estructura dj serien les dades corresponents a la plataforma j, sj la signatura d'aquestes dades i H(C) el hash del codi de control com a identificador de l'agent. Paral·lelament, es fixa la clau pública Pa al codi de l'agent, perquè

```
dades desxifrar( dades_xifrades ) {
    contingut = desxifrar( xifrades_data,clau_privada);
    si (data_contingut es una parella (d,A) I
        es pot verificar la integritat d'A)
    {
        retornar d
    }
    }
    sinó
    {
        destruir d
        error
    }
}
```

Figura 2.1: Algorisme desxifrar amb la clau privada de la plataforma

aquest pugui comprovar la signatura a cada plataforma. Amb aquest mecanisme si la verificació falla, l'agent sabrà que les dades han estat modificades i avortarà l'execució.

2.4 Conclusió

Com hem vist, aquest article presenta un mecanisme per implementar protocols de protecció d'agents que siguin gestionats pels propis agents. Des d'aquest projecte de final de carrera es pretén proporcionar un sistema que cobreixi les necessitats de seguretat, presentades en aquest article, per a la protecció de les dades envers el codi maliciós en l'entorn de les plataformes JADE.

Capítol 3

Anàlisi i planificació

3.1 Anàlisi

Un cop hem vist en quin entorn es situa el nostre projecte, procedirem a realitzar l'anàlisi dels requisits del projecte per a poder afrontar la fase de disseny coneixent tota la seva dimensió i intentar trobar la millor solució possible per a assolir els objectius.

Com s'ha dit a la introducció, l'objectiu principal d'aquest projecte és crear un servei criptogràfic per a les plataformes JADE, per tal de dotar a aquestes d'una clau asimètrica, que pugui ser utilitzada tant per la plataforma com pels agents que la visitin per xifrar i signar les dades que desitgin.

3.1.1 Restriccions de disseny

Una de les primeres restriccions de disseny que vam veure que ens imposava el projecte és que, donat que la part principal del projecte és un servei que ha de funcionar dins de la plataforma JADE, la programació d'aquest s'ha de realitzar en el llenguatge Java, ja que tota la plataforma està realitzada en aquest llenguatge.

El fet de realitzar el projecte en Java, igual que JADE, ens assegura que aquest

serà perfectament compatible amb tot tipus de sistemes operatius o, si més no, en tots els que incorporin una màquina virtual de Java, un fet molt comú.

3.1.2 Requisits funcionals

En aquest apartat procedirem a exposar quines són les necessitats funcionals bàsiques que ha d'acabar proporcionant el nostre servei criptogràfic.

Claus asimètriques

Com ja hem introduït anteriorment un dels principals objectiu d'aquest projecte és fer que les plataformes de JADE siguin posseïdores d'un parell de claus asimètriques perquè tot l'entorn dels agents mòbils pugui utilitzar-les per a xifrar dades que només puguin ser interpretades per la plataforma en qüestió o perquè la plataforma pugui signar dades demostrant que aquestes han estat generades a la mateixa plataforma.

La clau privada de cada plataforma ha d'estar protegida de tal manera que només la plataforma pugui accedir-hi. Per tant, cap usuari del sistema ni cap agent han de poder tenir accés a aquesta clau privada.

Identificació dels agents

El servei de criptografia ha d'implementar els mecanismes proposats a [ARO04], per a garantir que els agents fan un ús correcte de la clau privada de la plataforma. Per tal d'implementar els mecanismes descrits a l'article, el servei de criptografia ha de realitzar les operacions explicades a continuació.

Un cop l'agent ha arribat a la plataforma i abans de que aquest pugui fer servir la clau privada de la plataforma, el servei ha de calcular el hash de l'agent, per a posteriorment, poder-lo comparar amb el hash que acompanyen les dades. Al generar aquest hash, el servei ha de generar un nombre aleatori i ha d'entregar-lo a l'agent del qual ha calculat el hash. Aquest nombre aleatori ha de ser secret,

ja que servirà perquè l'agent s'identifiqui davant de la plataforma quan vulgui fer servir la clau privada d'aquesta.

El servei criptogràfic haurà de guardar les parelles de hash del codi de l'agent i l'identificador corresponent per a poder-los utilitzar quan siguin necessaris. El servei haurà de guardar aquesta informació de manera que no sigui accessible per cap agent ni cap altre servei, ja que l'esquema de seguretat es basa en el nombre aleatori associat a cada hash.

Funcions criptogràfiques

Quan un agent vulgui desxifrar unes dades amb la clau privada de la plataforma, haurà d'utilitzar l'identificador secret que li ha haurà facilitat prèviament el servei criptogràfic. Juntament amb les dades a desxifrar, l'agent haurà de passar l'identificador a la funció *decrypt* del servei criptogràfic perquè aquest pugui fer les comprovacions pertinents.

El servei criptogràfic a la funció *decrypt* haurà d'utilitzar l'identificador que li ha passat l'agent, per a accedir a la taula on guarda les parelles de hash i identificador, per tal d'aconseguir el hash corresponent al codi de l'agent. Tot seguit, el servei desxifrarà les dades que li ha passat l'agent i aconseguirà el hash del codi de l'agent, introduït en la construcció de l'agent. Un cop el servei té els dos hashos, comprovarà que aquests són iguals, i només en aquest cas retornarà les dades desxifrades a l'agent.

A part de fer servir la clau privada de la plataforma per a desxifrar dades, els agents també la podran fer servir per a signar dades. Quan una plataforma signi les dades d'un agent també es pretén poder demostrar quin ha estat l'agent que ha demanat que s'efectuï la signatura. Per a poder demostrar aquest fet, el servei criptogràfic quan hagi de signar les dades li afegirà el hash del codi de l'agent. Amb aquest sistema qualsevol que vulgui verificar la signatura, a part de verificar a quina plataforma s'han signat les dades, també podrà verificar quin agent l'ha demanada.

Per a fer això possible, quan un agent vulgui que es signin unes dades amb la clau de la plataforma, haurà de facilitar a la funció *sign*, juntament amb les dades, l'identificador que prèviament li ha facilitat el servei.

El servei criptogràfic a dins de la funció *sign* haurà d'utilitzar l'identificador que li ha passat l'agent, per a accedir a la taula on guarda les parelles de hash i identificador, i així obtenir el hash corresponent al codi de l'agent. Un cop tingui el hash l'adjuntarà a les dades a signar que li ha facilitat l'agent, n'efectuarà la signatura i la retornarà a l'agent.

Per tal de completar el servei criptogràfic, aquest també oferirà un conjunt de funcions criptogràfiques que no tenen relació amb la clau privada de la plataforma. D'aquesta manera l'agent podrà fer un llarg seguit d'operacions criptogràfiques de forma senzilla.

- *Xifrar amb Clau Asimètrica*.
- *Desxifrar amb Clau Asimètrica*.
- *Xifrar amb Clau Simètrica*.
- *Desxifrar amb Clau Simètrica*.
- *Realització de Signatura*.
- *Verificació de Signatura*.
- *Càlcul de Hashos*.
- *Verificació de Hashos*.
- *Creació parell de claus asimètriques*.
- *Creació de clau simètrica*.

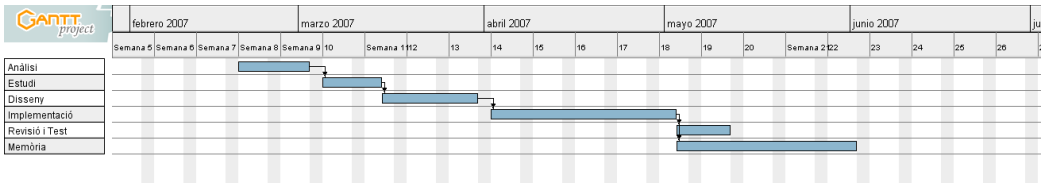

Figura 3.1: Planificació

3.2 Planificació

En aquest apartat veurem les diferents parts en les quals hem dividit el nostre projecte, quins són els objectius a realitzar cada una d'elles i el seu temps aproximat de realització.

Una consideració que s'ha de tenir en compte alhora de realitzar la planificació, es que aquest és un projecte que està pensat per a fer-se en el segon semestre, per tant la seva planificació estarà comprimida en menys temps, fet que comporta més dedicació diària a aquest. A la figura 3.1 podem veure quina és la planificació temporal del projecte.

3.2.1 Fites de la planificació

Anàlisi

A aquesta primera part del projecte li destinarem dues setmanes per a poder fer un estudi que ens permeti ser conscients de les dimensions del projecte i dels requisits exactes que aquest té. D'aquesta manera, a l'etapa de disseny, podrem trobar la solució més acurada als seus requisits.

Estudi

Abans de procedir a la realització del disseny del projecte realitzarem l'estudi de l'arquitectura dels serveis a JADE, en el qual volem adquirir el coneixement necessari per a realitzar el nostre propi servei. Hem destinat deu dies per a realitzar

aquesta tasca.

Disseny

A l'etapa de disseny estudiarem quina és la millor solució per a assolir cada un dels requisits del projecte. A la realització d'aquesta tasca hem destinat entre dues i tres setmanes.

Implementació

Un cop tenim el disseny fet, la implementació hauria de ser una part bastant automàtica. En aquest punt del projecte procedirem a fer la codificació del disseny que hem realitzat a l'etapa anterior. A aquesta part del projecte li hem destinat aproximadament un mes del temps total degut a què és una part on ens podem trobar contratemps que ens facin redissenyar algunes parts del projecte.

Revisió i test

Al final de la implementació dedicarem uns deu dies per a revisar el codi del projecte i fer les proves finals, per a assegurar-nos que el nostre projecte funciona correctament.

Redacció memòria

Per acabar hem destinat un mes de la realització del projecte a la memòria, en la qual exposarem per escrit tota la feina feta durant el projecte i explicarem el perquè de les decisions que hem pres, demostrant que aquests anys hem assimilat els coneixements necessaris.

Capítol 4

Estudi de l'arquitectura dels serveis de JADE

4.1 Introducció

La plataforma d'agents JADE està pensada per ser una plataforma distribuïda composta d'*Agent-Containers*, contenidors d'agents. Als contenidors d'agents és on s'executen els agents. Cada plataforma pot estar formada per diferents contenidors situats, possiblement, a diferents ordinadors connectats a través d'una xarxa.

A JADE no tots els contenidors són iguals. A cada plataforma un dels *Agent-Containers* és diferent dels altres, el *Main-Container*. El *Main-Container* té algunes característiques que el fan diferent de la resta d'*Agent-Containers*, com pot ser en l'àmbit de la migració. La migració és el moviment d'un agent entre diferents contenidors. En l'entorn dels agents mòbils ens trobem amb 2 tipus de migració, la migració intra-plataforma i la migració inter-plataforma. La migració intra-plataforma és el tipus de migració en la que un agent es mou entre contenidors d'una mateixa plataforma. D'altra banda la migració inter-plataforma es la que es produeix quan un agent és mou d'una plataforma a una altra. A diferència de la migració intra-plataforma, que es pot efectuar entre qualssevol contenidors,

la migració inter-plataforma només es pot realitzar entre *Main-Containers*.

En les versions de JADE 2.61, i anteriors, els diferents contenidors d'una mateixa plataforma es comunicaven utilitzant RMI (Remote Method Invocation). Aquesta comunicació es realitzava definint una interfície remota, de tipus RMI, que contenia tots els mètodes necessaris perquè els contenidors poguessin interoperar. El problema que presentava aquest sistema és que qualsevol modificació o ampliació de la funcionalitat de la plataforma, que requerís comunicació entre contenidors, comportava la modificació d'aquesta interfície RMI.

A partir de la versió 3.1 JADE modifica aquest punt, eliminant aquesta interfície i fent que la funcionalitat de la plataforma estigui basada en serveis. Un servei és un sistema per oferir un conjunt de funcionalitats als agents i a la resta de serveis, des de dins de la plataforma. La incorporació de serveis facilita que els programadors que no pertanyen a JADE puguin ampliar la funcionalitat de JADE, creant els seus propis serveis.

4.2 Els serveis

Els serveis fan servir per a comunicar-se un sistema que no està basat en interfícies, com era la comunicació fins ara, sinó que es comuniquen a través de comandes. Una comanda és el mètode que s'utilitza en els serveis per a demanar una funcionalitat d'un servei concret. Les comandes serveixen tant per a comunicar un mateix servei a diferents contenidors, com per a comunicar diferents serveis dins del mateix contenidor. Cada una d'aquestes comandes està definida pel servei al que pertany, que serà l'encarregat de implementar-ne la funcionalitat associada. Amb aquest sistema quan un servei afegeixi una nova funcionalitat, només cal que en defineixi la comanda associada. A partir d'aquest punt, la resta de serveis només s'hauran de modificar si volen fer servir la nova comanda, si no és així podran fer servir la seva implementació, com fins al moment.

