MEMORIA

Índice general

Índice de figuras

Capítulo 1

Memoria

Contents

1.1. Antecedentes y contexto

La necesidad de proveer servicios telemáticos en una red que va creciendo en el tiempo, pone al administrador ante una disyuntiva: incorporar más personal para hacerse cargo del incremento en la carga de trabajo, o utilizar tecnologías que hagan más eficiente su labor. Podemos ilustrar la idea con un ejemplo:

"Una pequeña organización requiere de un servicio de correo electrónico. El servicio se ofrece inicialmente utilizando un servidor SMTP/IMAP en una máquina y, entre su configuración, podría destacar un fichero que almacenase pares usuario:contraseña para la autenticación de los usuarios. El servicio es un éxito pues la comunicación a distancia mejora la eficiencia del trabajo, y es por ello que se despliegan nuevos servicios y que el aumento de rendimiento hace tener éxito a la organización, que poco a poco va creciendo en tamaño y a la que se incorporan nuevos departamentos. Ahora existen servicios de correo, mensajería instantánea y telefonía pero, desafortunadamente, cada uno tiene su propio servidor (a veces uno por cada departamento) y cada servidor sigue manteniendo su propio fichero para las autenticaciones. Ésto es un problema, porque cada vez que se quiere dar de alta a algún nuevo usuario a medida que crece la organización, hay que hacerlo en una máquina distinta y por repetido en cada fichero.

Efectivamente, los administradores se dan cuenta de que los servicios ponen en juego dos tipos de información: los datos del servicio en sí (correos en el servicio de correo, conversaciones en el servicio de telefonía...) y los metadatos que necesitan los servicios para funcionar (por ejemplo si el servicio quiere ofrecer subservicios de seguridad, necesitaba usuarios y contraseñas). Del primer grupo, los datos, se "ocupan" los usuarios (pues son ellos los que generan los correos o las conversaciones), sin embargo del segundo grupo, los metadatos se ocupan los administradores. Dadas las previsiones de crecimiento, en un principio se pretende aumentar el número de administradores en plantilla, pero puesto que ya son conscientes de que tienen un problema en la gestión de sus metadatos, investigan sobre el asunto y a tiempo descubren unas tecnologías, los DBMS (sistemas gestores de bases de datos), que les permiten centralizar cualquier metainformación de los servicios (de autenticación u otra). Con ello, el administrador indica a los servidores que los metadatos están en la base de datos y, a partir de ese momento, se puede olvidar de ellos y centrarse en la base de datos: cada vez que llegue un usuario nuevo, crea la cuenta en el DBMS.

Los administradores se felicitan satisfechos, sin embargo durante ese tiempo los usuarios también han ido desarrollado sus propias quejas: la existencia de varios servicios hace que cada vez que necesiten usar alguno (y en este momento ya es algo muy frecuente) se encuentran ante

un proceso de log-in. Pues bien, el tener que teclear constantemente sus credenciales, aunque sea siempre lo mismo, les resulta tedioso. Durante un tiempo se desconsideró el problema, tomándolo como un mal necesario, pero ésto no duró mucho puesto que el problema afectaba más a quienes más usaban los servicios: los propios administradores. Es por ello que que éstos, buscando una solución, descubren unas tecnologías englobadas bajo el nombre de SSO (Single Sing On): es posible conseguir que la autentificación se realice una sola vez si se implantan ciertas técnicas en servidores y clientes. A ello se dedican en las próximas semanas y el problema queda resuelto.

Todos felices, el crecimiento de la organización sigue y el acuerdo es unánime: la práctica totalidad de las actividades de la organización encontrarán ventajas en utilizar la red y los servicios telemáticos. Pero la totalidad de actividades de la organización ya no son sólo las comunicaciones internas, sino también las que se producen al colaborar con otras organizaciones o al atender a individuos interesados en las actividades de la organización. Éso conlleva en el primer caso que debamos de alguna forma federar los sistemas de esas otras organizaciones con los nuestros, y en el segundo caso que personas no técnicas (y con equipos y posibilidad de instalar software variada...) tenga que usar una cierta variedad de servicios y aplicaciones con lo que, para hacerlo fácil, hay incluso quien opina que podría escojerse el navegador web como interfaz y puerta de entrada a todos ellos..."

Aunque el pequeño relato anterior es imaginario, ciertamente parece basado en hechos recientes. Efectivamente la irrupción de los servicios telemáticos es una realidad que empezó en los 90 con la popularización de Internet, y sigue creciendo[1][2]. Las necesidades que imponen ese crecimiento están haciendo que tecnologías desarrolladas en las últimas dos décadas y que se consideraban pertinentes sólo en grandes entidades empresariales o académicas, sean ahora susceptibles de constituir una solución a los problemas de escalamiento de un abanico más amplio de organizaciones[3][4]. Además, una buena parte de esas tecnologías han evolucionado en forma de Software Libre, luego están disponibles en general y la barrera para su implantación es más el "Know-how", el conocimiento. Este proyecto intenta aportar parte de ese conocimiento.

1.2. Alcance del trabajo

Vamos a centrarnos en una solución muy concreta que ha demostrado ser especialmente robusta en ciertos entornos. En lo que sigue vamos a describir estos entornos, en el siguiente apartado "Desarrollo" nos centraremos en la solución que hemos planteado y que desplegamos.

Este proyecto es pertinente para:

- Una organización que dispone de un parque informático conectado a través de

IPv4/TCP.

- Donde se pretende dar un servicio de Shell para uso completo de las máquinas.
- También se pretenden ofrecer servicios típicos persona-persona: correo, mensajería instantánea y presencia, telefonía.
- Buena parte de las necesidades de almacenamiento consisten en crear y compartir documentos (en una red LAN o WAN).[1]
- Escalabilidad: necesidad de ofrecer esos servicios a miles de usuarios sin que la calidad del servicio se afecte, incluso con previsiones de crecimiento.
- Fiabilidad y seguridad: unido al requisito anterior, necesidades de una alta probabilidad de funcionamiento correcto[2] y protección ante posibles amenazas.
- Multiplataforma: compatibilidad con las máquinas, SO y software más utilizados, tal como SO POSIX(GNU/Linux, MAC OSX, ...) o MS Windows sobre i386+.
- Bajo coste.

Anotación: nótese que no hacemos necesario el uso, ni único ni alternativo, de los servicios a través de una interfaz web. Se hará referencia a ello en el apartado sobre líneas de trabajo futuras.

1.3. Desarrollo

Planteamos el trabajo en torno a los detalles prácticos de una solución concreta.

Efectivamente nuestro enfoque no es general ni teórico, al contrario y, como venimos anunciando, trata de mostrar los detalles prácticos de un despliegue concreto, contorneado por las necesidades enunciadas en el apartado anterior y definido por unos protocolos e implementaciones concretas en un número de máquinas concretado igualmente de antemano. Veamos:

[1] Con éstos estamos excluyendo situaciones de creación masiva de información que almacenar, como por ejemplo logs centralizados, sistemas SCADA importantes, datos experimentales ingentes (ATLAS del LHC, por ejemplo), etc

[2] http://www.openafs.org/pages/success.html
https://www.secure-endpoints.com/public/afs/

1.3.1. Elementos de nuestro despliegue

Utilizaremos una tabla sinóptica:

Recurso	Implementación
Máquinas	2 compatibles Intel x86
S.O.	Debian 6 "Squeeze"; Linux-PAM 1.0, nss-pam-ldapd 0.7
SSH	OpenSSH
V.P.N.	OpenVPN
DNS	Bind9
NTP	Ntp.org
LDAP	OpenLDAP
SQL	PostgreSQL
KERBEROS	MIT Kerberos
AFS	OpenAFS
XMPP/JINGLE	Jabberd2; Pidgin, Finch (cli)
SMTP	Exim4; Mozilla Thunderbird, Mutt (cli)
IMAP	Dovecot; Mozilla Thunderbird, Mutt (cli)
MANAGESIEVE	Dovecot; Sieve-connect (cli)

- Claramente la tabla deja ver tres grupos de elementos:
  - el primer nivel es el bajo nivel (hardware vestido por un SO interconectado) y que no es el objeto de este trabajo pero necesariamente debe organizarse como su punto de partida.
  - el segundo grupo constituirá nuestra "Infraestructura de Servicios Base", pues son base del funcionamiento del tercer grupo:
  - el tercer grupo son los "Servicios Finales de Usuario".

- Anotación: XMPP solía llamarse Jabber por la organización que tenía detrás (hasta el 2007[3]), y es por ello que en alguna ocasión podremos hablar de éste con ese nombre.

[3] http://xmpp.org/xsf/press/2007-01-16.shtml

1.3.2. Justificación

Podemos distinguir varios niveles en que interpretar nuestra infraestructura, y cada uno de ellos nos da las nociones básicas para justificar la presencia de los elementos que acabamos de enumerar.

En telemática y en informática en general, tres abstracciones representan un sistema: la computación, el estado, y la entrada/salida (i/o). El administrador puede manipular las implementaciones de estas abstracciones para conseguir cualidades que supongan una ventaja en su labor y en la calidad del servicio:

- Computación: en nuestro contexto consiste en efectuar modificaciones en cuanto a la programación del servidor, o al ajuste de su desempeño a través de ciertas variables establecidas por el programador. No insistiremos más sobre ésto porque en esencia son actividades del programador más que del administrador.

- Entrada/Salida: el administrador puede distribuir la i/o, en nuestro contexto ésto implica el despliegue de varias instancias de un mismo servidor en computadores diferentes. Ésto mejora:

  - la disponibilidad y resistencia: si un servidor deja de estar operativo, los clientes pueden (descubrir y) utilizar el resto de servidores para efectuar la i/o, de forma que el servicio en general no se interrumpe.
  - la escalabilidad: la carga de trabajo puede crecer a la vez que se mantiene la habilidad del sistema para ofrecer una calidad de servicio, simplemente porque la carga se reparte entre los servidores.

- Estado: en nuestro contexto nos referimos a los metadatos, y el administrador podría:

  - centralizarlos: es decir hacer que todos se almacenen tras un gestor de bases de datos (DBMS) y no donde se sitúe cada servidor que los utiliza. Ésto resulta en:
    - eficiencia: cada vez que haya que administrar los metadatos (añadir, modificar, eliminar) los cambios se realizan una vez y en un sólo sitio. De lo contrario (y teniendo en cuenta que tenemos distribuída la i/o) para dar el trabajo por hecho tendríamos que realizar la modificación instancia por instancia del servidor en la red.
    - negativamente, implica un punto débil en la infraestructura ya que si el DBMS cae, el servicio cae. Ésto nos lleva a la siguiente manipulación[4]:
  - redundarlos: es decir, de nuevo distribuir varias instancias pero en este caso del gestor de bases de datos, y activar algún mecanismo que replique los cambios entre todas. Como fruto de estas manipulaciones obtenemos:

[4] En ciertos sistemas, se podrían citar también problemas con la concurrencia, con la escritura de datos viejos.

    - disponibilidad: entonces las lecturas/escrituras[5] podrán realizarse en cualquier gestor que aún siga en pie, y la recuperación de los gestores con problemas irá seguida de un proceso de replicación que sincronizará el estado de los metadatos. Por tanto la disponibilidad se asegura.
    - negativamente, podemos tener problemas de coherencia (que la nueva versión del estado no llegue a sincronizarse antes de que los datos vuelvan a ser usados) y consistencia (cuándo estarán disponibles desde la escritura), excepto cuando la replicación es síncrona.
- Por tanto, resumidamente, si un modelo de gestión de metadatos admite un gestor al que puedan hacer peticiones varios clientes remotamente, puede participar de un modelo de metadatos centralizado. Si además el gestor permite replicación de actualizaciones entre varios gestores, puede participar de un modelo de datos centralizado que además sea redundado. Si la replicación para la redundancia es síncrona, la consistencia será transaccional (atómica). Podemos añadir también que, además del carácter síncrono, la replicación para la redundancia puede ser Master-Slave (implica poder leer en cualquier gestor, pero escribir sólo en uno que será el origen de todas las replicaciones) o la replicación para la redundancia puede ser Master-Master. Ésta última ya sí implica poder leer y escribir en cualquier gestor, de forma que todos se configuran igual. Por tanto es un esquema más eficiente administrativamente, y además aumenta la disponibilidad de modificaciones porque hay más oportunidades de introducirlas en el sistema, pero puede presentar más problemas de escalabilidad. En nuestro despliegue, siempre que nos sea posible nos decidiremos por esquemas Master-Master (en algunos casos Multi-Master).