Perquè un servei existeixi en un contenidor, cal especificar-lo quan es posa en marxa el contenidor. Això s'ha de fer encara que el contenidor sigui de tipus

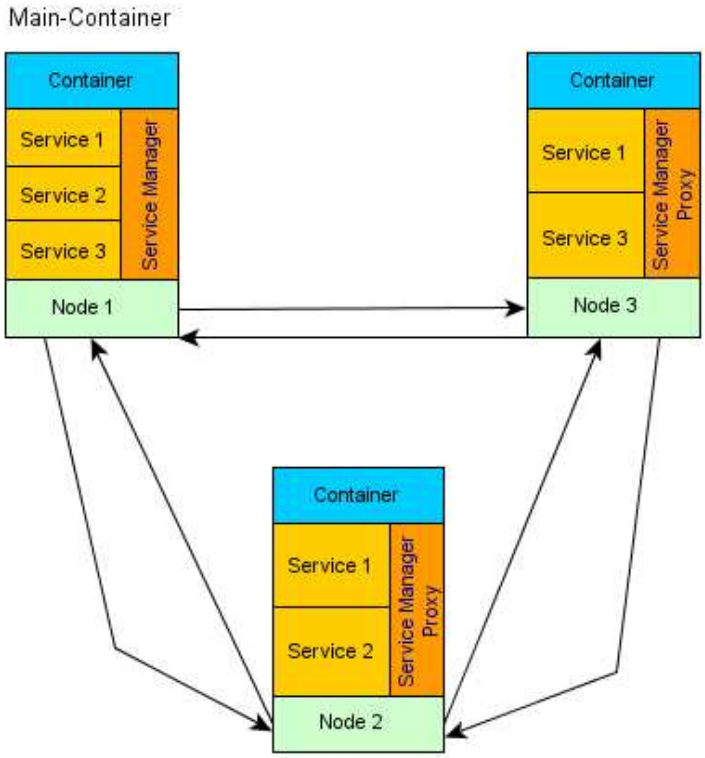

Figura 4.1: Diferents contenidors que formen una sola plataforma

Agent-Container. Aquesta necessitat és deguda a que dins d'una plataforma no tots els contenidors han de tenir els mateixos serveis, com podem veure a la figura 4.1.

4.2.1 Comandes

Com ja hem comentat anteriorment en l'entorn dels serveis, la comunicació entre diferents contenidors i entre diferents serveis del mateix contenidor es fa a través de comandes. Una comanda està composta pel nom i una llista de paràmetres. El nom està representat amb un *String* que porta associat una funcionalitat dins del servei propietari i els paràmetres són utilitzats per aquesta funcionalitat. JADE ja incorpora mètodes per processar les comandes i per facilitar que es pugui donar el

valor de retorn al servei que ha cridat la comanda.

De comandes a JADE n'hi ha de 2 tipus, les comandes verticals i les comandes horitzontals. Les comandes verticals són les comandes que viatgen dins d'un contenidor, i les comandes horitzontals són aquelles que viatgen entre diferents contenidors d'una mateixa plataforma.

4.2.2 Filtres

Els filtres són utilitzats per a detectar quan es produeixen certes comandes verticals i perquè el servei pugui actuar en conseqüència a aquestes comandes. Les comandes a interceptar poden ser comandes del mateix servei o de qualsevol altra. Quan un filtre intercepta una comanda, després de processar-la, pot decidir que la comanda segueixi el seu curs normal, o en canvi, pot eliminar la comanda, que ja no seguirà el seu camí a través de la resta de filtres.

De filtres n'hi ha de 2 tipus, el *Incoming Filter* i el *Outgoing Filter*, que es diferencien en quines comandes intercepten. El *Incoming Filter* intercepta les comandes que venen provocades per accions en contenidors remots, d'altra banda, el *Outgoing Filter* intercepta les comandes generades en el contenidor on es troba el filtre. En la figura 4.5 podem veure aquests 2 tipus de filtres.

4.2.3 *Sinks*

Els *sinks*, al igual que els filtres, s'encarreguen de processar les comandes verticals. La principal diferència és que els *sinks* només poden processar les comandes del propi servei, en canvi, els filtres poden processar les comandes de tots els serveis. És per això que normalment els filtres s'acostumen a destinar únicament a les comandes dels altres serveis. A la figura 4.2 veiem com una comanda vertical travessa tots els filtres, però acaba només en un sol *sink*, el *sink* del servei propietari de la comanda, sempre i quan cap filtre no hagi eliminat la comanda prèviament com fa el filtre del Servei 2 a la figura 4.3.

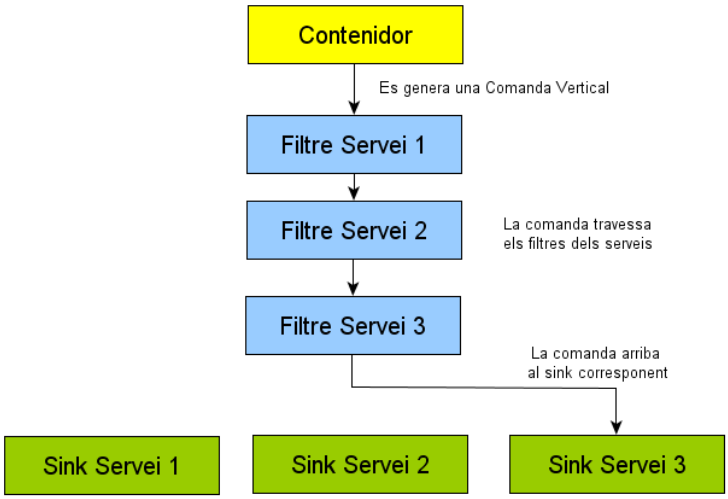

Figura 4.2: Recorregut d'una comanda vertical

Com en el cas dels filtres també tenim 2 tipus de sinks, el *Target Sink* i el *Source Sink*, que es diferencien en quines comandes processen. El *Target Sink* processa les comandes que venen provocades per accions en contenidors remots, després de travessar els *Incoming Filters*, en canvi, el *Source Sink* processa les comandes generades en el contenidor on es troba el *sink* després de travessar els *Outgoing Filters*. En la figura 4.5 podem veure els 2 tipus de *sinks* existents.

4.2.4 Processador de comandes

Un cop haguem definit els nostres *sinks* i/o filtres, el Processador de comandes s'encarregarà de gestionar-los per tal que les comandes pertinents arribin fins a ells. Perquè això sigui possible el programador del servei haurà d'implementar les funcions *getCommandFilter* i *getCommandSink*, en la classe *Service* del servei, com veurem més endavant en aquest estudi. Definint aquestes funcions el Processador de comandes ja podrà accedir als *sinks* i filtres, per a gestionar-los correctament.

Quan s'engegui el servei, el Processador de comandes agafarà els filtres a

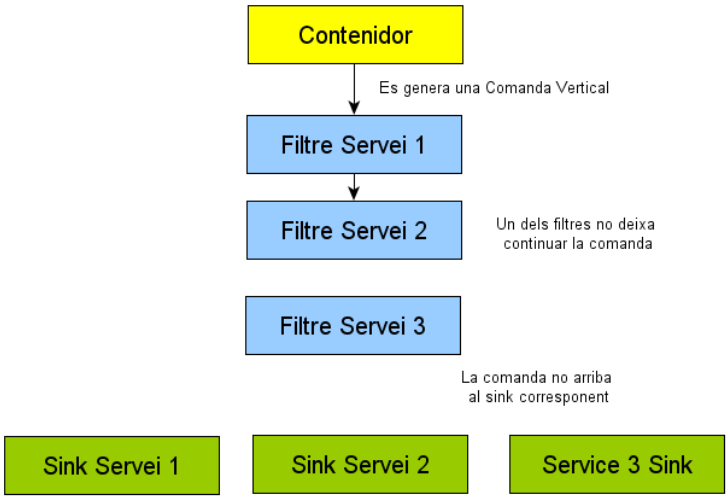

Figura 4.3: Comanda vertical eliminada per un filtre

través de la funció corresponent, i s'encarregarà de situar-los a la cua de filtres corresponent. D'altra banda també registrarà els *sinks*, perquè la plataforma en conegui l'existència. Per tant, quan es generi una comanda vertical, el Processador de comandes la farà passar per la cua de filtres corresponent, on es trobarà el nostre filtre, i finalment s'encarregarà de que la comanda finalitzi al *sink* del servei propietari de la comanda, si cap filtre no l'ha eliminat prèviament.

4.2.5 *Service Manager*

El *Service Manager* és la part de JADE encarregada de donar accés als diferents serveis de la plataforma. Per fer-ho s'encarrega de donar d'alta els serveis d'un contenidor i indicar en el *Main-Container* quins serveis estan actius a cada Contenidor. L'altra principal utilitat del *Service Manager* és la de proporcionar el *Slice* o el *Service* d'un servei d'un altre contenidor, per a poder utilitzar la seva funcionalitat. El *Service Manager* es troba únicament en el Main-Container, en canvi, la resta de contenidors només tenen un proxy al *Service Manager* del *Main-Container*. Aquest últim fet el podem veure gràficament a la figura 4.1.

4.2.6 Gestió de comandes horitzontals

Les comandes horitzontals es tracten de forma diferent a les comandes verticals, a més a més, en aquest tractament no hi intervenen ni filtres, ni *sinks*, com a les comandes verticals. En el tractament de les comandes horitzontals hi intervenen tres parts diferenciades. El *SliceProxy*, la part encarregada d'enviar les comandes horitzontals cap als altres contenidors, el Node, que s'encarrega de rebre les comandes horitzontals provinents d'un altre contenidor, i el *Slice*, la part encarregada de processar les comandes horitzontals que rep el Node.

El *SliceProxy* per tal d'enviar les comandes horitzontals ha de seguir un procés concret. Inicialment, un *sink* o un filtre ha de demanar al *SliceProxy* una funcionalitat d'un altre contenidor. Per atendre aquesta petició el *SliceProxy* ha de crear la comanda horitzontal, indicant de quin tipus és i quins paràmetres portarà. A continuació, agafa una instància del Node del contenidor que ha de rebre la comanda i li envia la comanda horitzontal. Com a retorn d'aquesta crida, quan la comanda sigui totalment processada, el *SliceProxy* rebrà el valor de retorn, que s'enviarà des de l'altre contenidor com a resultat de la comanda horitzontal en qüestió. Un cop hagi rebut aquest valor el retornarà al component del servei que l'hi hagi demanat que efectués aquest procés.

Per la seva banda, el Node un cop hagi rebut la comanda horitzontal enviada des d'un *SliceProxy*, s'encarregarà de passar-li al *Slice* del servei al que correspon la comanda horitzontal, perquè en pugui fer el tractament adequat. El Node és una part de JADE que ja està creada i és única a cada contenidor, per tant cada servei no ha d'implementar el seu Node.

Un cop el *Slice* rep una comanda horitzontal ha de decidir com la tracta. El *Slice* pot tractar les comandes horitzontals de diferents maneres, ja sigui executant la funcionalitat relacionada localment o generant una comanda vertical perquè el *Target Sink* corresponent, executi la funcionalitat adequada. D'aquesta segona manera es dona la possibilitat a altres serveis perquè actuïn amb conseqüència segons la comanda en qüestió.

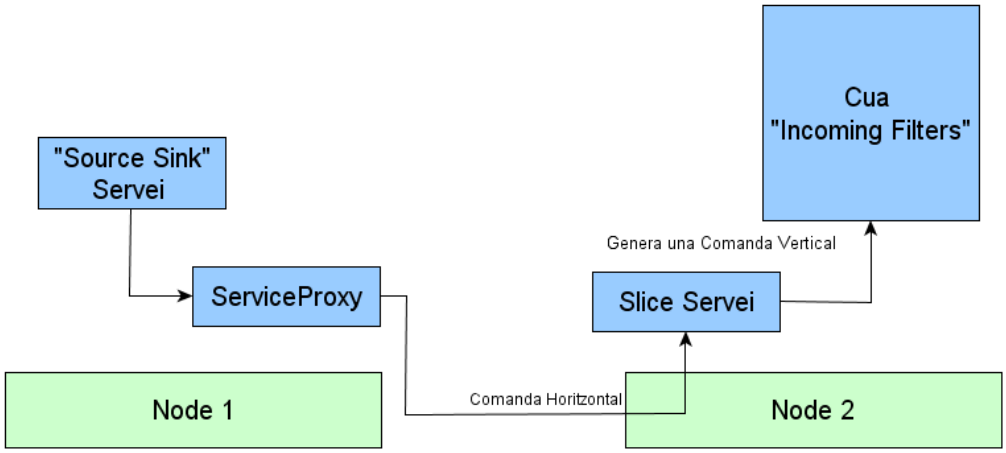

Figura 4.4: Recorregut d'una comanda horitzontal

En la figura 4.4 podem veure un exemple gràfic del procés que hem comentat en aquest apartat. En aquesta figura apreciem com el *SliceProxy* genera la comanda horitzontal i la envia cap al Node, que quan la rep, la proporciona al *Slice* del servei corresponent. A més a més, en aquest cas concret, el *Slice* genera una comanda vertical, com a resultat de la comanda horitzontal rebuda.

4.2.7 Interacció de les parts del servei

A la funcionalitat dels altres serveis s'hi accedeix a través de comandes. Quan un servei necessita la funcionalitat d'un altre servei, genera la comanda vertical associada. Aquesta comanda travessa els *Outgoing Filters* de tots els serveis i acaba en el *Source Sink* del servei propietari de la comanda, sempre i quan la comanda no hagi estat eliminada per un filtre. Mentrestant, quan els filtres són travessats per una comanda poden decidir si fer una acció com a conseqüència de la comanda vertical trobada o no.

Si quan la comanda vertical arriba al *sink*, aquest veu que la comanda requereix la intervenció d'un altre contenidor, fa que el *ServiceProxy* enviï una comanda horitzontal. Aquesta comanda horitzontal arriba al Node de l'altre contenidor, qui facilita la comanda al *Slice* del servei corresponent. Aquest *Slice* pot executar ell

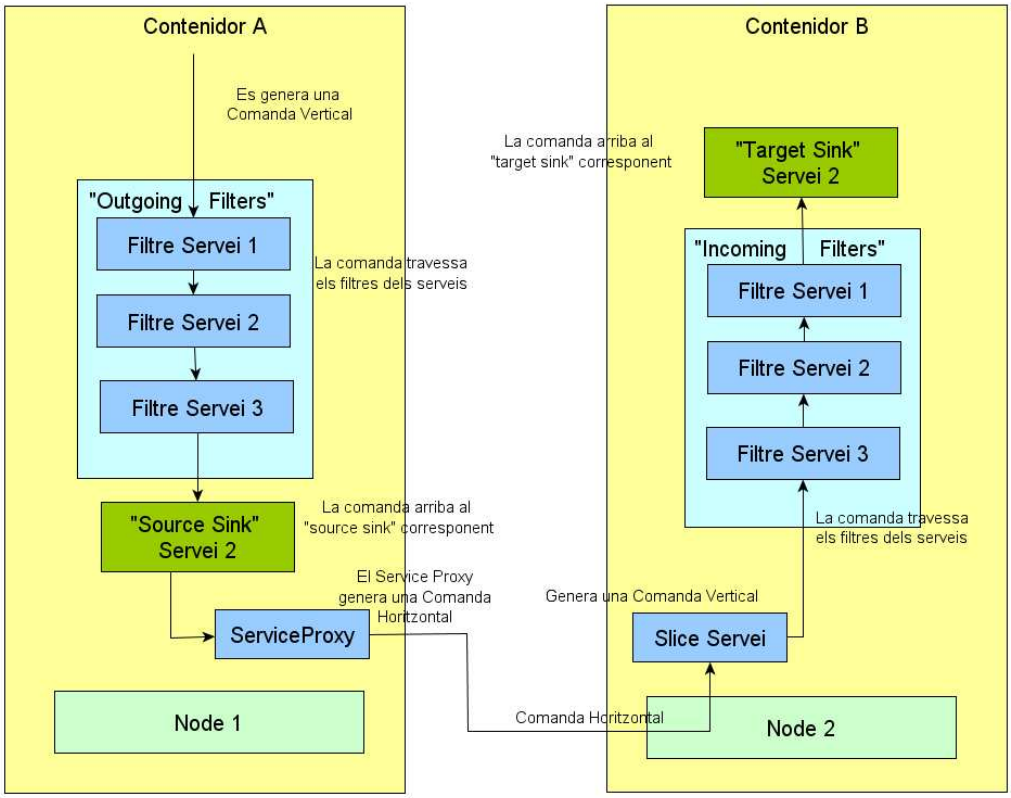

Figura 4.5: Comunicació entre contenidors

la funcionalitat o crear una comanda vertical.

Si es crea una comanda vertical, aquesta travessa els *Incoming Filters* i acaba en el *Target Sink* del servei corresponent, sempre i quan tampoc hagi estat eliminada prèviament. Com els *Outgoing Filters*, els *Incoming Filters* també poden fer accions a conseqüència de la comanda vertical. Quan la comanda vertical arriba al sink, aquest realitza les operacions adequades i produirà el valor de retorn, que es retornarà al servei que hagi generat la comanda inicial. Si les comandes han estat abans tractades pel *Source Sink* o pel *Slice* aquests seran els qui produiran el valor de retorn. El recorregut global que acabem de descriure es pot apreciar a la figura 4.5.

4.2.8 Helper

Fins al moment hem estat parlant de com els serveis que es troben en els contenidors es comuniquen entre sí, però en canvi no hem vist com els agent mòbils poden accedir a les funcionalitats dels diferents serveis. Els agents mòbils utilitzen aquestes funcionalitats a través del Helper.

El Helper és una interfície on el servei exposa tota la funcionalitat que ofereix als agents. A més a més, de la mateixa manera que fa amb les comandes, ha d'implementar tota la funcionalitat associada a aquestes crides. Amb aquest esquema, l'ampliació d'un servei amb noves comandes es pot fer de forma totalment transparent a la resta de serveis i agents.

Perquè els agents puguin accedir al Helper, JADE ja incorpora un sistema a través del qual, especificant el nom del servei desitjat, la plataforma retorna una instància del Helper per a poder-ne utilitzar la funcionalitat desitjada.

4.3 Com implementar un servei

Per acabar, un cop hem explicat quines són les diferents parts que formen un servei i com funcionen, a continuació veurem quina és la mínima implementació necessària per a realitzar un servei.

Service class

Aquesta és la classe central del Servei. Aquesta classe ha d'implementar els mètodes mínims necessaris perquè JADE es pugui comunicar i pugui afegir el nostre servei amb la resta dels serveis. A part també es necessari que implementi 3 tipus d'*inner classes*.

Els mètodes a definir són els següents:

- *init*: Mètode que es crida al arrencar el Servei. En ell podem afegir totes les

accions que volem que es facin al principi del servei.

- *getName*: Mètode que retorna el nom del Servei.
- *getHorizontalInterface* : Mètode que retorna l'interfície *Slice* del servei.
- *getLocalSlice* : Mètode que retorna la classe encarregada de processar les comandes horitzontals que es reben d'un altre contenidor.
- *getCommandFilter* : Mètode que retorna els Filtres que hi ha declarats en el servei, si n'hi ha.
- *getCommandSink* : Mètode que retorna els *Sinks* del servei.
- *getOwnedCommands* : Mètode que retorna una llista amb totes les comandes que formen part del servei.
- *boot* : Aquest mètode no és imprescindible definir-lo. Si existeix, es crida un cop tot el servei ja ha estat arrencat.

Com ja hem dit, apart de tots aquest mètodes, s'han d'implementar 3 tipus de classes internes. D'una banda s'han d'implementar els 2 *Sinks* dels quals hem parlat fins ara, el *Target Sink* i el *Source Sink*, per a processar les comandes verticals. Com ja hem comentat anteriorment, el *Source Sink* s'encarrega de processar les comandes verticals generades al mateix contenidor, ja sigui tractant-les ell mateix o creant comandes horitzontals perquè siguin tractades en un altra contenidor. Per la seva banda, el *Target Sink* processa les comandes verticals creades a partir de funcionalitats demanades en altres contenidors.

Els *sinks* obligatòriament han d'incloure un mètode, el *consume*. Aquest mètode serà l'encarregat de tractar les comandes verticals que rebi com a paràmetre, ja que, serà a través d'aquest mètode com el *Service Manager* farà arribar la comanda al *sink*.

Tot i ser una part important del Servei, els *Filtres* no són obligatoris. Podem definir filtres per a interceptar comandes d'altres serveis o no, depenent de les

necessitats del nostre servei. Si no necessitem interceptar cap comanda vertical d'un altre servei, aleshores no és necessari definir cap dels 2 filtres.

Els filtres que es decideixin implementar, hauran de contenir el mètode *accept*. En aquest mètode és en el qual el filtre rebrà la comanda vertical quan aquesta viatgi per la cua de filtres. Per tant, aquest serà el mètode en el qual el filtre haurà d'interceptar les comandes que desitgem. Aquest mètode com a valor de retorn tindrà un booleà, valor que indicarà si la comanda vertical ha de seguir el seu camí a través de la cua de filtres, o si per altra banda, ha de ser descartada.

Per últim, dins de la classe Service també s'ha d'implementar la classe ServiceComponent, la qual hereta de la classe Slice. Aquesta classe és necessària, ja que és l'encarregada de rebre les comandes horitzontals provinents d'altres serveis i transformar-les en comandes verticals. Per tant, sense aquesta classe no podríem tenir comunicació entre diferents contenidors en el nostre servei.

Aquesta classe haurà d'implementar 3 mètodes, el *getNode*, *getService* i el *serve*. El *getNode* i el *getService* hauran de retornar el Node del Contenidor i la classe *Service* respectivament. Per la seva banda, el *serve* serà el mètode que rebrà les comandes horitzontals, que li passarà el Node, i ha de retornar la comanda vertical a executar, en cas necessari.

Slice class

D'una altra banda, per a implementar un nou servei, també hem de definir la interfície *Slice*. El *Slice* és la interfície on es declaren els diferents mètodes del Servei que podran ser cridats des d'un altre contenidor.

Proxy class

El *Proxy* és la classe encarregada d'implementar l'anterior interfície *Slice* i per tant s'encarrega d'implementar els mètodes que podran cridar la resta de contenidors. Per a fer això possible aquesta classe haurà de crear les diferents comandes horitzontals i passar-les al Node.

Un cop vist com ha d'estar estructurat un servei i quines són les classes mínimes necessàries perquè aquest funcioni correctament, ja estem en condicions de començar a dissenyar el nostre propi servei de JADE.

Capítol 5

Disseny

Un cop tenim clars quins són els objectius principals dels projecte i quines són les funcionalitats que aquest ha de tenir ja podem passar a realitzar el disseny, seguint l'esquema que ha de tenir un servei de JADE, segons el que hem vist en el capítol anterior.

5.1 Funcionament del servei de JADE

5.1.1 Claus asimètriques

Com ja hem comentat en la part de l'anàlisi, un aspecte molt important en aquest projecte és la creació i la gestió del parell de claus asimètriques de la plataforma, sobretot en l'aspecte de la protecció de la clau privada. En aquest apartat s'ha de parar atenció principalment a dos punts: quin tipus de clau utilitzar i com protegir la clau d'atacs externs que tinguin com a objectiu robar la clau privada.

Vam començar decidint quin seria el tipus de clau que volíem que posseïssin les plataformes amb el nostre servei. Per a fer aquesta tria vam estar analitzant quins eren els principals tipus de claus asimètriques que es feien servir i també quines eren les claus que més proveïdors criptogràfics de Java oferien. En aquest

anàlisi vam veure que els principals tipus de claus asimètriques eren les claus RSA i DSA(Digital Signature Algorithm). Després de veure que ens oferien cadascuna, vam veure que les úniques que ens oferien totes les possibilitats que necessitàvem eren les claus RSA, ja que les claus DSA són claus destinades a signar dades.

Un cop teníem seleccionat el tipus de clau a fer servir, havíem de passar a triar un paràmetre molt important d'aquestes claus, la mida. La mida de la clau fixa quina és la limitació que tindrà la clau alhora de xifrar/desxifrar dades d'una sola passada. Com més llarga sigui la clau més quantitat de dades podrà xifrar. En aquest punt vam observar que les claus més freqüents anaven dels 512 fins als 4096 bits. Tot i que les claus poden arribar fins a 4096 bits, aquesta mida no és oferta per tots els proveïdors, per tant ens vam quedar amb la mida immediatament inferior, 2048 bits. Amb aquesta tria el màxim de dades a xifrar de cop ens quedava limitat a 245 bytes.

Un cop ja tenim el tipus de clau a fer servir, ara ens toca centrar-nos en com protegir la clau privada de la plataforma. Per a protegir la clau privada quan la guardem al disc, és clar que hem de fer servir algun sistema de xifratge. El problema és que no podem fer servir un sistema de xifratge basat en una clau, ja que després tindríem el mateix problema per protegir aquesta segona clau. Després d'analitzar diferents sistemes per protegir la clau, vam decidir que el sistema més adequat era mitjançant un PBE.

Un PBE, Password-Based Encryption, és un sistema de xifratge que fa servir una contrasenya introduïda externament per a xifrar i desxifrar les dades. A partir de la contrasenya, la *salt*, que és un conjunt de bytes i el número d'iteracions que es desitgin, es genera la clau amb la qual podem xifrar i desxifrar les dades, com podem veure a la figura 5.1. La *salt* i el nombre d'iteracions són paràmetres que s'han de fixar en el codi alhora d'implementar-lo. A partir de les dades xifrades no es pot reconstruir la contrasenya de cap de les maneres, per tant només qui conegui la contrasenya podrà desxifrar les dades. Un altre detall important és que el PBE és un sistema de clau simètrica per tant la contrasenya serà la mateixa tant per xifrar les dades com per desxifrar-les.

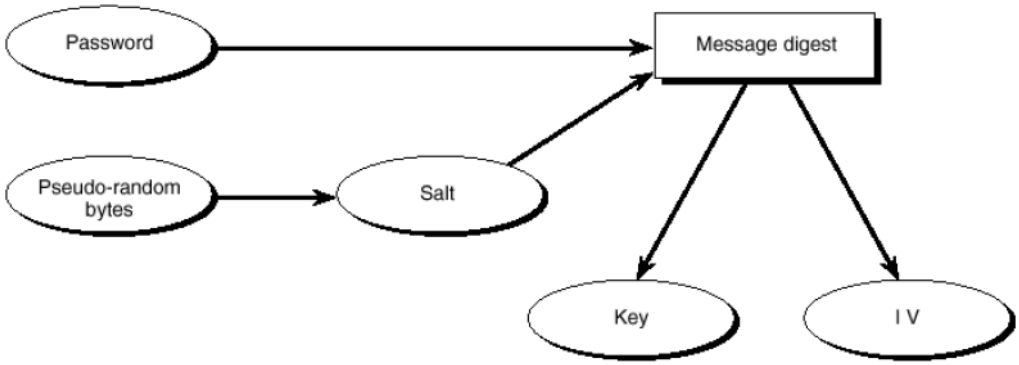

Figura 5.1: Password-Based Encryption

La utilització del PBE com a sistema per a protegir la clau privada comporta decidir qui coneixerà aquesta clau i la introduirà per teclat en el moment en què aquesta clau s'hagi de desxifrar. En aquest punt hem decidit la creació d'un rol que seria l'administrador del servei, en la majoria de casos l'administrador de la plataforma, que seria qui sap la contrasenya per desxifrar la clau.