Las propiedades (disponibilidad, escalabilidad, eficiencia) ventajosas que ofrecen las técnicas anteriores tienen en común que necesitan distribuir los servicios y por tanto ***justifican nuestra decisión de realizar el despliegue sobre más de una máquina***, escogiendo el número mínimo para ello, dos.

Hasta ahora hemos definido nuestro proyecto en términos de i/o y estado, y éso nos ha permitido comentar ciertas técnicas que justifican la elección de 2 computadoras. Otra forma de evaluar nuestro sistema es atendiendo a la interpretación de esos metadatos, qué representan.

- Efectivamente, recordando al ejemplo que nos ponía en contexto en el primer apartado, la información es labor de los usuarios, la metainformación y el despliegue de instancias de un servidor son labor del administrador. Por tanto, nos sería ventajoso poder separar la gestión de metadatos del resto, de forma que podamos tratar ésta eficientemente. Para conseguirlo, debemos además atender a que todos los metadatos no representan lo mismo, si bien sí que pueden distinguirse funcionalmente un número finito de grupos. ***Estos grupos de metadatos justificarán la presencia***

[5] Al incluir las escrituras, implícitamente nos referimos a esquemas Master-Master, nuestra preferencia, frente a Master-Slave.

de todos esos servicios que no son los servicios finales (correo SMTP/IMAP, mensajería XMPP...) ***sino que constituyen una "Infraestructura de Servicios Base" a los finales***:

- metadatos de autenticación →KERBEROS, un sistema y protocolo para las autenticaciones que provee SSO.
- metadatos de autorización →LDAP, almacén optimizado para lecturas, luego almacén de metadatos estables; modelo de directorio.
- metadatos de cuentas (específicos de cada servicio)→LDAP, SQL. En el caso del RDBMS SQL, es un almacén optimizado para escrituras, luego almacén de metadatos volátiles; modelo relacional.
- otros de configuración de los servidores, destacando globalmente→DNS. Efectivamente en un entorno distribuído, no se sabe dónde pero tampoco cuántos recursos llegará a haber. Los registros SRV hacen que el sistema DNS, además de ser un sistema para resolver Direcciones IP↔Nombre de Dominio[6], se convierta también en un sistema para resolver los recursos (IP's, Puertos, etc de los servidores) en un entorno distribuído dado su nombre. Su papel es central:

  DNS+SRV = Host↔IP + NombreEntornoDistribuído↔Recursos

- Anotaciones:

  - El servicio KERBEROS y, por otro lado, cualquier susbservicio de replicación de bases de datos, necesita que los relojes de los participantes estén sincronizados. Ésto explica la adición ineludible de servidores NTP que, similar al caso de DNS, configura de forma global una variable del sistema en su conjunto (en su caso el reloj).
  - El servicio de ficheros distribuído AFS puede considerarse servicio final pero también base de los finales. Ésto es así porque el correo por ejemplo, necesita almacenar ficheros y puede utilizar a AFS para ello (así ocurrirá en nuestro caso, con ciertas precauciones).
  - Sistema AA_. Aunque es un concepto que suele utilizarse en el ámbito de los servidores de acceso a redes (NAS, como los puntos de acceso Wi-Fi), es un sistema que describe muy bien nuestra gestión de metadatos. Un sistema es AA si utiliza protocolos para Autenticar y Autorizar. Un sistema es AAA si además utiliza protocolos para Accounting (contabilizar el gasto de recursos durante el uso del servicio, por ejemplo para tarificar el servicio ofrecido). Una cuarta A podría emplearse para indicar Auditoría. En nuestro caso el sistema dispone de protocolos para las dos primeras A (autenticación y autorización). Además, podemos mencionar que KERBEROS y LDAP pueden usarse en otros sistemas como backends de servidores de protocolos AA; por ejemplo el software FreeRADIUS ofrece a los clientes el protocolo RADIUS para sistemas AAA,

[6] Algo que en KERBEROS forma ya parte del sistema de verificación de algunos principals.

entonces el cliente se comunica con él a través de este protocolo RADIUS para autenticarse, autorizarse etc y acceder a la red, FreeRADIUS puede utilizar KERBEROS y LDAP para hacer efectivos esos procesos.

Las dos visiones anteriores de nuestro sistema (estado, i/o; grupos funcionales de metadatos), nos han permitido justificar el número de máquinas así como la necesidades de interponer una "Infraestructura de Servicios Base" con protocolos bien definidos y en los que se apoyará el trabajo de los servicios finales. Nuestra tercera visión del sistema surje de evaluar qué representan no los metadatos sino los datos, la información relacionada a cada servicio. ***Pero, puesto que estamos ante los servicios que utilizarán los usuarios, partiremos de las necesidades de éstos; pretendemos comprobar si los protocolos finales desplegados son pertinentes y, por fin, queda justificado el sistema completo***. Siguiendo el modelo la clasificación de servicios Berkom, los servicios son:

- Persona-Sistema.
- Persona-Persona.
  - Síncronos.
  - Asíncronos.

Evidentemente, los protocolos escogidos proporcionan servicios Persona-Sistema (Cuentas Centralizadas de Shell POSIX con OpenAFS), Persona-Persona síncronos (Mensajería Instantánea con XMPP, Telefonía con XMPP-Jingle/RTP), Persona-Persona asíncronos (Correo con SMTP/IMAP/MANAGESIEVE). Ciertamente el modelo Berkom distingue varios tipos de servicios Persona-Sistema pero, por un lado es normal que no implementemos todos los posibles, y por otro el servicio que ofrecemos en ese caso es especialmente potente.

Resumen, diagrama 3 capas:

tipos de servicios	persona-sistema, persona-persona...	Protocolos finales
grupos metadatos	eficiencia, disponibilidad, SSO, descubrimiento	Infraestructura base
estado, i/o	eficiencia, disponibilidad, escalabilidad	nº máquinas empleadas

Abstracciones	Manipulados aportan a nuestros sistema	Justifica

Puesto que nuestro despliegue pone principalmente de relieve la gestión de los metadatos, la siguiente representación gráfica del proyecto intenta poner de manifiesto el flujo de información y metainformación entre la infraestructura base, servidores de servicios finales y sus clientes:

- En general, el diagrama expresa que

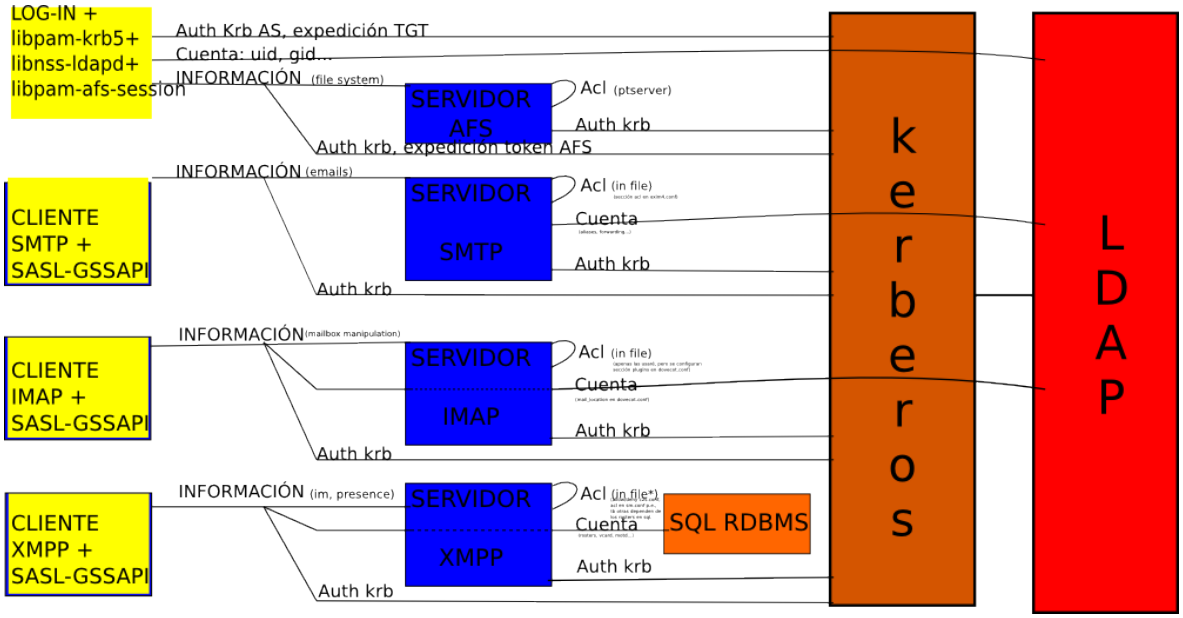


Figura 1.1: Diagrama de flujos aproximados de la información y metainformación (Auth/ACL/Cuentas) asociada a cada servicio. No se incluye, DNS (si bien los clientes lo utilizan para localizar los servidores) ni NTP.

  - toda autenticación ("Auth") implica el uso de KERBEROS.
  - toda resolución de metadatos de Autorización ("ACL") u otros específicos de cada servicio ("Cuentas") pasan por una consulta LDAP. Hay una excepción a ésto, y es que OpenAFS no soporta LDAP para almacenar esta información y en su lugar utiliza sus propios servidores y sistemas (ptserver) de replicación.
  - en el caso del servidor XMPP de mensajería instantánea, un subconjunto marcadamente volátil de metadatos de las cuentas (por ejemplo el estado de actividad de un usuario) se almacena en PostgreSQL.

- Todos y cada uno de los elementos del diagrama estarán desplegados por duplicado (una instancia en dklab1, otra en dklab2).
- En el servicio de Shell, los nombres bajo log-in son implementaciones concretas en Debian de los módulos responsables de las autenticaciones y metadatos de este servicio; sus nombres resaltan su propósito y pueden ser clarificadores:

  - uno (el reflejado para Debian es libpam-krb5) se autentica contra el AS (Autentication Server) KERBEROS. La autenticación contra el AS es interactiva, conlleva un prompt para recibir el nombre de usuario y contraseña y, si es exitosa, el AS expide una credencial TGT (Ticket-Granting-Ticket, imagínese como un pequeño fichero que se almacena en algún lugar prefijado). Esa credencial se utiliza para conseguir automáticamente otras, los ticket ST (de Service-Ticket) pidiéndola al KDC (Key Distribution Center, el otro componente en un despliegue para KERBEROS). Nótese que el TGT representa la entrada en

el sistema del usuario, pues ha de superar un log-in interactivo con el AS y es por ello que el SO delega la autenticación en el AS y no pide nada más para considerar autenticado al usuario. Sin embargo los demás servicios, ventajosamente no delegan ya su autenticación al AS, sino que utilizan el TGT que se consiguió entonces para evitar un prompt que disturbe al usuario: sólo es necesario que el cliente del servicio disponga de algún módulo que automatice el proceso de conseguir el ST a partir del TGT y entregarlo al servidor del servicio, que dará por completada la autenticación.

    - para el servicio de sistema de ficheros AFS el cliente (el SO en este caso) dispone de un módulo (el de Debian reflejado en el diagrama es libpam-afs-session) tal que nada más se reciba ese TGT se consiga el ST para AFS y el usuario entre directamente (es decir, que su CWD -de "Current Working Directory"- corresponda a) su espacio home en AFS.
    - para los demás servicios se ha estandarizado bastante el proceso: sea cual sea el servicio en concreto de nuestro despliegue, el cliente (el programa con que accedamos al correo, o con el que chateemos...) tendrá un módulo que implementa SASL-GSSAPI, un subprotocolo de negociación de autenticación pero que implica finalmente usar el TGT para conseguir el ST como decíamos.

  - El SO tiene otro módulo (en el caso de Debian puede ser libnss-ldapd) que lanza las consultas LDAP para recabar información de la cuenta de Shell. En el caso de los demás servicios, no son los clientes los que, en general, lanzan consultas para obtener sus metadatos de cuenta porque, de necesitarlos, el protocolo con el que dialoguen con el servidor ya tendrá soporte para mandárselos, de forma que, como se ve en el diagrama, son los servidores los que los recaban desde LDAP. Un ejemplo (no reflejado en el diagrama pero sí en nuestro despliegue) donde el cliente sí puede requerir también consultar directamente LDAP es el correo, donde los clientes usan LDAP para conseguir o autocompletar el nombre completo y la dirección de correo de un destinatario.