Quan generem la clau aquesta es trobarà en un sol ordinador. Com hem vist anteriorment una plataforma pot estar formada per diferents contenidors i ens interessa que els agents puguin fer servir la clau privada de la plataforma independentment de a quin contenidor es trobin. No podem permetre que aquesta s'hagi de trobar físicament en tots els ordinadors on hi hagi un contenidor que pertanyi a la plataforma, ja que podríem trobar-nos amb problemes d'inconsistència, en els quals diferents contenidors tinguessin diferents versions de la clau.

Per aquestes raons, vam decidir que la utilització de la clau privada havia de ser centralitzada, perquè només s'hagi de carregar i introduir la contrasenya una sola vegada. Per a centralitzar la clau privada vam decidir que aquesta només estigués en el *Main-Container*. D'aquesta manera quan s'arrenca el servei en el *Main-Container* l'administrador introdueix la contrasenya, la clau es desxifrada i queda carregada a memòria per a la seva futura utilització. Quan la resta de contenidors vulguin utilitzar la clau privada de la plataforma hauran de demanar al *Main-Container* que realitzi per ells les funcions corresponents.

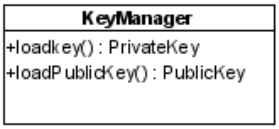

Figura 5.2: Classe KeyManager

Per a la creació, gestió i protecció de les claus de la plataforma crearem una classe específica, la KeyManager (Figura 5.2).

5.1.2 Identificació dels agents

A l'anàlisi hem vist que, per poder aplicar el mecanisme que es presenta a [ARO04], es requereix que el servei calculi quin és el hash del codi de l'agent. Aquest hash es necessari ja que quan un agent demani que se li desxifrin unes dades amb la clau privada de la plataforma, la plataforma ha de comprovar, utilitzant aquest hash, que les dades són d'aquell agent.

Identificador

En l'anàlisi vam veure que quan el servei calcula el hash del codi de l'agent, també ha de generar un identificador únic (CCHKey), que facilitarà a l'agent. Un cop hagi calculat aquesta informació el servei es guardarà internament aquestes dues dades juntament amb el *AID*, l'identificador de l'agent a les plataformes. D'aquesta manera podrà saber quin hash i CCHKey corresponen a cada agent. El fet de guardar la informació en la que es basa la seguretat conjuntament amb el *AID*, informació que tothom pot saber sobre un agent, no és un problema de seguretat ja que aquesta taula és d'ús exclusiu del nostre servei i no se li permetrà l'accés a ningú que no sigui el propi servei.

Aquesta informació es guardarà en un objecte d'una classe definida per nosaltres el CCHKeyTable. Aquesta classe conté dues *hashtable*, una per obtenir el CCHKey a partir del *AID* i una altre per obtenir el hash del codi de l'agent a partir

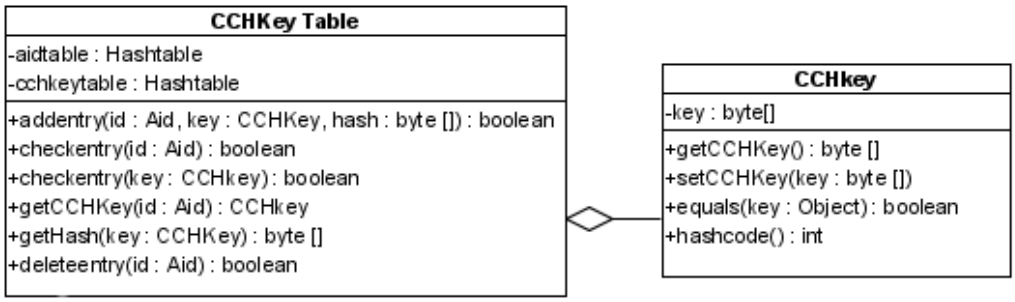

Figura 5.3: Classes de CCHKey

del CCHKey. A la figura 5.3 podem veure les classes CCHKeyTable i CCHKey.

Perquè l'esquema no es pugui trencar fàcilment, vam decidir que l'identificador generat pel servei fos un conjunt de 16 *bytes* generats de forma aleatòria. La llargada d'aquest identificador ens assegura que un atac per la força bruta tingui mínimes possibilitats de prosperar, ja que ens trobem davant de 2 elevat a 128 possibles identificadors, un número molt gran.

Perquè el CCHKey no pugui ser interceptat per cap agent extern, facilitarem el CCHKey a l'agent a través d'un mètode d'aquest. Per aquesta raó, perquè l'agent pugui demanar el CCHKey i fer servir la clau privada de la plataforma, l'agent haurà d'implementar el mètode *setCCHKey(SignedObject)* de la interfície CryptoAgent(Figura 5.4), interfície que nosaltres definirem. Aquesta interfície només constarà d'aquest mètode, el qual rebrà com a paràmetre un SignedObject que contindrà el CCHKey. Un SignedObject és un objecte que conté unes dades i la signatura d'aquestes dades, signatura realitzada utilitzant una clau privada i un algorisme subministrats a la seva creació. El fet que el CCHKey estigui dins un SignedObject servirà perquè l'agent pugui comprovar que el CCHKey prové de la plataforma abans d'emmagatzemar-lo.

Quan un agent ens demani la generació del CCHKey a través del mètode *generateCCHKey(AID)* el contenidor on es faci la petició passarà la petició al *MainContainer*, que generarà el CCHKey aleatòriament i el retornarà al contenidor on es troba l'agent. Un cop rebi aquest CCHKey el contenidor agafarà una instància

Figura 5.4: Interfície CryptoAgent

de l'agent a partir del seu *AID* i li passarà el CCHKey amb el mètode comentat anteriorment.

Vam decidir subministrar el CCHKey a l'agent amb aquest sistema, ja que d'aquesta manera ens assegurem que un agent no pugui obtenir el CCHKey d'un altre agent. Encara que un agent demani el CCHKey d'un altre agent, el CCHkey serà lliurat a l'agent correcte.

Donat que els agents poden haver canviat de contenidor dins de la mateixa plataforma, entre el moment de demanar el CCHKey i el moment de voler fer servir la clau privada de la plataforma no podem fer que cada contenidor guardi la seva pròpia taula de hashos. Per a això, vam decidir que la creació i la gestió dels CCHKeys i dels hashos estigués centralitzada en el *Main-Container*. D'aquesta manera cada cop que s'hagi d'efectuar una operació que requereixi l'accés a la CCHKeyTable aquesta haurà de ser efectuada en el *Main-Container*.

Quan un agent marxi de la plataforma no ens interessa seguir emmagatzemant el seu hash i el seu CCHKey, ja que fins que no torni a la plataforma, si ho fa, aquesta informació no es tornarà a fer servir. A més a més, si ens anéssim quedant amb tota aquesta informació, implicaria que cada cop que un agent demanés el CCHKey aquesta taula aniria creixent, i cada cop aniria acumulant més informació que no es faria servir.

Per evitar aquest problema, i també, per no deixar una possible porta oberta a atacs, vam trobar adequat que quan un agent marxi de la plataforma o mori en ella, s'elimini la seva entrada de la taula CCHKeyTable. D'aquesta manera cada cop que un agent arribi a la plataforma i vulgui fer servir la clau privada d'aquesta

haurà de demanar de nou el CCHKey, el qual es generarà cada cop de nou. Aquest sistema també evita altres problemes, ja que la classe del codi de control de l'agent no té perquè ser sempre la mateixa i per tant el seu hash tampoc serà el mateix.

Càlcul del hash

Quan un contenidor, com a resposta a una crida al mètode *generateCCHKey*, demani al *Main-Container* que generi un CCHKey, aquest contenidor a part de generar-lo calcularà el hash de l'agent. Per a calcular el hash del codi de control de l'agent necessitem saber quin és aquest codi concret, per tant necessitem que l'agent ens informi de quina és aquesta classe. Per a fer aquesta tasca vam decidir aprofitar la possibilitat de llençar els agents com a JAR, perquè a través d'aquest JAR el programador de l'agent pugui informar de quina és la classe del codi de control.

Aquesta informació, ens la facilitarà el programador a través del Manifest del JAR de l'agent. En aquest manifest el programador haurà d'especificar la classe del codi de control a través de la entrada *Main-Class*, posant el nom de la classe sense l'extensió *.class*. Aquesta entrada podrà ser llegida des del servei. D'aquesta manera el servei sabrà quina es la classe principal i podrà calcular-ne el hash.

Abans de poder calcular el hash el servei ha d'identificar quin és el JAR de l'agent entre tots els que es trobin a la plataforma. Per trobar el JAR vam decidir utilitzar el *Inter-Platform Mobility Service*, en aquest servei es guarda en una taula el *AID* de cada agent acompanyat de la ubicació del JAR de l'agent. Per tant per a poder aconseguir el JAR de l'agent necessitem que aquest ens faciliti el seu *AID* per a partir del *Inter-Platform Mobility Service* poder obtenir la ubicació del JAR.

Un cop ja tenim la ubicació de l'agent podrem mirar quina és la seva classe principal a través del manifest i llegir l'arxiu corresponent de dins del JAR.

Per a poder calcular el hash del codi de l'agent ens quedava decidir quin algorisme fer servir per a calcular-lo. Per a fer aquesta tria vam analitzar quins són els millors algorismes de hash. Vam veure que els algorismes més freqüents són el

MD5 i el SHA en les seves diferents versions. D'aquests vam descartar el MD5 i els SHA-0 i SHA-1, ja que són algorismes en els quals ja s'han trobat col·lisions. Finalment vam triar el SHA-512 que és l'algoritme de SHA que retorna un hash més llarg i per tant el que, en principi, té menys problemes de col·lisió.

5.1.3 Funcions criptogràfiques

Com ja vam veure a l'anàlisi de requisits les funcions criptogràfiques són l'essència del projecte, sobretot les relacionades amb la clau privada de la plataforma. Com hem dit a l'anàlisi, per a completar el servei a part de les funcions que utilitzen la clau privada de la plataforma, aquest oferiria un gran ventall de funcions relacionades amb la criptografia. Aquestes funcions les implementarem perquè el programador de l'agent només s'hagi de preocupar d'indicar els paràmetres desitjats a la crida i nosaltres ja els farem servir adequadament al llarg de l'operació.

El nostre servei oferirà les següents funcions criptogràfiques, de les quals les 2 primeres són l'essència del servei.

- *Desxifrar amb la clau Privada de la Plataforma*
- *Signar amb la clau Privada de la Plataforma*
- *Xifrar Clau Asimètrica*: Xifrar les dades amb la clau asimètrica subministrada per l'usuari, *Asymmetric_Encrypt(byte[], Key, String):byte[]*.
- *Desxifrar Clau Asimètrica*: Desxifrar les dades amb la clau asimètrica subministrada per l'usuari, *Asymmetric_Decrypt(byte[], Key, String):byte[]*.
- *Xifrar Clau Simètrica*: Xifrar les dades amb la clau simètrica subministrada per l'usuari, *Symmetric_Encrypt(byte[], SecretKey, String):byte[]*
- *Desxifrar Clau Simètrica*: Desxifrar les dades amb la clau simètrica subministrada per l'usuari, *Symmetric_Decrypt(byte[], SecretKey, String):byte[]*.

- *Signatura*: Signatura de les dades facilitades per l'usuari a partir de la clau privada, també facilitada per ell, *Sign(byte [], PrivateKey, String):byte[]*.
- *Verificació de Signatura*: Comprovació de si la signatura correspon a les dades i Clau Pública subministrades, *Verify(byte [], byte [], PublicKey, String):boolean*.
- *Realització de Hashs* : Càlcul del hash d'unes dades utilitzant l'algorisme facilitat per l'usuari, *Hashof(byte[], String):byte[]*.
- *Verificació de Hashs* : Aquesta funció es podrà realitzar de dues maneres, comprovant si 2 hashos són iguals *CheckHash(byte[], byte[]):boolean* o comprovant que un hash correspon a unes dades concretes *CheckHash(byte[], byte[], String):boolean*.
- *Creació parell de claus asimètriques*: Creació d'un parell de claus asimètriques del tipus i longitud demanat per l'usuari, *Create_KeyPair(String, int): KeyPair*.
- *Creació de clau simètrica*: Creació de clau simètrica de l'algorisme desitjat i utilitzant les especificacions subministrades, *Create_SecretKey(String, KeySpec):SecretKey*.

A banda, en les operacions de xifrar i desxifrar tant amb clau simètrica com asimètrica, excepte quan es fa servir la clau de la plataforma, permetrem que se'ns faciliti un paràmetre addicional per a configurar l'operació. Aquest paràmetre és el *AlgorithmParameterSpec*, en el qual, per exemple, s'indica la *salt* i el número d'iteracions en un xifratge de tipus PBE.

Totes les funcions anteriors estan condicionades a que les opcions triades per l'usuari estiguin disponibles als proveïdors criptogràfics que hi hagi a la màquina on s'executi el contenidor. Si no estigués disponible algun paràmetre s'informaria a l'usuari amb l'excepció corresponent, tal com fa Java.

A la llista anterior no hem descrit les funcions que utilitzaran la clau privada de la Plataforma, ja que requereixen i es mereixen, una explicació molt més detallada.

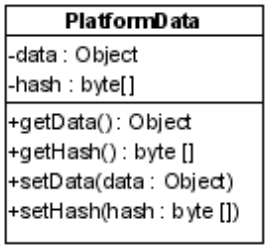

Figura 5.5: Classe PlatformData

La funció de desxifrar amb la clau privada de la plataforma (*Asymmetric_ Decrypt(byte [], CCHKey): Object*) serà la que a través del mecanisme que s'explica a [ARO04], ens permetrà evitar que un agent desxifri dades d'altres agents. Perquè això sigui possible, a la funció se li hauran de subministrar les dades xifrades i el CCHKey, que haurà d'haver estat prèviament demanat per l'agent utilitzant el mètode *generateCCHKey*. Donat que aquesta operació requerirà l'accés a la clau privada de la plataforma i als hashos emmagatzemats, aquesta operació haurà de ser realitzada en el *Main-Container*. Per aquest motiu quan un contenidor rebi una sol·licitud d'aquesta funcionalitat, l'haurà de transmetre al Main-Container.

Les dades a desxifrar hauran d'haver estat posades en un element de la classe PlatformData, conjuntament amb el hash del codi de control de l'agent, abans de ser xifrades. El PlatformData (figura 5.5) serà una classe que definirem per a poder implementar el mecanisme de l'article i contindrà un camp per a dades i un altre pel hash. Aquests camps corresponen al que en el segon capítol ens hem referit a la parella (D,A), on D és el camp de dades i A el camp del hash. El camp del hash òbviament ha de ser un byte[], perquè és el tipus de dades amb els quals es representen els hashs a Java, d'altra banda, per a les dades vam decidir posar el camp com a Objecte perquè fos el més genèric possible.

Al desxifrar les dades que ens passi l'agent, el servei haurà de comprovar que es tracta d'un element PlatformData, si és així el servei comprovarà que el hash es correspon amb el que prèviament ha calculat ell, que podrem recuperar gràcies al CCHKey que ens passarà l'agent. Només en el cas que les dues comprovacions

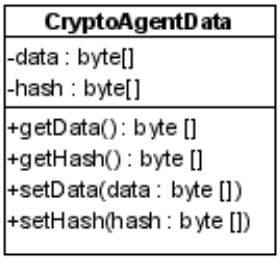

Figura 5.6: Classe CryptoAgentData

siguin correctes el servei tornarà les dades desxifrades a l'agent, en qualsevol altre cas les dades seran eliminades.

L'altra funció amb la clau privada de la plataforma és la signatura. L'objectiu d'aquesta signatura és que el propietari de l'agent pugui certificar que ha estat un agent en concret qui ha demanat a la plataforma que es signin les dades i la plataforma s'assegura que l'agent no es pugui desentendre de les dades, si vol demostrar la signatura en qüestió.

Com ja hem comentat a l'anàlisi per a identificar l'agent, igual que fèiem al desxifrar dades, utilitzarem el hash del codi de l'agent. Quan un agent demani que si li signin unes dades amb la clau privada de la plataforma (*Sign(byte[], CCHKey, String):SignedObject*), a part de les dades a signar, haurà de facilitar a la plataforma el seu CCHKey. A través d'aquest CCHKey la plataforma, accedint a la taula CCHKeyTable, podrà recuperar el hash de l'agent per a afegir-lo a les dades a signar.

Igual que en el cas del PlatformData haurem de definir una classe que consti d'un camp per a les dades i d'un camp per al hash. Aquesta classe serà el CryptoAgentData (figura 5.6). El CryptoAgentData estarà compost per 2 byte[] un per a les dades i un altre per al hash.

Un cop haguem posat les dades i el hash en un CryptoAgentData, aquest objecte serà posat i signat en un SignedObject, que serà el que se li retornarà a l'agent. A partir d'aquest SignedObject l'agent podrà recuperar les dades i

comprovar-ne la signatura fàcilment. A més a més, vam decidir que l'agent pugui triar l'algorisme del hash que s'efectua abans de realitzar la signatura. Aquest algorisme ha d'estar disponible a la plataforma, en cas contrari s'informarà a l'agent que l'operació no s'ha pogut realitzar.

5.1.4 Claus públiques

Perquè el nostre servei pugui ser utilitzat pels agents necessitem que aquests tinguin accés a les claus públiques de les plataformes. Sense aquest accés no es podrien xifrar els elements PlatformData abans de que l'agent arribi a la plataforma, ni es podrien comprovar les signatures que facin les plataformes. Per tant, per a poder provar tota la implementació del servei vam decidir implementar una solució temporal per a l'accés a les claus públiques. Per això vam implementar un servidor de claus públiques que s'executi a part del servei, perquè els diferents agents puguin accedir a les claus públiques de les plataformes.

Per a poder obtenir les claus públiques de les plataformes, els agents hauran de demanar les claus a la plataforma, les quals les demanaran al servidor de claus públiques. Això s'efectuarà sempre i quan no s'estigui demanant la clau pública de la plataforma en la qual es troba l'agent.

Com les claus privades, les claus públiques es guardaran en el *Main-Container*, ja què és on es crearà la parella de claus. Quan el servei arranqui en aquest contenidor carregarà a memòria les 2 claus. Posteriorment quan un agent li demani al servei la clau de la plataforma on es troba, el servei comprovarà si la petició s'ha produït en el *Main-Container*, en cas afirmatiu li retornarà la clau pública que tindrà en memòria. En canvi, si la petició s'ha originat en un *Agent-Container*, aquest demanarà al *Main-Container* la clau pública de la plataforma per a poder-la facilitar a l'agent.

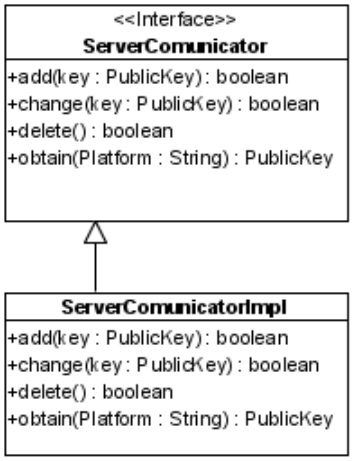

Figura 5.7: Classe ServerComunicator

5.1.5 Comunicació amb el servidor de claus públiques

El servidor de claus públiques s'executarà en una màquina, en la que no és necessari que hi hagi una plataforma executant-se. Vam decidir que les plataformes es comuniquessin amb el servidor de claus públiques via una connexió TCP amb aquest. A través d'aquest port s'enviaran les peticions d'operació al servidor de claus públiques i les respostes corresponents.

Perquè les plataformes puguin trobar el servidor de claus públiques, la informació del host on es troba i el port TCP pel qual està escoltant haurà de ser indicat en un fitxer dins del directori del servei.

El servei, quan s'iniciï per primer cop i creï un parell de claus per a la plataforma on s'està executant, farà una petició al servidor de claus públiques perquè aquest afegeixi la seva clau al registre de claus públiques que conté. La resta de vegades que s'executi el nostre servei, un cop hagi carregat la clau, procedirà a comprovar que la seva clau pública es troba emmagatzemada en el servidor de claus públiques i si no és així l'afegirà. Aquest pas s'efectuarà per si durant el temps que la plataforma no s'ha executat el servidor de claus públiques ha canviat

o per si ha hagut de reiniciar les claus que tenia emmagatzemades. D'altra banda, la comprovació de la clau també servirà perquè el servei pugui assegurar-se que la clau que està emmagatzemada al servidor de claus públiques és la correcte, i en cas contrari ho pugui notificar a l'administrador de la plataforma.

La comunicació entre el servei i el servidor de claus públiques es realitzarà a través de la classe ServerComunicator (Figura 5.7). Aquesta classe serà una interfície on definirem les funcions bàsiques que serviran per a comunicar-se amb el servidor de claus públiques. Aquesta interfície serà implementada per ServerComunicatorImpl (Figura 5.7). Aquesta separació en dues classes es deu a què aquesta implementació és una solució temporal.

5.2 Disseny del servei de JADE

Ara que ja hem explicat el funcionament del nostre servei, passarem a explicar com ho estructurarem específicament dins d'un servei de JADE. Per a fer això explicarem quines comandes definirà el nostre servei, on s'implementarà cada una de les funcionalitats, quins filtres crearem, a quines funcionalitats es podrà accedir a través del Helper i tot el que sigui necessari per a especificar com serà el nostre servei.

5.2.1 Helper

En el *Helper* oferirem tota la funcionalitat de la que disposa el nostre servei als agents. A través del nostre *Helper* (*CryptoHelper*) serà com els agents hauran de demanar el CCHKey i accediran a totes les funcions criptogràfiques del servei. Per tant en aquesta interfície definirem les funcions que faran servir la clau privada de la plataforma per a desxifrar i signar, seguint els mecanismes prèviament explicats, i tota la resta de funcions que podran utilitzar els agents: xifrar/desxifrar amb clau simètrica/asimètrica, signar dades, verificar signatures, calcular i comprovar hashos, demanar la generació de claus simètriques i asimètriques i obtenir

les claus públiques de la plataforma. A la figura 5.8 podem veure un exemple d'ús del nostre Helper en el qual l'agent demana que se li generi el CCHKey i posteriorment l'utilitza per a desxifrar unes dades amb la clau privada de la plataforma.

Totes les funcions definides en aquesta interfície seran implementades en una inner class que es trobarà en la classe *Service* del nostre servei (*CryptoService*). A la figura 5.9 podem veure les classes directament relacionades amb el nostre Helper.

5.2.2 Comunicació entre contenidors

Com hem vist quan hem anat explicant les diferents decisions de disseny que hem efectuat al llarg del projecte, hi ha diferents llocs en els quals el servei requereix que hi hagi una comunicació entre diferents contenidors. En tots els casos aquesta comunicació es realitzarà enviant comandes horitzontals des de qualsevol contenidor al *Main-Container*. Això es deu a que diferents parts del nostre servei es troben centralitzades en aquest contenidor.

Per a realitzar aquesta comunicació en el nostre *Slice* (*CryptoSlice*) definirem les comandes horitzontals, corresponents a aquestes comunicacions i les funcions que les enviaran al contenidor destí. D'altra banda en el *SliceProxy* (*CryptoSliceProxy*) implementarem aquestes funcions encarregades d'enviar cada una de les comandes horitzontals.

En el nostre servei definirem cinc comandes horitzontals diferents, una per cada una de les següents necessitats.

- Per a desxifrar unes dades amb la clau privada de la plataforma, ja que

```
CryptoHelper sh=(CryptoHelper)myAgent.getHelper("jade.core.crypto.CryptoService");
ch.generateCCHKey(myAgent.getAID());
Object data=ch.Asymmetric_Decrypt(encrypted_data,cchkey);
```

Figura 5.8: Exemple d'ús del CryptoHelper

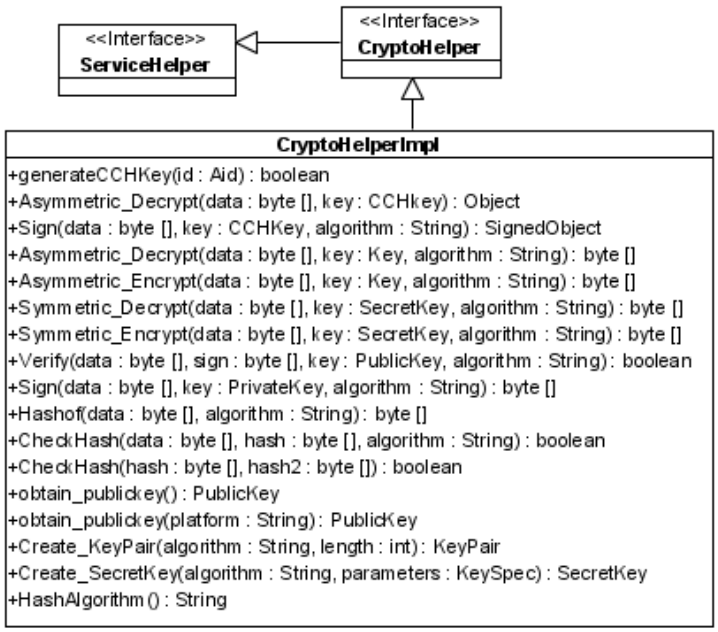

Figura 5.9: Diagrama de classes del CryptoHelper

aquesta només es troba en el *Main-Container*.

- Una altra comanda horitzontal per a quan s'hagin de signar les dades amb la clau privada de la plataforma.

- Quan se'ns demani la generació del CCHKey aquest es generarà en el *Main-Container* i es guardarà en una taula en aquest contenidor. Per això els contenidors quan se'ls hi demani un CCHKey passaran la petició al *Main-Container*, que els hi retornarà el CCHKey signat.

- Quan els agents morin a la plataforma o migrin a una altra hem d'eliminar el CCHKey de la CCHKeyTable que tenim en el *Main-Container*. Per això quan un d'aquests fets succeeixi enviarem una comanda horitzontal perquè s'elimini de la taula la entrada corresponent.