- Nótese que hay una línea que conecta directamente el bloque de KERBEROS con el de LDAP. Ésto será porque MIT Kerberos almacenará su metainformación ("principals") en LDAP; ésto no puede hacerlo OpenAFS y por ello no existe una línea similar entre ambos. Además la unión tiene un segundo sentido, ya que LDAP puede usar KERBEROS (presentar un ST conseguido a partir de un TGT etc) como esquema de autenticación para sus clientes (excepto, claro, cuando el cliente es MIT Kerberos utilizándolo de almacén, caso en el que se usan credenciales almacenadas en un fichero y que no tienen nada que ver con ST o TGT).

- AFS se representa como servicio final, pero también puede considerarse un servicio base para cualquier servicio final que necesite almacenar ficheros. En nuestro despliegue también se dará efectivamente esa situación porque el sistema de correo lo utilizará para almacenar, efectivamente, los correos.

1.3.3. Desarrollo del despliegue

A continuación se da un repaso general, el despliegue en sí se encuentra en los anexos que iremos referenciando.

Secuencia general del despliegue

Es lineal.

- Testbed: se presentan dos imágenes de Debian y el emulador qemu como las dos máquinas del despliegue. En conjunto, las llamaremos "testbed", por separado una se llama "dklab1" y la otra "dklab2". Además, puesto que OpenSSH añade valor a la labor administrativa desde ese temprano momento, se instala entonces.
- Infraestructura de Servicios Base:
  - Configuración de red IP; muestra de despliegue de una VPN con OpenVPN.
  - DNS con Bind9
  - NTP con Ntp.org
  - LDAP y KERBEROS con OpenLDAP y MIT Kerberos.
  - AFS con OpenAFS
- Servicios Finales:
  - Cuentas Centralizadas de Shell POSIX
  - XMPP y Jingle/RTP con Jabberd2
  - SMTP, IMAP y MANAGESIEVE con Exim4 y Dovecot.
- Cómo incorporar una nueva máquina a nuestro parque informático, y cómo crear una nueva cuenta centralizada a un usuario nuevo de nuestra organización. Para no ser reiterativos se incluyó este capítulo como apartados de otro anterior: el de Cuentas Centralizadas de Shell POSIX.

En cada capítulo se incluye:

- Exposición breve de los fundamentos de cada protocolo e implementación empleada. Referencias a los documentos y páginas web oficiales. Justificación de su elección.
- Descripción paso a paso de todas las acciones de administración necesarias para desplegar el servicio con un nivel de detalle tal que otra persona pudiese llevar a cabo el despliegue independientemente y con la sola ayuda de esta documentación. Las acciones de administración a las que nos referimos se traducen básicamente en:
  - ejecutar comandos en un intérprete de comandos,
  - y editar ficheros de configuración.

Instrucciones para seguir los anexos

Todos los anexos están redactados siguiendo unas reglas diseñadas por nosotros. Es decir, la presentación de los comandos que introducir y los ficheros que editar se acompaña de ciertas convenciones que reforzarán el significado de lo que se está haciendo. Si se escoje alguno de los anexos y se abre en alguna página intermedia, se podrá comprobar que:

- Cuando se expone un comando o edición de fichero, se realiza un punto y aparte y se estrecha el margen respecto al texto explicativo. Ésto es básico evidentemente, pero además nos sirve para introducir elementos más interesantes:
- El nuevo contexto tendrá un color de fondo que identifique al host:
  - Rojo claro: significa que esa acción se realiza en dklab1.
  - Azul claro: significa que esa acción se realiza en dklab2.
  - Blanco: significa que esa acción ilustra un caso hipotético: una máquina que no es dklab1 ni dklab2, o una acción para dklab1 o dklab2 en un supuesto imaginario (como contraejemplos) tal que no se deben efectuar realmente.
  - Salidas de los comandos: éstas son siempre grises y planas (sin lado perfilado alguno, véase a continuación) y con tamaño de tipografía algo más pequeño. Puesto que simpre representan la salida del comando anterior, no nos es necesario colorear el fondo en azul o rojo porque el host ya lo indica el contexto inmediatamente anterior, mientras que cambiándolo al gris resaltamos que es una salida en la terminal (usualmente de fondo negro).
- El nuevo contexto tendrá algún lado perfilado que identifique el tipo de acción:
  - Lateral izquierdo: aparece cuando la acción administrativa es la *ejecución de comandos*, e intenta aludir al prompt que la shell suele disponer a la izquierda de la pantalla para recibir los comandos. Además:
    - Si este lateral es oscuro, la shell es de root (uid=0). Intenta aludir al denso carácter # del prompt para root típico de una shell.
    - Si el lateral es blanco, la shell es de una cuenta no privilegiada. El no tener color, alude a no tener permisos de administrador.
  - Superior e inferior: entonces la acción administrativa es la *edición de un fichero*. Puesto que a veces tendremos que editar ficheros muy largos, en ocasiones nos encontraremos varios de estos contextos seguidos (uno por página por ejemplo) y, para saber qué se está editando en esas páginas deberíamos retroceder hasta encontrar el contexto de comando en que se llamó al editor, Vim. Puesto que el logo de Vim es verde, la línea superior e inferior son verdes.
- Política de edición de ficheros:

- Utilizaremos marcas para distinguir lo que hemos editado respecto de la configuración por defecto que traen los paquetes. Esas marcas de principio y fin serán "####fx:" y "####endfx" pudiendo sustituir el símbolo "#" por otro que se interprete como inicio de comentario en el fichero en cuestión.
- Si necesitamos evitar la interpretación de una parte predefinida de la configuración, en lugar de borrar esa parte, la convertimos en comentarios y la diferenciamos de nuestros verdaderos comentarios antecediéndolos con una marca, en concreto un guión "-"[7].
- Sólo expondremos las partes que editamos, de forma que antes o después de las marcas de inicio y fin, habrá puntos suspensivos o una referencia de lo que la configuración por defecto tiene por esa zona del fichero y puntos suspensivos. Si no se ven los puntos suspensivos por encima de nuestra marca de inicio, será porque nuestra edición se sitúa al principio del fichero, si no los hay abajo porque se sitúa al final, etc. Por tanto, esos puntos suspensivos no deben aparecer.

- Política de nombres del software: a veces el nombre de un proyecto, un protocolo y un programa es el mismo o parecido. Hemos intentado procurar no utilizar un nombre cuando realmente nos referimos a otro (ejemplo un protocolo cuando queremos referirnos a un proyecto que lo implementa), siguiendo además que:
  - El nombre del protocolo se escribirá en mayúsculas (suele ser unas siglas/acrónimo): "SSH".
  - El nombre del proyecto se escribirá comenzado con mayúscula: "OpenSSH".
  - El nombre del programa esté completamente en minúsculas: "ssh".

- Posibles erratas: los anexos en su conjunto ocupan más de 800 páginas. Desafortunadamente no hemos tenido oportunidad de depurar las explicaciones y comentarios que allí se presentan, pudiendo incurrir en errores de estilo. A este respecto, diremos también que se ha procurado utilizar el plural de autoría y el tiempo verbal presente del indicativo. Además, hasta casi el final de su redacción los anexos se idearon como parte de la memoria, luego nunca se aluden a sí mismos como anexos que son ahora. Independientemente de las explicaciones y comentarios, el despliegue en sí (comandos, configuraciones) debiera estar completamente libre de errores.

- Adaptación de los anexos a un dominio dado: como se ha comentado, los anexos describen el despliegue con todo detalle, tal que para reproducirlo basta ir copiando y pegando desde aquel. Este planteamiento es ciertamente atractivo cuando no se dispone de mucho tiempo, pues el despliegue es largo. No obstante, si queremos utilizarlo directamente (es decir, utilizarlo no para desplegar sobre dos máquinas virtuales, sino sobre reales y definitivas), nos encontraremos que todos los comandos

[7] Inicialmente esa marcar era el carácter "·", pero problemas del coloreador de sintaxis nos obligaron a elegir otra algo más ambigua.

y ficheros de los anexos utilizan un nombre (de dominio, de REALM Kerberos, de DIT LDAP...) mientras que en el lugar de despliegue definitivo se usarán otros. Idealmente, podríamos sustituir todos esos nombres de la documentación con los que corresponderían en el nuevo entorno. A este respecto, en dklab2.ovl (la copia de nuestro despliegue en dklab2 presente en el DVD de este proyecto) se incluye:

- Una carpeta /root/DOC con el código fuente LaTeX y LyX de la documentación de este proyecto. Además se incluyen algunas instrucciones adicionales de un modo informal, para preparar las dependencias de compilación de la documentación.
- Un script sed que realiza las sustituciones (cuatro tipos), y que podemos modificar con los nombres del nuevo entorno. Por tanto, utilizando el propio dklab2 (u otra máquina), podemos obtener una versión de los anexos lista para copiar y pegar. Orientativamente, por tanto:
  1. cp -Rf /root/DOC /var/tmp
  2. cd /var/tmp/DOC/latex
  3. view INSTRUCCIONES # y síganse las indicaciones excepto la ejecución final de fx_build.sh que debe ser postpuesta aún
  4. vim Utils/filtro_dominio_base.sed #modificamos convenientemente los 4 filtros.
  5. sed -i -f Utils/filtro_dominio_base.sed *lyx #aplicamos los filtros
  6. Utils/fx_build.sh #creamos los documentos
  7. ls ../cdrom/DOCUMENTOS/ANEXOS/
- El nombre de dominio utilizado en los anexos es: "casafx.dyndns.org". Las razones que nos llevan a presentar éste son:
  - Durante el desarrollo lo usamos, debido a que en algún servicio (el correo) necesitamos comunicarnos desde las máquinas virtuales con servidores de otras redes (Gmail, Live) pasando por el SO host. Para recibir una respuesta etc, pensamos en ponerle un nombre resoluble externamente a la máquina con el SO host de forma que las respuestas a acciones de las máquinas virtuales pudiesen responderse al SO host al menos.
  - A posteriori, no hemos querido sustituirlo por el dominio reservado example.org[8] utilizando el script sed anterior por una razón: nosotros sólo hemos desplegado con casafx.dyndns.org, es el único que sabemos con seguridad si induce o no a algún error peregrino (de hecho, el cliente XMPP Psi nos pareció que tenía problemas -un bug- con REALM de más de 2 niveles).

Vamos ya a reseñar los puntos más relevantes de cada parte del despliegue y a referenciar el anexo que lo detalla.

[8] http://tools.ietf.org/html/rfc2606

1.3.4. Testbed, OpenSSH

Anexo:	0

El presente proyecto comenzó a ser desarrollado en dos máquinas, un portátil y una torre, que teníamos completamente disponibles. De hecho, el portátil era la máquina con que trabajábamos habitualmente (prácticas de la titulación, vida online...) y la torre se utilizaba para ofrecer servicios de oficina doméstica: impresora y escáner en red, servidor de ficheros SMB, modesto servidor web... Ambas máquinas usaban el mismo SO, Debian GNU/Linux, pues era capaz de satisfacer ambas formas de trabajo con computadoras. Como expondremos más tarde, estos equipos cumplen varios requisitos para desarrollar nuestro proyecto, pero no todos. A medida que avanzábamos en nuestro recorrido, fuimos considerando un acierto nuestra elección del número de máquinas, dos, y su SO, Debian, pero se descubrió como un error el que ambas estuvieran modificadas previamente para, como dijimos, adaptarse al uso de ordenador personal por un lado, y a servidor de una pequeña oficina doméstica por otro. La razón es que el despliegue que contempla este proyecto es muy complejo, depende del preciso ajuste de un gran número de variables y cualquier divergencia puede conllevar una denegación de servicio por parte de alguno de sus componentes, en muchos casos (es nuestra experiencia) acompañado de un críptico mensaje de error. Por tanto, si el planteamiento óptimo en torno al despliegue de servicios es detallar los pasos de forma que pueda ser reproducido por otras personas, entonces nosotros diremos que la única forma que tenemos de hacer reproducible todo lo que se dirá aquí es proponiendo un entorno de referencia. Ese entorno de referencia será aquel en que nosotros hemos logrado desplegar el proyecto y por tanto donde podemos garantizar que desde un estado inicial y siguiendo nuestras indicaciones, el sistema funciona.

Por tanto, llamamos "testbed" a un par de imágenes con Debian 6 "Squeeze" preinstalado, que constituyen el entorno de referencia, punto de partida y soporte del despliegue para todo este proyecto. Cada imagen ocupa inicialmente unos 250 MB de una instalación mínima.

A continuación pretendemos justificar por qué nuestra elección de Debian. Después pasaremos a discutir qué tecnologías usaremos para lanzar ambas imágenes.

Elección del Sistema Operativo

Tres son los requisitos que debe tener:

- Debe tener completamente disponible y reciente la pila de software que necesitamos (todos los servidores, clientes y librerías de este proyecto).
- Debemos tener suficiente dominio sobre él tal que podamos investigar con garantías cualquier problema. Adicionalmente, si es de uso mayoritario obtendremos además ventaja de todo el bagaje que, disperso por la Web, podremos encontrar cuando necesitemos resolver alguna duda o problema puntual.
- Idealmente debe estar orientado al uso en servidores: ser estable, fiable, seguro y fácil de mantener.

La versión de Debian GNU/Linux 6 "Squeeze" para i386 cumple todos esos requisitos y, como habíamos anunciado, es la que usamos finalmente. La versión 6 es, específicamente, la última que el proyecto liberó como versión estable[9]. Finalmente diremos que, a modo de aval, la elección de Debian es bastante común entre organizaciones que realizan despliegues del estilo del que nosotros efecturemos.

Elección de la técnica de virtualización

Efectivamente, el lanzamiento de las imágenes utilizando alguna técnica de virtualización nos permite llevar a cabo el despliegue sin necesidad de máquinas dedicadas a tal fin, incluso a falta de otros recursos nuestro testbed se puede desarrollar en las máquinas previstas para un entorno en producción, proveyendo la técnica de virtualización el aislamiento necesario para poder operar en uno y otro entorno sin intromisión alguna.

Los requisitos que contemplamos son:

- Deben existir implementaciones para una amplia variedad de arquitecturas de microprocesador y SO host ("anfitrión"), de forma que cualquiera lo pueda poner en marcha donde más le convenga.
- Debe soportar la arquitectura y SO guest ("invitado") que ya hemos elegido: Debian 6 "Squeeze" sobre i386 (forma en que el proyecto Debian denomina a la arquitectura x86 de Intel).
- No debe depender de hardware especial más allá de lo mencionado.
- Debe implementar robustamente soporte para:
  - Discos.
  - Video.
  - Audio.
  - Red, incluyendo:
    - Posibilidad de simular la interconexión por red de dos instancias del emulador.
    - Posibilidad de utilizar la conexión a internet del SO anfitrión.
    - Posibilidad de redirijir puertos del SO anfitrión al invitado.

Podríamos añadir otro criterio más, el coste, ya que en ningún momento nos planteamos técnicas que supusiesen un gasto económico adicional. En el contexto del Software Libre teníamos varias tecnologías que fuimos descartando.

Por un lado, encontramos la *paravirtualización asistida por hardware* (KVM[10], Xen[11]...), pero su necesidad de extensiones especiales en el microprocesador (algo comunes hoy en

[9] Ésto ocurrió el de Febrero del 2011.
[10] http://www.linux-kvm.org
[11] http://xen.org/

día pero de las que nosotros no disponíamos) nos hizo desecharla desde el principio. Las dos siguientes técnicas serían la *contextualización* por un lado, y la *traducción completa* por otro. El proyecto Qemu brinda ésta última y además cumple sólidamente todos nuestros requisitos, pero el hecho de que toda computación en la máquina invitada deba ser traducida a la máquina anfitrión lo hace muy lento. Por otro lado la contextualización constituye la técnica de virtualización más rápida de las tres, pero es dependiente del SO anfitrión.

Una primera solución fue seguir dos vías: daríamos los pasos necesarios para usar la rápida contextualización LXC de Linux si se usaba este SO como anfitrión, mientras que mantendríamos a Qemu como alternativa en el caso de no usar Linux. Ciertamente, se superaron muchas de las dificultades[12], pero no todas: no la que sobrevino con OpenAFS.

Efectivamente el cliente AFS y "Cache Manager" está implementado como un módulo del kernel, el cual no tiene posibilidad alguna de virtualización: o lo ejecuta el SO anfitrión, o lo hace un invitado, o lo hace el otro invitado. Ésto desvirtuaba nuestro despliegue y, por tanto, LXC se desechó.

Qemu es la única solución de virtualización que usamos.

Elección del formato e imagen utilizada

Qemu soporta una cierta variedad de formatos de imágenes, desde la utilización directa de discos duros o particiones en el SO anfitrión hasta imágenes en bruto de estos últimos u otros especiales para técnicas de virtualización. Sin perder todas esas posibilidades de vista, los requisitos que proponemos nosotros son:

- Puesto que no conocemos de antemano el tamaño que ocupará todo el proyecto, decidimos utilizar un formato de imagen que pueda ir creciendo según las necesidades.
- Puesto que necesitamos crear más de una imagen, pero ambas tendrán un origen común, sería especialmente económico que cada imagen se pudiese crear como diferencial de una imagen base. Ésto redundaría también en el tamaño de las copias de respaldo que hiciésemos durante nuestro desarrollo.
- Complementando las ventajas anteriores, debiera existir la posibilidad de tener una interfaz estándar de dispositivo de bloque a las imágenes, de forma que pudiésemos montarlas en el SO sin necesidad de arrancar Qemu, o que otras soluciones de virtualización interesantes para otras personas tuviesen la oportunidad de arrancarlas a través de esa interfaz estándar.

[12]Fue especialmente útil aprovechar la posibilidad de crear espacios de nombres per-proceso de Linux (que habíamos leído en algún artículo) para satisfacer las visiones contrapuestas que debían tener LXC y Qemu respecto del sistema al arranque. Es decir, si LXC necesitaba que no existiese cierto script de inicio, la técnica mencionada permitía mapear ese script a /bin/true para que fuese inconsecuente, y todo ello sin modificar realmente el sistema de ficheros. Además, la presencia de una variable de entorno permitía detectar si la máquina la arrancaba LXC o Qemu y por tanto hacer que el mecanismo se llevase a cabo o no.
http://glandium.org/blog/?p=217

Un formato que cumple esos requisitos es Qcow2, y su funcionalidad de imágenes diferenciales respecto de una base se llama Overlay's. Adicionalmente ofrece otras posibilidades como compresión, encriptación, soporte de snapshots en el mismo formato que, en cualquier caso, no usamos.

Instalación del SO; creación de dklab1 y dklab2

Nuestra intención es crear una imagen en formato Qcow2[13] donde resida una instalación mínima de Debian 6 "Squeeze" que nos sirva como punto de partida. Una posibilidad es crear una imagen Qcow2 vacía y, utilizando a Qemu e imágenes netinstall[14] u otros medios que pone Debian a nuestra disposición, realizar esa instalación mínima paso a paso.

Pero no es necesario, Debian (uno de sus DD -Debian Developer-) pone a disposición esas imágenes en el formato y condiciones que nosotros necesitamos.

Es por ello que en lugar de realizar una instalación mínima, podemos descargar directamente la imagen desde:

- http://people.debian.org/~aurel32/qemu/i386/

A partir de ella creamos dos overlay's que representarán las dos máquinas que componen nuestro despliegue, dklab1 y dklab2[15]. Pueden leerse todos los detalles en el apartado 1.1 del anexo.

Lanzamiento de las imágenes.

Tomadas las decisiones iniciales, y disponibles las imágenes y el emulador, llega el momento de levantar las imágenes y modificarlas para poder encarar el despliegue de los servicios.

En el apartado 1.3 del anexo se detallan qué opciones acepta Qemu para ofrecer todas las funcionalidades que necesitamos:

- Creación de dos tarjetas de red: una de ellas representa el hardware que enlazaría el SO invitado y el anfitrión, la otra representa el hardware que enlazaría dklab1 y dklab2.
- Creación de una redirección de puertos que comunique el SO anfitrión con el puerto TCP/22 del SO invitado.
- Otras opciones que nos permitirán incluir hardware virtual emulado por Qemu.

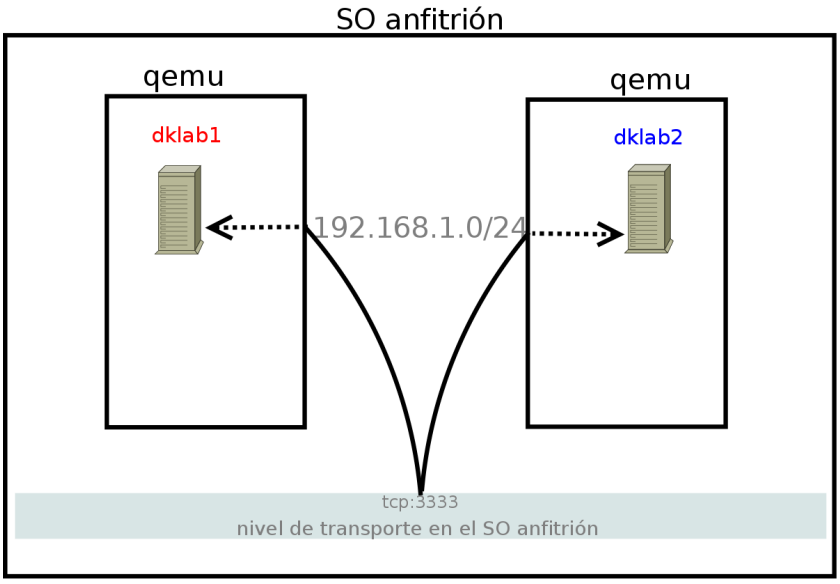
[13] http://people.gnome.org/~markmc/qcow-image-format.html

[14] http://www.debian.org/CD/netinst/

[15] El origen de los nombres viene del nombre del SO y de las implementaciones de los servicios de la infraestructura base que albergan: **D**ebian, Mit **K**erberos, Open**L**DAP, Open**A**FS, **B**ind9 (sí, se echa en falta una referencia a Ntp.org, la implementación del servicio de tiempo). Quizás dkland1 y dkland2 nos parezcan, a posteriori, más apropiados. El nombre no refleja, por otro lado, que esas máquinas albergan también los servicios finales.

Efectivamente, necesitamos tres interfaces de red: la primera ("eth0") permite a dklab1 y dklab2 comunicarse entre sí y por lo tanto ser el soporte de red que necesitarán los servicios de nuestro despliegue. La segunda interfaz ("eth1") será aquella que nos permita comunicar las máquinas virtuales con Internet a través del SO anfitrión; ésto es útil por ejemplo para instalar online el software de nuestro despliegue, en lugar de recurrir a un DVD etc. Por último, la tercera es la interfaz de retorno "lookback", no es asunto de Qemu pues corresponde a una creación por software de la implementación IP del SO anfitrión, concretamente es la interfaz virtual de red que usa el segmento reservado de direcciones IP 127.0.0.0/8[16] para pruebas o IPC local ("Inter Process Communication"). Por tanto, de las 3 interfaces dos necesitan de Qemu la emulación de sendas tarjetas de red como se apuntó.

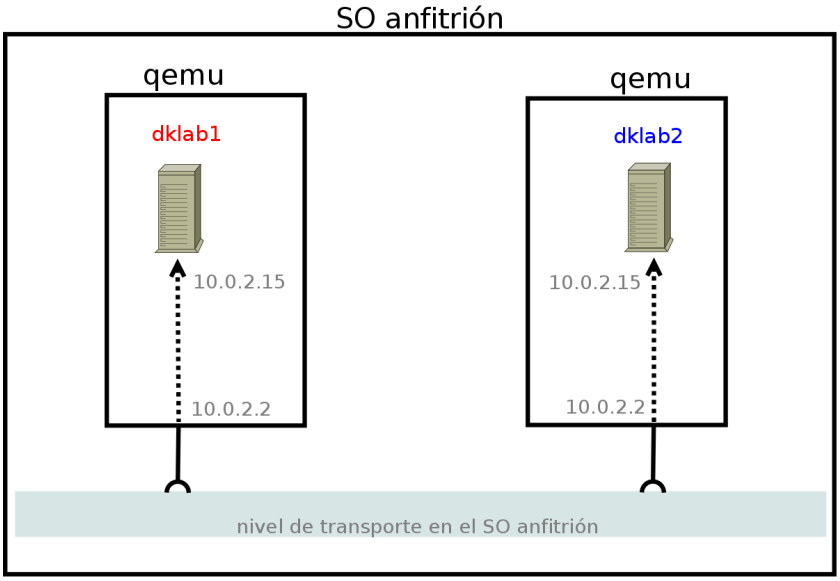
La primera tarjeta, la que representa el enlace entre dklab1 y dklab2, es solucionada por Qemu a través de una comunicación entre procesos pero que, para dotar de potencial, implementa con una comunicación TCP/IP. Así, una instancia de Qemu (dklab1) escucha en un puerto del SO anfitrión y la otra (dklab2) se conecta a éste. Este detalle de la implementación es importante porque implica un orden de lanzamiento de las imágenes que debe seguirse sistemáticamente: primero dklab1, después dklab2. Es también reseñable que, cuando se interconectan dos instancias, la dirección MAC que Qemu asigne en cada una a la tarjeta de red ha de ser explicitada para que resulten diferentes. Por último, indicaremos que el segmento de red que compartirán dklab1 y dklab2 ha sido la subred privada clase C: 192.168.1.0/24, donde 192.168.1.1 corresponderá a dklab1 y 192.168.1.2 corresponderá a dklab2.