- Finalment, definirem una comanda per a demanar la clau pública de la plataforma al *Main-Container*.

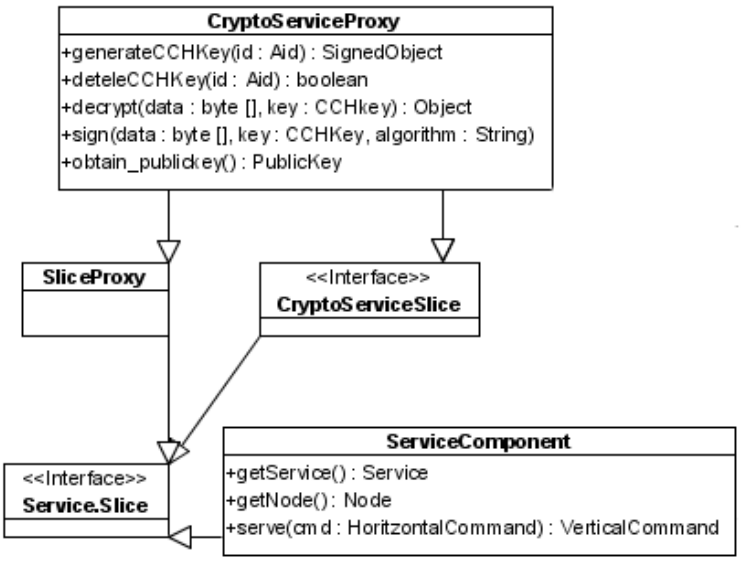

Figura 5.10: Diagrama de classes de la comunicació entre contenidors

Totes les comandes horitzontals del nostre servei quan arribin al contenidor destí les transformarem en comandes verticals perquè el *TargetSink* les processi. Aquesta transformació la farem a la classe *ServiceComponent*. A la figura 5.10 podem veure aquestes classes i la relació entre elles.

5.2.3 Sinks

En el nostre servei implementarem els 2 tipus de sinks que hi ha en el serveis a JADE, el *SourceSink* i el *TargetSink*. En el *TargetSink* hi implementarem les funcionalitats corresponents a les comandes horitzontals del anterior apartat. Per tant hi haurem d'implementar les funcions corresponents a desxifrar i signar amb la clau de la plataforma, implementant tots els mecanismes descrits anteriorment, crear un nou CCHKey, eliminar una entrada a la taula de CCHKeys i retornar la clau pública de la plataforma.

D'altra banda en el nostre servei també crearem un conjunt de comandes verticals perquè altres serveis existents en el contenidor puguin demanar-li fun-

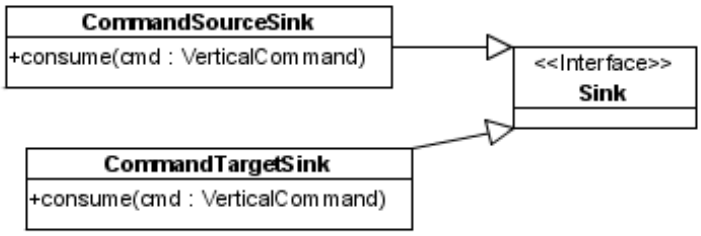

Figura 5.11: Diagrama de classes dels sinks

cionalitats criptogràfiques. En aquestes comandes posarem a disposició dels altres serveis totes les funcionalitats criptogràfiques del servei exceptuant les relacionades amb la clau de la plataforma.

Vam decidir que les funcions amb la clau de la plataforma no estiguessin disponibles per a altres servei perquè els mecanismes que implementen estan directament dirigits als agents i no es poden extrapolar als serveis. A més a més, si permetéssim la utilització lliure de la clau privada de la plataforma als altres serveis aquesta seria una via perquè els agents es saltessin els mecanismes corresponents.

En el *SourceSink* processarem totes aquestes comandes verticals i en retornarem el resultat al servei que hagi fet la petició. A la figura 5.11 podem veure el diagrama de classes dels sinks del nostre servei.

5.2.4 Filtres

En el nostre projecte només necessitarem la creació d'un filtre per a interceptar les comandes d'altres serveis, el *OutgoingFilter*(Figura 5.12). En aquest filtre interceptarem les comandes del *Agent Mobility Service* per a detectar quan un agent mor i una comanda del *InterPlatform Mobility Service* per a saber quan un agent migra cap a una altre plataforma. Interceptarem aquestes comandes per a eliminar el CCHKey de la taula corresponent.

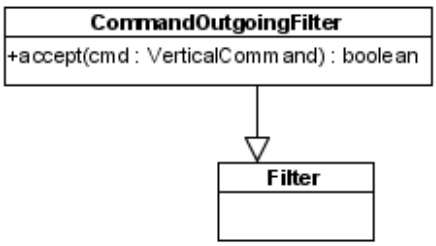

Figura 5.12: Diagrama de classes del *OutgoingFilter*

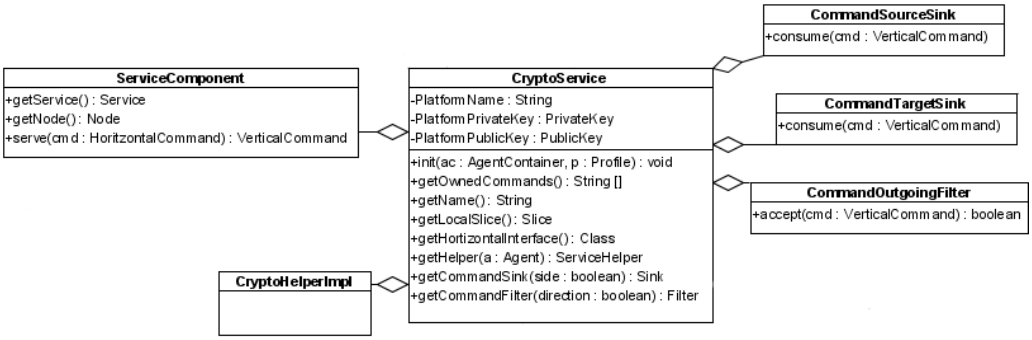

Figura 5.13: Classe CryptoService

Donat que el nostre servei no substitueix cap d'aquestes comandes i, que no volem interferir en la migració dels agents, un cop haguem actuat en conseqüència a la comanda interceptada deixarem que continuï el seu recorregut per la cua de filtres.

A part de definir aquestes classes del servei, en la classe *CryptoService* hem de definir tots els mètodes necessaris que hem comentat en el capítol 4, estudi de l'arquitectura dels serveis a Jade. A la figura 5.13 veurem tots aquests mètodes juntament amb classes de les que hem parlat en aquest capítol que estan definides com a *inner-classes* de la classe *CryptoService*, de la classe *CryptoHelperImpl* ometrem els mètodes ja que són molt nombrosos.

Finalment a la figura 5.14 podem veure com estan relacionades les classes principals del nostre servei criptogràfic.

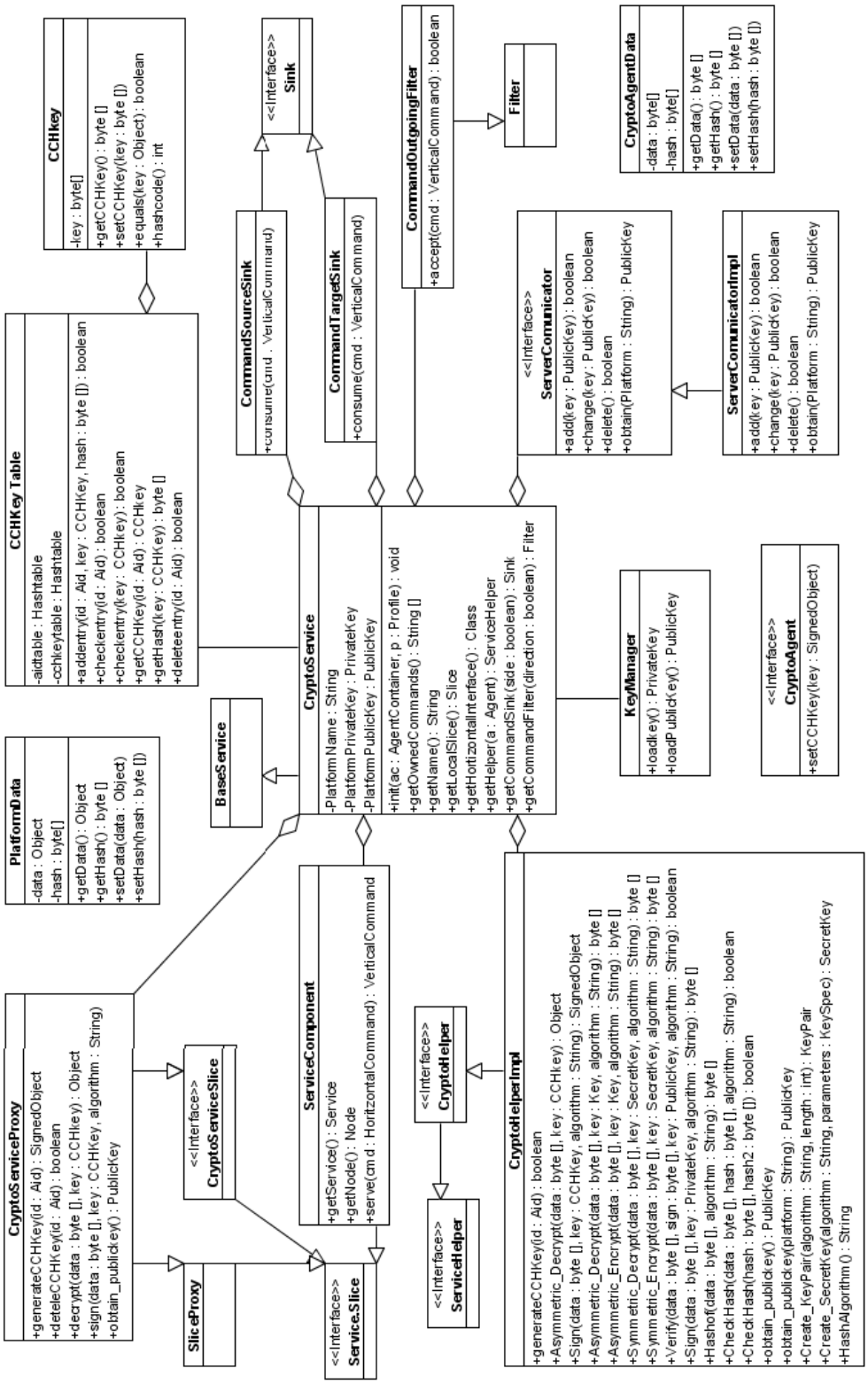

Figura 5.14: Diagrama de classes del servei

5.3 Servidor de claus públiques

5.3.1 Funcionament del servidor de claus públiques

El servidor de claus públiques emmagatzemarà les claus acompanyades del nom de la plataforma a la qual correspon aquella clau pública. Per tal de poder obtenir la clau a partir del nom de la plataforma. Vam decidir que el servidor de claus públiques atendria 4 tipus d'operacions: afegir, obtenir, canviar i eliminar la clau pública i que les resoldria tal i com expliquem a continuació.

Afegir una clau pública

Quan el servidor de claus públiques rebi una petició d'afegir una clau, aquest comprovarà que no existeix cap entrada corresponent a aquell nom de plataforma. Si aquesta entrada no existeix crearà una entrada per a la nova clau pública i li notificarà a la plataforma que l'operació s'ha realitzat correctament. En cas contrari, si ja hi ha una clau corresponent a aquella plataforma se n'informarà a la plataforma que ha fet la petició.

Obtenir la clau pública

Les peticions d'obtenció de clau tenen un mecanisme molt senzill. Quan el servidor de claus públiques rebi aquest tipus de peticions comprovarà si el nom de la plataforma es troba a la seva taula de claus, si és així, retornarà la clau corresponent, en cas contrari notificarà que no es troba la clau demanada al servidor de claus públiques.

Canviar i eliminar una clau pública

En el cas de canviar i eliminar les claus públiques ens hem d'assegurar que qui ens demana realitzar l'operació és el propietari de la clau i no pas algú intentant atacar a la plataforma propietària. Aquest atac provocaria que les dades no es poguessin

desxifrar a la plataforma i que les comprovacions de signatura fallessin. Per a fer aquesta comprovació les peticions d'aquest tipus hauran d'anar acompanyades d'unes dades concretes signades amb la clau privada de la plataforma. Posteriorment, el servidor de claus públiques comprovarà la signatura amb la clau pública que hi ha al servidor de claus públiques i si aquesta correspon sabrem que qui intenta modificar la clau pública és el seu propietari.

En el cas de canviar la clau pública inicialment el servidor de claus públiques comprovarà que la clau existeix en el servidor de claus públiques, a continuació comprovarà la signatura comentada anteriorment i si aquesta es verifica correctament canviarà la clau del servidor de claus públiques. En cas que la verificació falli o la clau no es trobi al servidor de claus públiques se li notificarà adequadament a la plataforma que hagi fet la petició. L'eliminació de clau seguirà el mateix protocol, però enlloc de substituir la clau, l'última operació seria l'eliminació de l'entrada de la taula.