La segunda interfaz, la que representa el enlace entre el SO invitado y el anfitrión, debe utilizar una solución totalmente distinta a la anterior de comunicación entre instancias Qemu. Concretamente, ahora debemos usar su módulo de red "user-mode"[17]. Éste

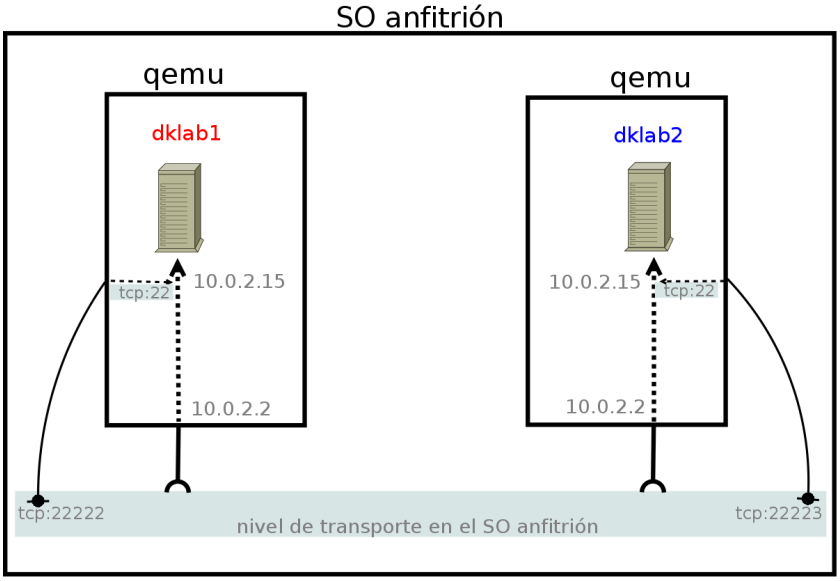
[16] http://tools.ietf.org/html/rfc3330
[17] http://wiki.qemu.org/Documentation/Networking#User_Networking_.28SLIRP.29

permite al SO invitado obtener la dirección de red a través de DHCP, resultando (si no se reconfigura de otra forma) en 10.0.2.15 para el extremo en el SO invitado, y 10.0.2.2 para el extremo en el SO anfitrión. Hay que decir que esa IP, 10.0.2.2 es sólo una construcción de Qemu para permitir operar al SO invitado y que pueda acceder a Internet, pero la implementación es de nuevo totalmente asimétrica y, del lado del SO anfitrión no existe interfaz alguna con esa dirección IP, sino que Qemu traduce las conexiones TCP necesarias (al SO invitado) como iniciadas por él[18].



Respecto a la redirección de puertos en el SO anfitrión al TCP/22 del SO invitado, ésto nos permitirá iniciar una sesión SSH en cualquiera de las máquinas virtuales desde el SO anfitrión, conservando la configuración de teclado de éste, ofreciendo la posibilidad de utilizar redirecciones de la salida gráfica de las aplicaciones, etc. Será la forma recomendada de administrar las máquinas virtuales, y de hecho se instará a instalar el servidor OpenSSH tras el primer arranque de cada máquina. Las redirecciones son, específicamente: que el puerto TCP/22222 del SO ainfitrión comunique con el puerto TCP/22 del SO invitado dklab1. De la misma forma el puerto TCP/22223 del SO anfitrión se relaciona con el TCP/22 para dklab2.

[18] Concretamente, user-mode soporta tanto TCP como UDP. Puesto que no soporta ICMP, no es posible utilizar ping.



Por último, debíamos indicar a Qemu la emulación de hardware adicional. Concretamente ese hardware se compone de un teclado con distribución española de las teclas, una tarjeta de video Cyrrus Logic y una tarjeta de sonido SoundBlaster16. Desafortunadamente, ésta última no funcionó (ni siquiera el simple soporte para el zumbador de PC), su presencia acababa provocando que Qemu consumiera todos los recursos de tiempo de la máquina. Lo consideramos un bug de los binarios en Debian 6 (donde nosotros usamos Qemu), y es posible que haya sido solucionado en versiones más recientes.

El apartado 1.3 del anexo tiene todos los detalles sobre las opciones de Qemu para activar las funcionalidades anteriores y arrancar finalmente las imágenes.

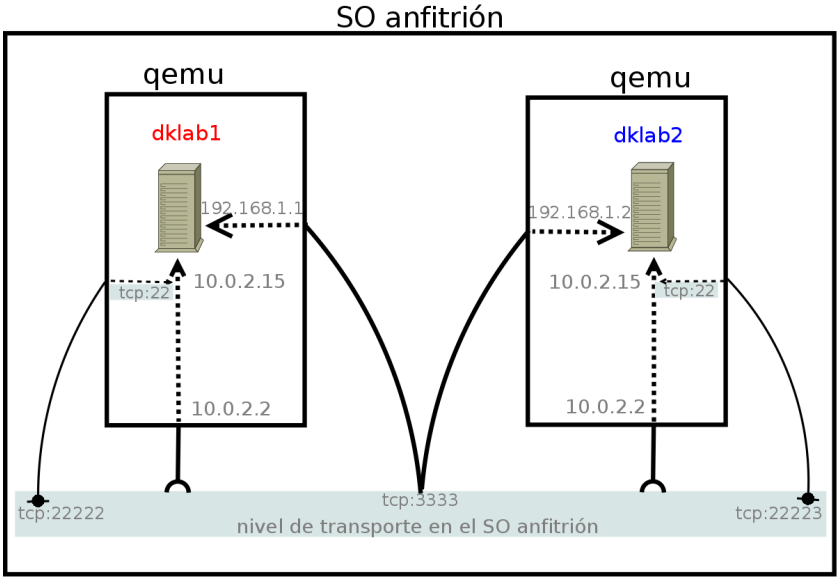
Instalación de OpenSSH y herramientas básicas en las imágenes

Una vez arrancadas las máquinas virtuales con las opciones que se detallan en el apartado 1.3, debemos modificar las imágenes para que, una vez comencemos a desplegar servicios, tengamos instalado todas las herramientas que pudieran ayudarnos en el camino, así como una configuración de red funcional.

Nosotros aconsejamos instalar la implementación OpenSSH[19] de servidor SSH como primera medida, y realizar el resto de pasos desde un cliente SSH en el SO anfitrión. Por tanto una vez hecho ésto con las indicaciones dadas al final del apartado 1.3, ya podemos proseguir con la configuración permanente de red y el resto del software:

La configuración de red sigue los detalles que hemos introducido al configurar Qemu:

[19] http://www.openssh.org



...la primera interfaz debe ser configurada manualmente con una IP en el segmento de red 192.168.1.0/24, concretamente 192.168.1.1 para dklab1 y 192.168.1.2 para dklab2. La segunda interfaz (user-mode) debe utilizar el protocolo DHCP para conseguir su IP 10.0.2.15, que es adicionalmente la que usamos en la redirección de puertos. El apartado 1.4.1 da buena cuenta de cómo acepta un sistema Debian estas modificaciones de una forma permanente.

Respecto al software base, ya hemos insistido durante este documento que el despliegue de los servicios consiste básicamente en introducir comandos en una Shell (en alguna ocasión en varias en paralelo), y editar ficheros de configuración. Es por ello que GNU Screen (multiplexor de terminales), GNU Bash (shell "Bourne Again") o Vim (editor tipo vi/ed "Vi Improved") son las utilidades fundamentales, pero también lo son otras que nos ayudan a depurar problemas (iputils-ping, nmap, socat, tshark, strace, man) y que se explican en el apartado 1.4.4. Además y, como preámbulo, ese apartado instruye sobre algunas modificaciones al sistema de soporte de localización, al sistema Debconf (el sistema, precisamente, de configuración de paquetes en Debian) para que todo funcione idealmente durante las instalaciones.

Los siguientes apartados suponen que se ha seguido paso a paso el Anexo 0, y que se dispone de dos máquinas virtuales interconectadas plenamente operativas.

1.3.5. OpenVPN y CA

Anexo:	1

En el apartado dedicado al testbed, hemos conseguido que ambas máquinas virtuales estén enlazadas y puedan encontrarse en un segmento de red IP. Ésto es más que suficiente tanto para dotar a los servicios de la tecnología IP de la que dependen, como para representar las condiciones que a nivel de red se presentarían en un entorno en producción. Es

por todo ello que la presencia de una red privada virtual (VPN) no puede venir impuesta por esas necesidades inmediatas pues ya han sido resueltas.

Efectivamente, nuestra VPN trata de dar solución a un problema no inmediato sino de fondo que presenta por sí solo nuestro proyecto: si vamos a desplegar y depender de una compleja infraestructura de servicios, debiéramos asegurarnos de que esté siempre disponible, sea cual sea nuestra ubicación excepcional.

No tenemos la intención de quedarnos con ninguna situación en concreto. En un extremo, nuestro parque informático podría disponer de servidores plenamente accesibles desde otras redes y una VPN no aportaría nada en términos de accesibilidad. En otro extremo, nuestro parque informático estaría completamente desconectado del exterior de forma que los servicios sólo serían accesibles desde dentro y una VPN no podría tampoco ayudar a hacerlos disponibles. El resto son situaciones intermedias en las que algunos nodos del parque (y que no sirven servicios al exterior) son accesibles total o parcialmente (firewall) desde otras redes.

Estos nodos entonces podrían ejecutar un software que les permitiese ser un extremo del túnel de la red privada. Su contraparte, sería aquel ordenador (por ejemplo, un portátil) que, encontrándose en una ubicación excepcional y alejada de nuestro parque informático, ejecutaría el software que le permitiese ser el otro extremo del túnel. Cuestiones como si el segmento de red virtual debiera ser distinto o no del habitual dependerían de cada situación específica.

La discusión anterior debemos llevarla ahora a nuestro testbed. Queremos mostrar en él un despliegue de VPN, pero éste sólo dispone de dos máquinas. Una forma de abordarlo sería la siguiente:

- Dklab1 será un extremo y dklab2 el otro. Ésto será válido en nuestro reducido testbed pero advertimos que no es correcto en un despliegue en producción. Efectivamente, no tiene sentido replicar los servicios en dklab1 y dklab2, a la vez que sus clientes se hacen depender de dklab1 para acceder a la red: si dklab1 cae, no podrán beneciarse de la replicación en dklab2.
- La red privada virtual estará en otro segmento de red: 10.168.1.0/24.
- Para facilitar el despliegue, los servicios se ofrecerán sobre la VPN.

Tecnología elegida: OpenVPN

Tras la anterior discusión preliminar, debemos elegir el tipo de tecnología de VPN. La idea tras las redes privadas virtuales es el transporte de la información a partir del nivel de red como payload en otros niveles. Existen diferentes tecnologías, cada una montando la red en uno u otro nivel y ofreciendo servicios adicionales. Nuestra elección se basa en los siguientes requisitos:

- Debe ser suficientemente versátil como para resolver nuestro problema de accesibilidad a nuestra infraestructura. Específicamente, debe poder operar incluso desde redes privadas que utilizan NAT para alcanzar el resto de redes.

- Puesto que la información será transportada en redes y equipos independientes (y, por tanto, considerables inseguros), no debe ser sólo un protocolo de encapsulación. Al contrario, debe proveer también de servicios de seguridad (autenticación, autorización, confidencialidad...) robustos e independientes de nuestro posterior despliegue KERBEROS.