Emmagatzematge

Quan s'apaga el servidor de claus públiques i perquè en futures execucions segueixi tenint les claus que ja li havien afegit, vam decidir que aquestes es guardin en un fitxer (*savetable(HashTable, char[]):void*). Per a protegir-les i que ningú les manipuli, aquest fitxer es protegirà amb un sistema PBE com fem amb les claus privades. Per tant quan s'arranqui el servidor de claus públiques s'haurà d'introduir una contrasenya per a desxifrar l'arxiu i poder carregar les claus, *loadtable(char[]):HashTable*.

Per acabar amb el servidor de claus públiques, el posseïdor de la contrasenya també tindrà la possibilitat d'eliminar entrades manualment. Aquesta opció s'ha contemplat per, si alguna plataforma perd les seves claus i n'ha de crear unes altres, tenir un sistema per a substituir-les en el servidor de claus públiques.

5.3.2 Disseny del servidor de claus públiques

Un cop hem vist el funcionament del Servidor de claus públiques en mostrarem el disseny. El servidor de claus públiques consta principalment de 4 classes.

La classe Server és la classe principal. En aquesta classe situarem tota la gestió de les dades del servidor de claus públiques a l'arrancar-lo i apagar-lo i també tota la interacció amb l'usuari via teclat. Quan arranquem el servidor de claus públiques en aquesta classe preguntarem la contrasenya a l'usuari i carregarem la taula a memòria, si existeix. Durant la execució, en aquesta classe, esperarem les comandes de l'usuari per teclat, perquè pugui eliminar una entrada manualment o per a guardar la taula quan vulgui apagar el servidor de claus públiques.

Perquè el servidor de claus públiques pugui respondre les peticions de les plataformes un cop haguem carregat la taula de claus, obrirem el socket i llençarem un *thread* que s'encarregui d'escoltar pel socket obert. El thread en qüestió serà el *ConsultThread*. Aquest *thread* s'encarregarà d'esperar les peticions de connexió que arribin pel *socket* i llençar un altre *thread* per resoldre cada una d'elles. D'aquesta manera el servidor de claus públiques no es quedarà bloquejat mentre resol una petició.

Aquest últim *thread* és el *AnswerThread*, aquest thread obtindrà la petició de la connexió corresponent, la processarà i respondrà a la plataforma que l'hagi originada. Aquest *thread* serà el que tindrà contacte directe amb la taula de claus.

Per acabar, el servidor de claus públiques tindrà 1 classe més, la CryptoFunctions. En aquesta classe es on agrupem les funcions criptogràfiques per a utilitzar-les en la resta de classes. El servidor de claus públiques utilitzarà la CryptoFunctions per a verificar les signatures que vinguin amb les peticions de canvi o eliminació de clau pública del servidor de claus públiques. Finalment veurem a la figura 5.15 el diagrama de classes que mostra la relació d'aquestes 4 classes, en aquest diagrama de la classe CryptoFunctions només mostrarem el mètode necessari.

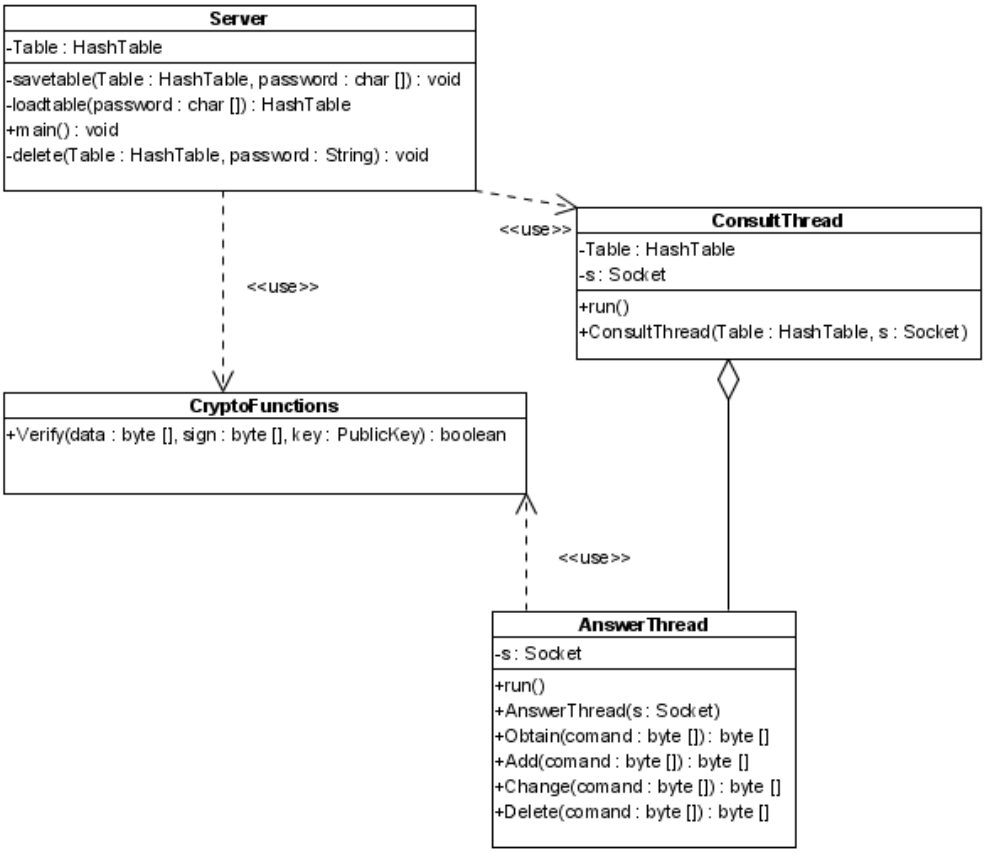

Figura 5.15: Diagrama de classes del servidor de claus públiques

Capítol 6

Implementació i proves

6.1 Implementació

En aquest apartat explicarem aspectes del projecte que no han quedat del tot definits en capítols anteriors o aspectes que vam haver de canviar respecte el disseny inicial.

6.1.1 Servei

Protecció de la clau

La protecció de la clau privada de la plataforma recau en una contrasenya que s'ha d'introduir cada cop que es posi en marxa el servei. El que fins ara no hem comentat és que quan escrivim quelcom en un terminal el text escrit queda allà imprès. Lògicament, no podíem deixar que la contrasenya quedés allà visible a l'abast de qualsevol que pogués mirar el terminal, per això vam haver de buscar un sistema de protecció. Després de realitzar una recerca vam localitzar a la web de Sun ([Sun07]) un thread de Java que substitueix els caràcters escrits per asteriscos. D'aquesta manera vam aconseguir que la contrasenya no quedés escrita en el terminal. Un exemple del funcionament d'aquest codi es veu a la figura 6.1.

Canvi de clau

Cada cop que s'arranqui el servei, donarem la possibilitat a l'usuari de renovar el parell de claus asimètriques de la plataforma i de canviar la contrasenya que les protegeix. Així, quan s'arranqui el servei demanarem a l'usuari que introdueixi la contrasenya. Un cop comprovat que la contrasenya es correcta procedirem a preguntar a l'usuari si vol canviar el parell de claus asimètriques de la plataforma. Si l'usuari decideix canviar la clau, es generarà el nou parell de claus asimètriques i es demanarà al servidor de claus públiques que canviï la clau pública que té emmagatzemada.

Després de preguntar si es vol canviar la clau es preguntarà a l'usuari si vol modificar la contrasenya. Si així ho desitja se li demanarà que introdueixi la nova contrasenya. Aquesta possibilitat l'hem realitzada perquè una bona protecció de les dades recau en modificar la contrasenya periòdicament. Totes aquestes contrasenyes seran ocultades pel mètode comentat anteriorment.

A la figura 6.1 veiem un exemple en el qual es modifica la clau de la plataforma, però no la contrasenya que la protegeix.

La clau a disc

En el capítol de disseny hem dit que les claus de la plataforma es guardarien a disc per a poder-les carregar cada cop que s'arranqui la plataforma. El que fins ara no hem comentat és on les guardarem. Al fer la implementació del servei hem pensat que el millor lloc on guardar les claus és en 2 fitxers, un per cada clau, dins de la carpeta del nostre servei. D'aquesta manera tots els fitxers relacionats amb el servei es troben junts en el mateix lloc del disc.

Els contenidors de JADE no tenen perquè ser únics en una màquina. Es pot donar el cas que en una màquina s'executin diferents contenidors, de diferents plataformes. Aquest fet obre la porta a que en una màquina s'executi més d'un *Main-Container*. Per defecte a JADE les plataformes tenen com a identificador el nom de la màquina seguit pel port RMI que utilitza la plataforma per les co-

```
Please, enter the Administrator CryptoService Password
*******
Do you want to change the Platform KeyPair?(yes/no)
yes
Do you want to change the Administrator CryptoService Password?(yes/no)
no
Generating a new random KeyPair for the Platform... Done
Changing the PublicKey on the Server...Done
```

Figura 6.1: Canvi de clau

municacions entre els diferents contenidors. Per aquest motiu per a identificar les claus de les plataformes quan les guardem a disc utilitzarem el nom complet de la plataforma.

Localització del servei

Des de dins de la plataforma no tenim forma de saber en quin directori es troben els arxius de la plataforma JADE. Com hem comentat fins ara hi ha un conjunt d'arxius que es guarden dins del directori del nostre servei, les claus asimètriques de la plataforma i el fitxer amb la informació del servidor de claus públiques. Aquest directori es troba en un subdirectori de JADE. Per a poder localitzar aquests arxius des de dins del servei hem d'obligar a l'usuari de JADE a engegar la plataforma en unes condicions especials.

D'una banda la plataforma pot ser engegada des de dins del directori de JADE. D'aquesta manera podrem arribar fins el directori on es troben els fitxers del servei. D'altra banda donem una segona opció a l'usuari si vol engegar la plataforma des de qualsevol ubicació. Perquè l'usuari pugui fer servir aquesta opció abans de cridar a la plataforma haurà de crear una variable d'entorn *JADE_HOME*, indicant a quin directori es troba JADE. Aleshores el servei podrà utilitzar aquesta variable per localitzar el directori del servei i poder accedir als arxius necessaris.

6.1.2 Servidor de claus públiques

Un altre aspecte del que no hem parlat en el disseny és com guardar la clau pública en el servidor de claus públiques. Per tal de fer el servidor de claus públiques el més estàndard possible, tot i que sigui una solució temporal, es va decidir emmagatzemar les claus en el format estàndard *Encoded* per tal de deslliurar-lo de l'objecte que representa les claus públiques a Java. D'aquesta manera la utilització de les claus del servidor de claus públiques no queda lligat a l'ús de Java per ferles servir. Per a poder reconstruir la clau a partir del format *Encoded* necessitarem guardar l'algorisme de la clau.

Tal com passa a l'iniciar el servei, a l'iniciar el servidor de claus públiques també es demana una contrasenya per a desxifrar la taula que hi ha a disc. Per a ocultar aquesta contrasenya quan s'escriu en el terminal hem fet servir el mateix mecanisme que en el servei.

Igual que quan guardem les claus a disc per a identificar les plataformes en el servidor de claus públiques farem servir el nom complet que també inclou el port.

6.1.3 Comunicació

Per acabar amb els aspectes de la implementació falta descriure la comunicació entre el servei i el servidor de claus públiques. Aquesta comunicació es farà a través d'un byte[] que tindrà les diferents dades a enviar. La no utilització d'una classe serialitzada, tot i que facilitaria la construcció dels missatges, es decideix per a permetre que aquest servidor de claus públiques es pugui comunicar amb programes que no són Java. El missatge estarà compost d'un primer camp que indicarà l'operació a realitzar, una segona part, que contindrà camps, limitats per un separador, que indiquen la longitud de cada un dels paràmetres i una tercera part amb els paràmetres.

Els paràmetres de les crides podran ser: la clau pública, l'algorisme de la clau pública, el nom de la plataforma, la signatura i l'algorisme de la signatura.

OP CODE	S	longitud camp 1	S	longitud camp 2	...	longitud camp N	S	camp 1	camp 2	...	camp N
Add	S	longitud alias	S	longitud clau	S	longitud algorisme clau	S	alias	clau		algorisme clau

Figura 6.2: Comandes de comunicació amb els servidor de claus públiques

No totes les comandes tindran el mateix nombre de paràmetres, cada comanda només portarà els paràmetres necessaris per a realitzar l'operació corresponent. A la figura 6.2 podem veure l'estructura genèrica de les comandes i l'estructura en concret de la comanda d'afegir una clau al servidor de claus públiques (la S significa separador).

6.2 Proves

Des que vam iniciar la implementació s'han anat realitzant proves de les parts que anàvem finalitzant per tal de comprovar que funcionaven correctament. Un cop acabada la implementació vam centrar-nos a realitzar proves generals per a provar el correcte funcionament de tot el projecte. Aquestes seran les proves que explicarem en aquest capítol.

En aquest capítol veurem els resultats de les proves de càrrega i gestió de la clau privada de la plataforma, funcions criptogràfiques, gestió de CCHKey i obtenció de les claus del servidor de claus públiques.

Anteriorment a la implementació, a la figura 6.1, ja hem vist un exemple de la inicialització en la qual es modifica la clau privada de la plataforma. Sobre aquest punt també vam realitzar proves de modificació de contrasenya i intentar posar en marxa el servei amb una contrasenya incorrecte amb resultats satisfactoris. El resultat d'aquesta última prova es pot veure a la figura 6.3.