- Debe ser eficiente.

- Debe tener soporte para que participen muchos clientes sin que ello suponga una gran carga de trabajo de mantenimiento para el administrador.

- Debe ser una solución probada, multiplataforma y sencilla de instalar en el cliente.

Soluciones conocidas son IPsec o la combinación L2TP[20] e IPsec[21]. L2TP surge para encapsular sobre UDP tráfico PPP (nivel de enlace) entre un par LNS y LAC; si bien no tenemos necesidad de usar PPP en nuestro proyecto, es una solución bastante interoperable. L2TP no provee servicios de seguridad y por ello no se implementa aislado: al contrario, se suele usar conjuntamente con el modo Transporte de IPsec. IPsec por su lado es un robusto estándar industrial implementado en el kernel del SO. IPsec describe un protocolo, ESP[22], que operando ahora en un modo distinto, modo Túnel, permite la creación de túneles confidenciales. Conjuntamente a ESP, IPsec define un protocolo auxiliar, IKE/ISAKMP[23] (Internet Key Exchange, del que el protocolo ISAKMP suele ser parte) el cual, sobre una conexión en puerto UDP 500, se encarga de negociar y renegociar los parámetros criptográficos de las transformaciones que lleva a cabo IPsec en los paquetes IP, así como ofrecer otra serie de subservicios que aseguren que ESP pueda operar exitosamente. Respecto a éstos, por ejemplo permite discernir si el iniciador, el respondedor o ambos se sitúan tras pasarelas NAT, caso en el que ESP irá encapsulado sobre UDP en puerto 4500; otro servicio es la derivación de políticas de asociación IPsec del iniciador al respondedor; o también servicios de autenticación y configuración de parámetros de red (bien de forma integrada con sistemas NAS - Network Access Server, usando protocolos AAA como RADIUS -, bien de forma aislada usando PKIX, un pool de gestión local de IP ...), etc.

Tras probar las dos posibilidades anteriores y a pesar de lo mucho que ha contribuído IKEv2 para hacer IPsec adaptable a una amplia variedad de escenarios, hemos encontrado aún más idónea la utilización de una tercera opción, el software OpenVPN[24]. Éste, implementado como una aplicación, tuneliza una red sobre UDP o TCP ofreciendo todos los servicios que mencionábamos para IKE y satisfaciendo todos los requerimientos que mencionamos para nuestra VPN. Efectivamente es compatible con NAT, puede utilizar

[20] http://tools.ietf.org/html/rfc2661
[21] http://datatracker.ietf.org/wg/ipsec/charter/
[22] http://tools.ietf.org/html/rfc4303
http://www.unixwiz.net/techtips/iguide-ipsec.html#esp
[23] http://tools.ietf.org/html/rfc4306
http://book.soundonair.ru/cisco/ch13lev1sec4.html
[24] http://openvpn.net/

(entre otros) un esquema PKIX[25] para ofrecer servicios de seguridad, tiene un modo multicliente fácil de configurar, es multiplataforma, sencillo y no está considerado ineficiente[26]. Es la opción más flexible y adaptable y por éso la hemos considerado más idónea que IPsec en un escenario abstracto, ilustrativo como es el nuestro; en cualquier otro concreto deberíamos evaluar la compatibilidad con IPsec y decantarnos por éste si no preveemos problema alguno.

El apartado 1.1.1 del correspondiente anexo detalla el proceso de instalación de OpenVPN. Ésto no tiene resultado alguno más allá de tener disponible el ejecutable "openvpn". Para crear la VPN se requiere tomar una serie de decisiones y escribir un fichero de configuración que declare el comportamiento final del servicio. A este respecto, utilizaremos:

- Modo mayor de operación "server" en dklab1, "client" en dklab2.
- Protocolo "tcp-server" en dklab1, "tcp-client" en dklab2.
- Topología "subnet".
- Interfaz TAP, por tanto el túnel lleva tráfico ethernet, nivel 2 en adelante.
- Submodo "client-to-client".
- Sistema CCD (Client Config Directory) para configurar cada cliente.
- Servicios de seguridad PKIX: "tls-server" en dklab1, "tls-client" en dklab2.
- Cuenta local no privilegiada con que se ejecutará openvpn, y que tendrá "openvpn" por id.
- Integración en la configuración de red de Debian tal que la VPN se establezca automáticamente al arranque.

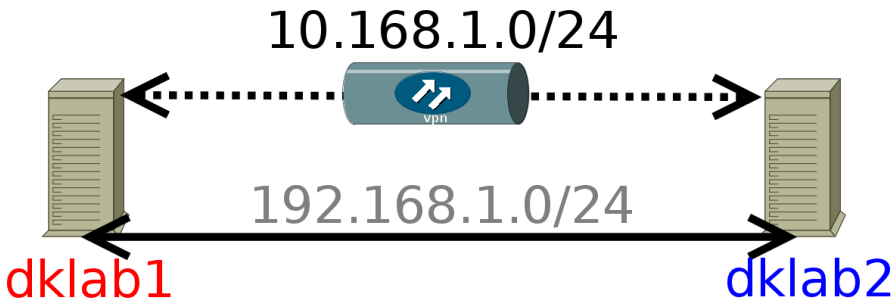
El modo mayor "server" permite que una única instancia de openvpn en dklab1 pueda gestionar un túnel con cada cliente que se conecte. Ésto no era así en versiones antiguas, y el coste administrativo era mayor porque requería configurar una instancia por cada posible cliente. Por otro lado, podemos considerar relativamente fiables las redes de hoy en día y utilizar TCP para el transporte, en otro caso OpenVPN nos ofrece también a UDP como alternativa.

La topología "subnet" implica que dklab1 se configura a nivel 3 como una de las máquinas de una subred (en lugar de, por ejemplo, ser un extremo punto a punto) y en la que representa la puerta de enlace predeterminada a otras redes; como declaramos en el apartado anterior, el segmento de red elegido entonces y que hemos de declarar ahora

[25] http://datatracker.ietf.org/wg/pkix/charter/

[26] Como sí lo es, por cierto, el soporte de VPN de OpenSSH. Por esa razón no lo contemplamos tampoco como solución VPN a pesar de tenerlo ya desplegado, pero sí merece la pena comentar que existe esa funcionalidad, completando el heroico conjunto de posibilidades que hacen de SSH un protocolo tan sumamente versátil y útil.
http://www.vicente-navarro.com/blog/2009/05/24/
creando-tuneles-tcpip-port-forwarding-con-ssh-los-8-escenarios-posibles-usando-openssh/

fue 10.168.1.0/24 y la dirección que corresponderá a dklab1 será la 10.168.1.1. Es digno de mención, si bien nosotros no lo usaremos, que OpenVPN puede crear un conmutador a nivel 2 por software (un "bridge" en la terminología usada por OpenVPN) útil en escenarios donde preferimos no segmentar la red preexistente.



Respecto al submodo "client-to-client", permite que los clientes que se conecten a dklab1 se puedan ver entre sí. Por otro lado el sistema CCD nos permite opcionalmente tener una configuración por cada cliente (por ejemplo la IP que recibirá) una vez que éste ha conseguido autenticar su Common Name.

Finalmente, OpenVPN soporta un sistema robusto de seguridad: utiliza a TLS (Transport Layer Security) y PKC X.509 (certificados de llave pública X.509) para implementar un sistema PKIX (es decir, el sistema "Public Key Infrastructure" del estándar X.509 pero adaptado a Internet [27]). PKIX ofrece servicios de autenticación, autorización (PMI), confidencialidad e integridad[28] a través de mecanismos como criptografía asimétrica ("de llave pública"), firma digital, notaría (y no ACL que son, al contrario, innecesarias si se usa PMI). Efectivamente, lo podemos desarrollar con independencia de nuestro servicio de autenticación basado en KERBEROS.

CA

Como se ha comentado, PKIX utiliza certificados de llave pública X.509. Uno de sus atributos es el "Subject Public Key Info", y su valor es efectivamente una llave pública. Otro de los atributos es el "Subject", cuyo valor sigue una sintaxis X.500 de DN (Nombre Distinguido), luego está dividido en varias partes que representan una ruta en un directorio X.500. En cualquier caso una de esas partes (técnicamente un RDN, de "Relative DN") es llamado CN (de "Common Name") y representa la identidad del propietario de la llave pública mencionada, es decir la llave anunciada en "Subject Public Key Info". También,

[27] X.509 fue originariamente desarrollado para el sistema de directorio de sistemas abiertos OSI X.500. Incluye dos sistemas independientes: PKI permite relacionar una llave pública con un sujeto y se usa por tanto en mecanismos de autenticación. PMI permite relacionar una lista de autorizaciones con un titular y se usa por tanto en mecanismos de autorización. Llaves públicas y atributos se almacenan en estructuras de datos llamados certificados X.509 (certificado de clave pública en PKI, certificado de autorización en PMI). Algunos de sus atributos fundamentales (Subject, Issuer) utilizan una sintaxis de Nombres Distinguidos X.500. En cualquier caso, la estructura de datos está definida usando a ASN.1 y codificada a través de DER, PEM u otros.

[28] http://technet.microsoft.com/en-us/library/cc700808.aspx

claro, CN es la identidad del par criptográfico completo (llave pública y llave privada que, por su rol en un sistema criptográfico asimétrico, permanece al margen, no en el certificado, tal que el sistema pueda ofrecer servicios de confidencialidad, no repudio etc).

Para poder asegurar la relación entre el par criptográfico y el Common Name, es necesario una tercera entidad de confianza a la que llamamos CA, de "Certificate Authority" y que utiliza mecanismos de firma digital para avalar esa relación. La identidad de esa CA viene definida en otro atributo, "Issuer", con un contenido parecido al del atributo Subject, es decir con una parte (RDN) llamada CN cuyo valor es la identidad de dicha CA.

Llevado el esquema anterior a nuestra VPN, necesitamos crear una CA que firme un certificado a dklab1 y otro a dklab2, además necesitamos distribuir el certificado de la CA en dklab1 y dklab2. Con ello, por ejemplo dklab2 se identificará en dklab1 presentándole un certificado cuyo atributo Subject, en concreto en la parte Common Name, contiene su nombre. Y en segundo lugar, el atributo Issuer, en concreto el Common Name, contiene el nombre de la CA, y por tanto permite a openvpn en dklab1 buscar el certificado de la CA y comprobar si ésta es la responsable de la firma del certificado presentado desde dklab2. Si todo es correcto, la identidad de dklab2 se considera comprobada y queda autenticado (todo ello sin menoscabo de otras acciones, como la comprobación de que se es poseedor de la llave privada asociada a la llave pública del certificado, etc).

Volviendo al estándar X.509, éste propone adicionalmente al esquema anterior que posibilita la autenticación, otro para gestionar autorizaciones (PMI, Privilege Management Infraestructure). Se utilizan certificados de atributos AC ("Attribute Certificates", en lugar de los anteriores PKC, "Public Key Certificates") los cuales contienen información de autorización (en lugar de llaves públicas), y donde el atributo Issuer no designa la identidad de una autoridad de certificación (CA) sino de una autoridad de autorización (AA). La AA certifica que la entidad expresada en el atributo "Holder" (esta vez no es "Subject" pero en ocasiones puede serlo también) le corresponden las autorizaciones expresadas en los atributos del certificado. Puesto que esa correspondencia se implementa a través de una firma digital, PMI puede depender de PKI en el sentido de que se utilice un PKC asociado a la AA para verificar su firma, pero PMI y PKI son independientes en cuanto a que la AA y la CA pueden ser entidades distintas, así como que los AC y los PKC son certificados distintos, y esa distinción es pertinente puesto que la gestión de las autorizaciones suele ser más volátil que la de identidad, entre otras diferencias.

Sea como fuere el estándar, diremos que, en el caso que nos ocupa, OpenVPN utiliza información de autorización en sus certificados, pero no considera necesario crear certificados y autoridades separadas y, efectivamente, los scripts que el proyecto provee simplemente introducen alguna extensión en un PKC firmado por una CA. En general, respecto a las extensiones interpretadas como de autorización de alguna clase, podríamos destacar:

- "Netscape Cert Type: SSL Server". Se añade para autorizar al Subject a operar en modo "tls-server" de openvpn, evitando así ciertos ataques MitM (man-in-the-middle, de intermediario)[29].

[29] http://openvpn.net/index.php/open-source/documentation/howto.html#mitm

- "X509v3 Subject Alternative Name", permite añadir o reemplazar identidades en el atributo Subject. Efectivamente el "Common Name" expresaba una identidad, pero adicionalmente puede agrupar, separando con "/", la misma identidad pero desde diferentes puntos de vista: nombre en un sistema directorio, dirección en un sistema de correo, nombre en un sistema de nombres de dominio... Por ejemplo: "...CN=<nombredirectorio>/emailAddress=<email>". Pues bien, gracias a esta extensión se pueden añadir formas de expresar la identidad que no están presentes en CN del Subject.
- "X509v3 Basic Constraints", identifica si el Subject es también una CA y puede por tanto firmar otros certificados y con qué nivel de profundidad.
- X509v3 Key Usage, indica los usos permitidos de la llave.

Estamos ya en condiciones de definir la estrategia que seguiremos para crear nuestra CA y los demás certificados:

- Utilizaremos un conjunto de scripts, Easy-RSA v2, proveídos por OpenVPN.
- La llave privada del par de la CA firma todos los certicados.
- El uso de las extensiones producirá 3 tipos de certificados: para la CA, para el openvpn en modo tls-server, para el openvpn en modo tls-client.
- Se añadirá el nombre de dominio como Subject alternativo.

Efectivamente, OpenVPN viene empaquetado con Easy-RSA v2, un conjunto de scripts, ficheros y árbol de directorios para crear, mantener y utilizar una CA. Sus scripts de shell constituyen una interfaz de alto nivel a la utilidad openssl, la cual es a su vez la interfaz en línea de comando a las librerías criptográficas del proyecto OpenSSL[30]. La utilidad openssl recoge su configuración de un fichero openssl.cnf, que debemos entender y adaptar a nuestras necesidades. Resaltando lo fundamental, diremos que este ".cnf" declara, en una sintaxis tipo INI, tres secciones, y que cada una es interpretada por un script de Easy-RSA v2: el script "build-ca" interpreta la sección "[CA_default]", el script "build-key-server" interpreta la sección "[server]", el script "build-key" (o "build-key-pkcs12" si queremos el formato de salida pkcs12[31]) interpreta la sección "[usr_cert]". Puesto que cada sección configura qué extensiones tendrá cada uno de los 3 tipos de certificados que, mencionamos, produce Easy-RSA v2, el resultado final es que lanzando uno u otro script, conseguimos un comportamiento u otro de openssl y por tanto un tipo u otro de certificado. Así pues, la extensión "Netscape Cert Type" tendrá asignado el valor "SSL Server" para el certificado del "tls-server" (dklab1). "X509v3 Key Usage" tendrá asignado "Digital Signature" para todos y "Key Encypherment" para el "tls-server" (dklab1). "X509v3 Basic Constraints" será FALSE excepto para el certificado de la CA.

Así es, la llave privada de la CA firma todos los certificados, incluyendo el suyo y por tanto su certificado es autofirmado y el RDN "CN" del atributo "Subject" y del atributo

[30] http://www.openssl.org
[31] http://www.rsa.com/rsalabs/node.asp?id=2138

"Issuer" son el mismo. Por otro lado este certificado ha de distribuirse en dklab1 y dklab2 de tal forma que lo puedan utilizar para verificar las firmas.

Otro elemento importante de Easy-RSA v2 es un fichero "vars" donde se definen variables de entorno y que, referenciadas en el ".cnf", permite que sean utilizadas al vuelo por openssl. Nosotros las usaremos o no para que el nombre de dominio aparezca como nombre alternativo (es decir, en la extensión "Subject Alternative Name"), o decidir el correo electrónico que aparecerá en el "CN".

A propósito de éste, el "CN" del "Subject" será tanto un correo electrónico como un nombre pasado como argumento al script y que, en el caso de los certificados para dklab1 y dklab2, se hará coincidir con el nombre de dominio también[32].

La sección 1.1.2 del anexo expone todos los detalles sobre la creación y uso de nuestra CA, incluyendo la creación y distribución de una lista de revocación, algo imprescindible dado que OpenVPN no implementa aún el protocolo OSCP. También muestra cómo verificar el material criptográfico creado.

En las secciones 1.1.5 y 1.2.4 se detalla la configuración para todas las decisiones que tomamos y que definen el comportamiento de cada openvpn.

1.3.6. DNS

Anexo:	2

Empezamos pues con la Infraestructura de Servicios Base.

Nuestra infraestructura tiene necesidades de almacenar y consultar información relativa a los nodos de la red:

- En general, el parque informático va a ser usado por humanos, pero los identificadores binarios de IP no son apropiados para ellos.
- En un sistema distribuído es necesario un sistema especial para conocer cuántos y dónde se sitúan los servicios.
- Los esquemas de seguridad usados necesitan comprobar coincidencias con el nombre de dominio, como parte de sus procesos de verificación de credenciales.

El sistema DNS resuelve todas nuestras necesidades y, aunque aún hoy en día tiene un diseño inseguro, es la tecnología recomendada.

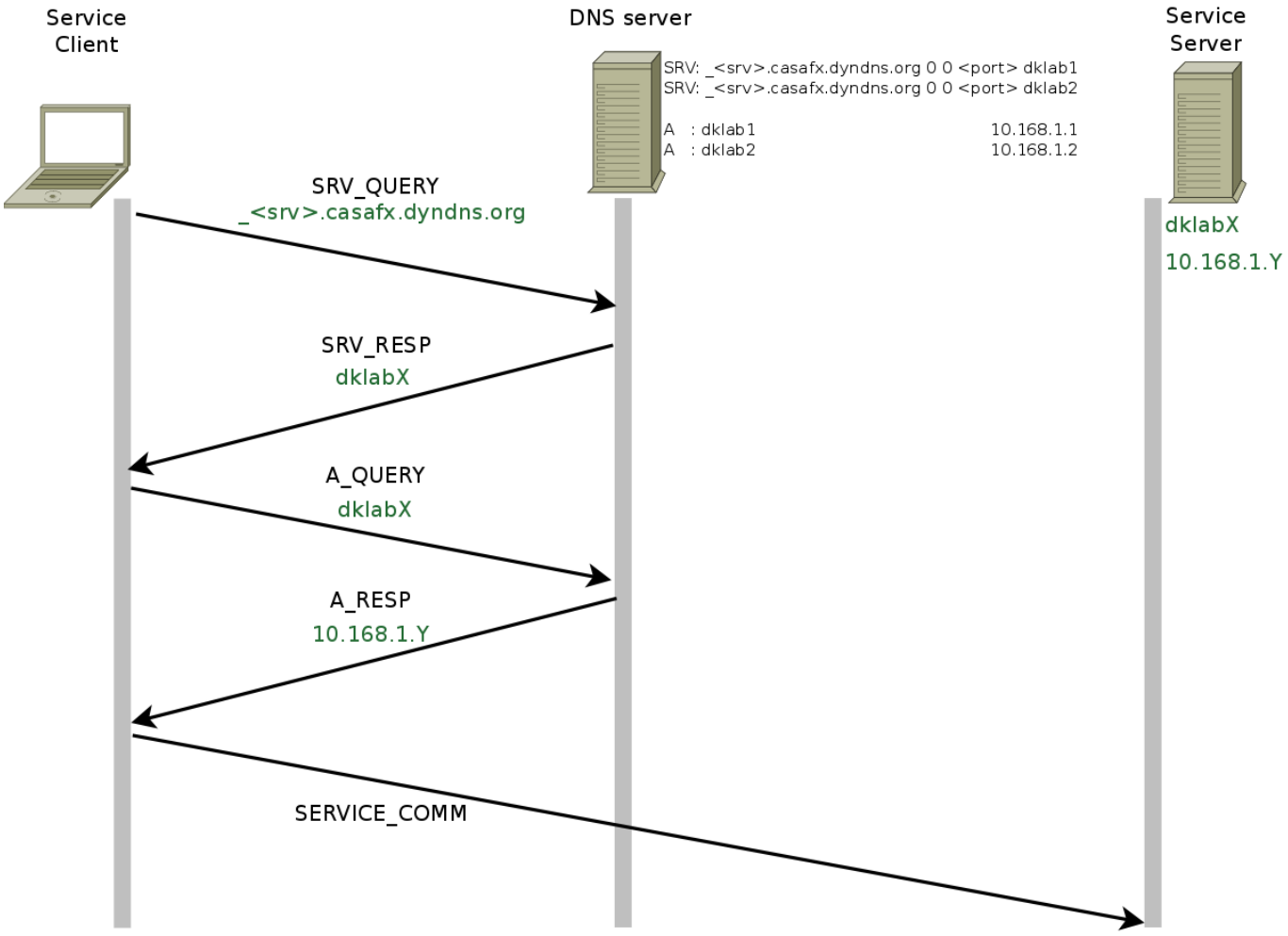
Efectivamente, cuando un conjunto de computadoras se relacionan intensamente demandando y ofreciendo servicios, se crea una supraestructura (el dominio, reino o célula según diferentes terminologías) que adquiere entidad propia pero que se apoya en el trabajo subyacente de cada nodo. DNS, a través de sus diferentes RR ("Resource Registry"), permite almacenar las distintas correspondencias que permiten encapsular y descubrir los

[32]En futuros usos de nuestra CA donde sea necesaria cierta multiplicidad para un mismo dominio, el CN será otro nombre convenientemente elegido y el nombre de dominio sólo irá en el "Subject Alternative Name".

nodos y sus servicios a través de nombres simples. Por un lado, utilizando registros SRV[33] (también, con resultados parciales, otros) y dado el nombre de un dominio de ordenadores, DNS nos permite corresponder qué servicios están disponibles, el nombre de los nodos que los ofrecen, el punto de acceso al servicio de transporte o con qué prioridad se deben usar. Por otro, utilizando registros A y dado el nombre de un nodo (que, como acabamos de mencionar, ha podido ser devuelto de una consulta SRV), DNS nos permite hacer corresponder finalmente cuál es su IP, y a la inversa.

El siguiente diagrama pretende ilustrar la anterior dinámica de descubrimiento de servicios a través del nombre de dominio, los registros SRV y los registros A. El diagrama presenta esquemáticamente cómo el cliente realiza una consulta conociendo sólo el tipo de servicio <srv>y el nombre del dominio "casafx.dyndns.org". El servidor DNS tiene los registros SRV para ese servicio, y los registros A con las direcciones IP de los servidores que ofrecen ese servicio. Así pues, el servidor DNS responde la consulta con el nombre de alguno de los servidores que ofrecen el servicio, "dklabX". El cliente recibe la respuesta e inmediatamente consulta por la dirección IP asociada a "dklabX". El servidor DNS recibe la consulta y responde con la IP "10.168.1.Y". Ésto permite al cliente, por fin, abrir una conexión con el servidor que ofrece el servicio. Queremos anotar también que, a partir de ahora, cuando en los diagramas aparezca "dklabX" bajo la figura de algún servidor, significa que éste se descubre a través de registros SRV en un proceso como el explicado.

[33] http://tools.ietf.org/html/rfc2782
http://www.dns-sd.org/ServiceTypes.html



Continuando con la necesidad de disponer de un servicio de nombres DNS, diremos que además de la flexibilidad y eficacia de usar un sistema así, nos es necesario porque es requisito para el correcto funcionamiento de los esquemas se seguridad que usaremos, KERBEROS y TLS.

Ya se comentará en su respectiva sección los rudimentos de KERBEROS, pero por ahora diremos que la identidad en un sistema de este tipo es llamada "principal" y tiene la forma <primario>/<instancia>@<REALM>, donde <instancia>se suele corresponder en la versión 5 del protocolo con el FQDN ("Fully Qualified Domain Name"), el nombre del nodo sufijado por el nombre de su dominio. El sistema verifica[5] la <instancia>a través de la llamada "gethostbyaddr(gethostbyname(<nombre host>))" o su equivalente en API's no POSIX. Efectivamente, esa verificación resultará exitosa sólo si el sistema DNS está bien configurado y devuelve <instancia>. Otro problema es el de si cada host conoce cuál es su FQDN, ya que aunque algunos servidores admiten que se les diga explícitamente cuál es su principal, otros no y lo averiguan por sí mismos a través de la llamada "getaddrinfo(gethostname())" de forma que si el resultado es incorrecto no encontrarán su principal y no podrán ofrecer autenticación basada en KERBEROS. Remitimos a la sección 1.8 de otro anexo, el 7, que presenta los pormenores de ambas resoluciones (la primera dependiente de DNS, la segunda no pero relacionada). Una última cuestión en relación a KERBEROS y

DNS son los escenarios "multi-home", es decir cuando las máquinas tienen varias interfaces de red con distintas IP. Puede consultarse una discusión al respecto en el apartado 2.14 de la FAQ de KERBEROS[34].