Una altra prova que vam realitzar va ser, un cop ja teníem la clau de la plataforma creada, modificar el servidor de claus públiques que fa servir la plataforma. En aquesta prova vam apreciar com la plataforma afegeix la clau al servidor de claus públiques (Figura 6.4) i com la clau es posteriorment accessible des d'una

```
Please, enter the Administrator CryptoService Password
**************
Problem while reading the PrivateKey Given final block not properly padded
The AdminPassword authentification has failed, the CryptoService is going to Run without PlatformPrivateKey
```

Figura 6.3: Contrasenya errònia

```
Please, enter the Administrator CryptoService Password
********
Do you want to change the Platform KeyPair?(yes/no)
no
Do you want to change the Administrator CryptoService Password?(yes/no)
no
Adding the Publickey to the Server... Done
```

Figura 6.4: Afegint la clau pública al servidor de claus públiques

altra plataforma (Figura 6.5, els missatges d'aquesta figura són generats per un agent).

Posteriorment vam realitzar les proves per a comprovar que les funcions criptogràfiques de la plataforma funcionaven correctament. A la figura 6.6 podem veure la generació de diferents tipus de claus i la utilització d'aquestes per a xifrar i desxifrar dades (els missatges d'aquesta figura són generats per un agent). En aquesta figura també podem veure una prova en la qual vam efectuar una signatura i després vam intentar verificar la signatura amb les dades correctes i amb dades incorrectes. Com es pot apreciar a la figura només en el cas en que coincideixin les dades, la verificació és correcta.

D'altra banda, vam realitzar proves de les funcions de hash utilitzant diferents combinacions de dades i algorismes. Totes les proves realitzades van donar els

```
Obtaining the Platform PublicKey
Sun RSA public key, 2048 bits
  modulus: 164873221435500749118592167238660778582166904275947805117679154723597146109616128453565342029010883246250049404326
179134527701557669121969685683717563055079791455320374664962917615483864743981990584875258261344224476470863733220433431921562221085639247654921222792540886313338370001869287686159577137628866589910226490911910312361797449622536079893245183724619321647145642400331696341141906557824414618197761252628051469251426468967060071000515994269508119502746044042839873436660347588793882133694107251502498670022952806464551297378057825311550039413427676041438149890724810981741547472184373405914772509214751277 09
  public exponent: 65537
```

Figura 6.5: Obtenció de clau pública

```
Creating a DSA KeyPair...
Signing Data...
Verifying Data: true
Verifying Antoher Data :false

Creating a RSA KeyPair...
Encrypted Data: Jennifer Morrison
Decrypted Data: Jennifer Morrison

Creating a SecretKey...
Encrypted Data: Adison Cameron
Decrypted Data: Adison Cameron
```

Figura 6.6: Funcions criptogràfiques

```
Generating some Hashes (hash1==hash2!=hash3)
Verify hash1 with data of hash1: true
Verify hash3 with data of hash1: false
Verify hash1 with hash2: true
Verify hash1 with hash3: false
Verify hash of same data with diferent algorithm: false
```

Figura 6.7: Funcions de hash

resultats esperats, una mostra d'aquestes proves les trobem a la figura 6.7 (els missatges d'aquesta figura són generats per un agent).

Finalment, a la figura 6.8 podem veure els diferents usos del CCHKey a la nostra plataforma (els missatges d'aquesta figura són generats per un agent). En el punt 2 de la figura podem veure el resultat de desxifrar les dades amb la clau de la plataforma. D'altra banda, en el punt 3 podem veure el resultat d'intentar desxifrar les dades d'un altre agent. I finalment en el punt 4 veiem com al demanar la signatura d'unes dades amb la clau de la plataforma, el servei ens inclou les dades i el Hash de l'agent (punt 1) dins de la signatura.

També vam fer passar un agent, diferents cops, per la mateixa plataforma sense parar aquesta, per a poder comprovar que el CCHKey es calcula cada cop de nou (Figura 6.9, els missatges d'aquesta figura són generats per un agent). També vam comprovar que quan un agent mor a la plataforma també es borra el seu CCHKey i hash de la taula de la plataforma.

Per acabar amb les proves del CCHKey vam provar les situacions en les quals

```
Agent Hash: (1)
84,-38,47,-8,62,43,114,103,-126,-84,-91,-56,72,-87,-116,-81,101,65,-44,-26,54,14,-30,36,-103,100,91,80,-14,-44,22,14,63,115,-
48,107,116,-25,29,65,-13,-52,114,108,97,-20,-89,92,42,-114,78,-54,-107,-35,-20,115,7,-9,113,39,109,60,30,-68,

Plataform Decrypt Test (2)
Encrypted Data: Kate Austin
Decrypted Data: Kate Austin

Plataform Decrypt with wrong hash Test (3)
Encrypted Data: Lilly Evangeline
Decrypted Data: null

Platform Signature Test (4)
Data to Sign: Lilly Evangeline
Data inside the Signature: Lilly Evangeline
Hash inside the Signature
84,-38,47,-8,62,43,114,103,-126,-84,-91,-56,72,-87,-116,-81,101,65,-44,-26,54,14,-30,36,-103,100,91,80,-14,-44,22,14,63,115,-
48,107,116,-25,29,65,-13,-52,114,108,97,-20,-89,92,42,-114,78,-54,-107,-35,-20,115,7,-9,113,39,109,60,30,-68,
Signature Checking: true
```

Figura 6.8: Utilització CCHKey

```
Asking for the CCHKey:
122,-27,-124,-100,123,-108,-82,-122,-42,-92,-109,33,53,-14,97,108,
Going to Container-1

Asking for the CCHKey:
0,17,35,-118,-111,-18,-77,67,26,66,-45,-51,118,74,90,-124,
Going to Container-1
```

Figura 6.9: Eliminació CCHKey

l'agent no reuneix les condicions necessàries per a demanar el CCHKey. Aquestes situacions són quan l'agent no implementa la classe CryptoAgent, quan no ens informa degudament de quina és la seva *Main-Class* o quan la classe indicada no es troba entre les classes de l'agent. En totes aquestes proves vam obtenir els resultats esperats. A la figura 6.10 podem veure com reacciona el servei, quan l'agent li sol·licita el CCHKey, i el servei no pot obtenir la *Main-Class* de l'agent.

En resum, les proves realitzades a la versió final del servei han estat completament satisfactòries.

```
The Main-Class has not been found
The CCHKey generation has aborted
```

Figura 6.10: Main-Class no indicada o errònia

Capítol 7

Conclusions

En aquest projecte s'ha realitzat el desenvolupament d'un servei criptogràfic per a la plataforma JADE. En aquest servei, d'una banda, s'han implementat els mecanismes presentats a [ARO04], necessaris perquè es puguin implementar agents autoprotegits en les plataformes JADE i, d'una altra, s'han implementat funcions criptogràfiques de fàcil utilització per part de l'agent.

S'han implementat mecanismes a través dels quals es pretén donar solució a una part dels problemes de seguretat que ens trobem en l'entorn dels agents mòbils. Aquests mecanismes tenen per objectiu protegir els agents d'atacs en els quals l'objectiu és la modificació del codi o de les dades, per alterar-ne el funcionament o el resultat respectivament.

Un d'aquests mecanismes garanteix que les dades que portin els agents, degudament emmagatzemades, no podran ser desxifrades per un atacant que les robi amb la intenció de descobrir-ne el contingut. A part, també s'ha implementat un sistema a través del qual es pot demostrar l'autoria d'unes dades per part d'un agent, podent comprovar que unes dades provenen d'un agent en concret.

Amb el nostre servei hem dotat les plataformes d'un parell de claus asimètriques RSA de 2048 bits de longitud. Mentre el servei està en funcionament, la clau privada de la plataforma no és directament accessible pels agents, sinó que els agents

només poden demanar al servei que faci operacions amb aquesta clau. Per protegir la clau quan guardem aquesta a disc la xifrem amb un PBE, Password-Based Encryption.

Per no tenir problemes de coherència en el parell de claus, a cada plataforma només hi haurà una còpia de la seva clau privada que es trobarà en el *Main-Container*. La resta de contenidors hauran d'acudir al *Main-Container*, perquè els faci les operacions que requereixin aquesta clau.

Els agents podran demanar al nostre servei que s'utilitzi la clau privada de la plataforma per a desxifrar i signar dades. Al desxifrar unes dades el servei implementa el mecanisme explicat a l'article [ARO04] en el qual s'ha de realitzar una verificació que l'agent que intenta desxifrar les dades és el propietari.

D'altra banda, al signar unes dades amb la clau privada de la plataforma aquestes s'acompanyaran d'un identificador de l'agent, perquè l'usuari pugui comprovar que les dades venen d'un agent en concret. Per a implementar els 2 mecanismes explicats al desxifrar i signar les dades hem definit els tipus de dades necessaris.

Com s'indica a l'article [ARO04] hem utilitzat el hash del codi de l'agent com a identificador de l'agent en els mecanismes anteriors.

A part de les funcions que utilitzen la clau privada de la plataforma, en el nostre servei oferim a l'agent funcions per a xifrar/desxifrar amb clau simètrica/asimètrica, signar dades, verificar signatures, calcular i comprovar hashos i demanar la generació de claus simètriques i asimètriques.

Finalment, per a poder provar tota aquesta funcionalitat hem implementat un servidor de claus públiques. Al no ser una part vital del projecte l'hem realitzat com una solució temporal, que en el futur hauria de ser tractada com es mereix. Les plataformes al generar el seu parell de claus asimètriques afegiran la seva clau pública al servidor de claus públiques. Posteriorment quan un agent demani a la plataforma una clau pública, aquesta accedirà al servidor de claus públiques per a obtenir-la.

Al finalitzar aquest projecte considerem que hem assolit tots els objectius que ens havíem marcat a l'inici del mateix.

7.1 Línies de millora

Després d'aquests mesos de treball hem vist quins són els punts forts i el punts febles del projecte i per això, a continuació veurem les línies de millora obertes que hi queden.

La principal cosa a millorar en aquest projecte és el servidor de claus públiques. Aquest servidor s'ha realitzat com una solució temporal ja que no era un requisit del projecte i vam preferir centrar el nostre temps en aconseguir una solució més acurada en els seus requisits. Per això vam realitzar un servidor de claus públiques que cobrís les nostres necessitats, però sense pretendre que fos la solució definitiva.

De la mateixa manera, la comunicació entre el nostre servei i el servidor de claus públiques que hem realitzat també és un punt en el qual es pot treballar en el futur per a aconseguir una solució més acurada i elegant.

Finalment, una altra millora a implementar és la protecció de les comunicacions del nostre servei que circulen entre els contenidors de la plataforma. Amb aquesta protecció evitaríem possibles atacs que volguessin trencar l'esquema de seguretat a partir de la informació extreta d'aquestes comunicacions.

Bibliografia

[ARO04] J. Ametller, S. Robles, i J. Ortega-Ruiz, *Self-protected mobile agents*, in 3rd International Conference on Autonomous Agents and Multiagent Systems. ACM Press, 2004.

[Cu06] Jordi Cucurull Juan, *Contribució a la mobilitat i seguretat dels agents*, Maig 2006. Disponible a: <https://tao.uab.cat/ipmp/>

[ECK] Bruce Eckel, *Thinking in Java Third Edition*. Disponible a: <http://www.mindview.net/Books/TIJ/>

[eclipse] The Eclipse Foundation; *Eclipse SDK*. <http://www.eclipse.org/>

[gantt] http://Sourceforge.net; *Gantt project*. <http://ganttproject.biz/>

[GC04] Giovanni Caire, *JADE: the new kernel and last developments*, 2004. Disponible a: <http://jade.tilab.com/papers/Jade-the-services-architecture.pdf>

[JADE] Java Agent DEvelopment Framework. 19 de Febrer del 2007 <http://jade.tilab.com/>

[Sun07] Sun Microsystems, Inc. <java.sun.com>

[Sun07-2] Sun Microsystems, Inc. *JavaTM 2 Platform Standard Edition 5.0 API Specification*. 19 de Febrer del 2007 <http://java.sun.com/j2se/1.5.0/docs/api/>

[uml05] UML Unified Modeling Language, juny 2005.
<http://www.uml.org/>

Firmat: Jaume Bigas Vence

Bellaterra, Juny de 2007

Resum

El present projecte de fi de carrera té com a objectiu principal el desenvolupament d'un servei criptogràfic per a la plataforma JADE, perquè es puguin implementar agents mòbils autoprotegits. Aquest objectiu s'ha aconseguit dotant les plataformes amb un parell de claus asimètriques i facilitant a l'agent funcions que utilitzen la clau privada de la plataforma entre un gran ventall de funcions criptogràfiques diferents.

Resumen

El presente proyecto de fin de carrera tiene como objetivo principal el desarrollo de un servicio criptográfico para la plataforma JADE, para que se puedan implementar agentes móviles autoprotegidos. Este objetivo se ha logrado dotando las plataformas con un par de claves asimétricas y facilitando al agente funciones que utilizan la clave privada de la plataforma entre un gran abanico de funciones criptográficas diferentes.

Abstract

This project has as main goal the development of a cryptography service for JADE platform, allowing the implementation of self-protected mobile agents. This goal has been accomplished equipping the platforms with an asymmetric key pair and facilitating to the agent functions that use the private key of the platform between a great set of different cryptographic functions.