El caso de TLS es parecido, pues los clientes tratarán de comprobar la validez del certificado presentado por el servidor. Ésto implica una serie de acciones que contemplan la verificación de la firma a través del certificado del Issuer (CA), pero también otras que inciden en DNS. Así, se comprueba que el nombre utilizado para acceder al servicio[35] coincide con el declarado en el certificado. El certificado puede explicitar ésto en el CN del Subject, y en este caso es válido usar wildcards (en el sentido de las expresiones regulares "glob pattern" de una shell POSIX), por ejemplo "<subdominio>.dominio.tld" casaría si en el CN apareciese "*.dominio.tld". Alternativamente, puede aparecer en el "Subject Alternative Name" y, de hecho, los clientes obbiarán el CN cuando aparece un "Subject Alternative Name", por lo que en éste se suele dejar copiado el CN cuando éste incluye información DNS.

Implementación escogida: Bind9

Bind9[36] es la implementación del ISC ("Internet Systems Consortium") y es aquella sobre la que descansa el sistema DNS actual (más de un 70 % de los despliegues [37]). Es por tanto la implementación de referencia y ofrece todas las funcionalidades a las que haremos referencia.

Aunque el proyecto se llama Bind, Bind9 representa el nombre del proyecto para la versión 9. Por su lado, "named" es el nombre del binario que implementa las funcionalidades.

Vistas

Cuando se diseñó la VPN, se eligió ofrecer los servicios sobre ésta pues era el escenario más versátil. Efectivamente, en cuanto a reproducibilidad de nuestro despliegue el primer lugar lo ocupa utilizar nuestro testbed, pero gracias a la decisión de ofrecer los servicios sobre la VPN una alternativa factible todavía con ciertas garantías sería montar una VPN idéntica a la nuestra e intentar seguir desde ahí.

De cualquier forma, lo cierto es que en nuestro testbed dklab1 y dklab2 son máquinas "multi-homed", puesto que están conectadas entre sí a través de más de un par de direcciones de red.

Según nuestro conocimiento, OpenAFS es el único de los servicios que desplegaremos que implementa una solución para desenvolverse controladamente en sistemas "multi-homed". Para ello utiliza varios mecanismos, incluído, por ejemplo, el registro de las

[34] http://www.faqs.org/faqs/kerberos-faq/general/section-47.html

[35] En nuestro despliegue el nombre con el que se accede al servicio es el del dominio (a partir del cual actúan los registros SRV y luego los A), en otros despliegues más reducidos el nombre con el que se accede al servicio es directamente el nombre de la máquina que lo aloja.

[36] http://www.isc.org/software/bind/documentation

[37] http://www.caida.org/publications/presentations/2007/dns_survey/dns_survey.pdf

interfaces disponibles en cada host, listas blancas y listas negras de IP..., véase sección 1.2.7 del anexo 6.

A pesar de ello, DNS provee una forma de poder resolver de forma independiente los nombres de las máquinas según criterios como la IP de origen de la consulta. Esta funcionalidad es llamada "Vistas" y vemos en ella una capacidad de control añadida que se puede operativizar enseguida para ayudar en escenarios "multi-home"[38]. Así, aunque nosotros vamos directamente a restringir nuestros servicios a la VPN, sí mostraremos al menos cómo se configuran las Vistas DNS.

Tendremos dos tipos de vistas por IP de origen de la consulta, en el orden en que se declararán en el fichero de configuración son:

Nombre de la vista	Rango de IP's de origen a la que se aplica
vpn	10.168.1.0/24
external	*

Las vistas se aplican en orden, es por ello que el filtro para la vista "external" se puede entender en principio como que "external" es la vista aplicada por defecto, cuando no hubo coincidencia en el filtro de la vista precedente.

Política de nombres y RR

A continuación detallamos los registros usados en la vista vpn. En primer lugar todos aquellos para la zona directa y por último los de la zona inversa, es decir para resoluciones de dirección IP a nombre de host.

Como norma general, los servicios se descubren a través de SRV, para cada servicio existirá un servidor en dklab1 y otro en dklab2, ambos con la misma prioridad y peso. El nombre del dominio (que usarán, por ejemplo, los registros SRV) es casafx.dyndns.org

Dominio	**casafx.dyndns.org.**

DNS

Tipo (RR)	Nombre (FQDN)	Datos específicos del tipo
SOA	casafx.dyndns.org.	...
NS	casafx.dyndns.org.	ns1.casafx.dyndns.org.
NS	casafx.dyndns.org.	ns2.casafx.dyndns.org.
HINFO	casafx.dyndns.org.	"PC" "Debian 6"

[38]Por ejemplo, el sistema de vistas podría ayudar a resolver algunos de los problemas de KERBEROS relacionados con escenarios "multi-home". O podría ayudar a independizar el sistema de nombres para la VPN y el de la red física en que se apoya.

Nos declaramos autoridad de la zona casafx.dyndns.org a través del registro SOA ("Start Of Authority"). Explicitamos a ns1.casafx.dyndns.org. y ns2.casafx.dyndns.org. como servidores DNS de esa zona; más tarde mencionaremos el registro A que traduce ambos nombres a su IP. También añadimos un registro HINFO que caracterice a este servidor DNS.

NTP

Tipo (RR)	Nombre (FQDN)	Datos específicos del tipo
A	ntp.casafx.dyndns.org.	10.168.1.1
A	ntp.casafx.dyndns.org.	10.168.1.2
SRV	_ntp._udp.casafx.dyndns.org.	0 0 123 dklab1.casafx.dyndns.org.
SRV	_ntp._udp.casafx.dyndns.org.	0 0 123 dklab2.casafx.dyndns.org.

La implementación Ntp.org que usaremos no tiene soporte para descubrimiento de servicios a través de registros SRV. Sin embargo éstos últimos sí están definidos desde el año 2000 en el rfc2782[39] y clientes basados en otras implementaciones (como la de Windows) sí parecen usarlo y por ello se indican. Como decimos, lo cierto es que nuestra solución en el testbed estará basada en el pool de registros A que les antecede.

LDAP

Tipo (RR)	Nombre (FQDN)	Datos específicos del tipo
SRV	_ldap._tcp.casafx.dyndns.org.	0 0 389 dklab1.casafx.dyndns.org.
SRV	_ldap._tcp.casafx.dyndns.org.	0 0 389 dklab2.casafx.dyndns.org.

Los servicios de directorio se pueden descubrir a través de sendos registros SRV. Como ocurrirá para el resto de ocasiones, el campo de prioridad y peso de los registros SRV tienen por valor cero. Ésto implica que a ambos se les concede la misma prioridad, y que los servidores devolverán a dklab1 y dklab2 en rigurosa alternancia de forma que el efecto de distribución de carga que tienen los registros SRV sea en nuestro despliegue equitativa.

KERBEROS

Tipo (RR)	Nombre (FQDN)	Datos específicos del tipo
TXT	_kerberos.casafx.dyndns.org.	"CASAFX.DYNDNS.ORG"
SRV	_kerberos-adm._tcp.casafx.dyndns.org.	0 0 749 krb1.casafx.dyndns.org.
SRV	_kerberos-adm._tcp.casafx.dyndns.org.	0 0 749 krb2.casafx.dyndns.org.
SRV	_kpasswd._udp.casafx.dyndns.org.	0 0 464 dklab1.casafx.dyndns.org.
SRV	_kpasswd._udp.casafx.dyndns.org.	0 0 464 dklab2.casafx.dyndns.org.
SRV	_kerberos-master._udp.casafx.dyndns.org.	0 0 88 dklab1.casafx.dyndns.org.
SRV	_kerberos-master._udp.casafx.dyndns.org.	0 0 88 dklab2.casafx.dyndns.org.
SRV	_kerberos._ucp.casafx.dyndns.org.	0 0 88 dklab1.casafx.dyndns.org.
SRV	_kerberos._ucp.casafx.dyndns.org.	0 0 88 dklab2.casafx.dyndns.org.

[39] http://tools.ietf.org/html/rfc2782

El caso de KERBEROS es algo más complejo. Por un lado debemos declarar el nombre del reino de autenticación KERBEROS que usarán las máquinas del dominio a través del registro TXT. Por otro lado KERBEROS se componen de varios subservicios y cada uno se descubre de manera independiente. Así los registros prefijados con _kerberos-adm se refieren al gestor de base de datos de KERBEROS[40]. Por su lado _kpasswd se refiere al servicio especial que expide tickets de servicio para cambiar la contraseña de un principal; en MIT Kerberos está implementado en el propio gestor de la base de datos que acabamos de mencionar. En el caso de _kerberos y _kerberos-master se refieren al AS/TGS ("Authentication Server"/"Ticket Granting Server"), pero el segundo caso se usa específcamente ante el primer error de autenticación ante el AS. Ésto permite especificar AS que pudiesen conocer antes que otros la contraseña actual; en nuestro caso no existen AS privilegiados de esa forma debido a nuestro esquema de replicación de la base de datos de MIT Kerberos (usará a LDAP como backend de almacenamiento).

AFS

Tipo (RR)	Nombre (FQDN)	Datos específicos del tipo
SRV	_afs3-vlserver._udp.casafx.dyndns.org.	0 0 7003 dklab1.casafx.dyndns.org.
SRV	_afs3-vlserver._udp.casafx.dyndns.org.	0 0 7003 dklab2.casafx.dyndns.org.
SRV	_afs3-prserver._udp.casafx.dyndns.org.	0 0 7002 dklab1.casafx.dyndns.org.
SRV	_afs3-prserver._udp.casafx.dyndns.org.	0 0 7002 dklab2.casafx.dyndns.org.
AFSDB	casafx.dyndns.org.	1 dklab1.casafx.dyndns.org.
AFSDB	casafx.dyndns.org.	1 dklab2.casafx.dyndns.org.

El servicio de sistema de ficheros AFS distribuye partes del árbol de ficheros en distintos "fileserver". Ésto implica que el cliente debe primero contactar con un servicio que mapee cada parte del árbol de ficheros al "fileserver" que la gestiona y de ahí el par de registros prefijados como _afs3-vlserver. Las autenticaciones o la metainformación relativa a las listas de control de acceso sobre el árbol se implementa como otro servicio separado, y de ahí el otro par de resgistros, _afs3-prserver.

Por último, los registros especiales AFSDB son el precedente a los registros SRV (en 1990, cuando éstos últimos no se habían definido aún), de forma que ofrecen una funcionalidad similar: descubrimiento automático del servicio. Por cuestiones históricas, el valor 1 indica que el registro se refiere a componentes de AFS y no a otros posibles sistemas acompañantes[41].

XMPP

[40] Lo cierto es que los clientes de este subservicio, como kadmin, no soportan aún su descubrimiento a través de registros DNS SRV y por ello se declarará también en ficheros de configuración locales.

[41] Según el rfc1183, a componentes del "Open Software Foundation's (OSF) Distributed Computing Environment (DCE) authenticated naming system using HP/Apollo's NCA".
http://tools.ietf.org/html/rfc1183

Tipo (RR)	Nombre (FQDN)	Datos específicos del tipo
SRV	_xmpp-client._tcp.casafx.dyndns.org.	0 0 5222 dklab1.casafx.dyndns.org.
SRV	_xmpp-client._tcp.casafx.dyndns.org.	0 0 5222 dklab2.casafx.dyndns.org.
SRV	_xmpp-server._tcp.casafx.dyndns.org.	0 0 5269 dklab1.casafx.dyndns.org.
SRV	_xmpp-server._tcp.casafx.dyndns.org.	0 0 5269 dklab2.casafx.dyndns.org.

El servicio de mensajería instantánea y presencia XMPP (y su extensión Jingle de señalización multimedia) permite federaciones. Por ello, además de los registros _xmpp-client con que los clientes descubren el servicio, declaramos registros _xmpp-server con que los servidores de otro dominio puedan descubrir a los servidores del nuestro.

Correo

Tipo (RR)	Nombre (FQDN)	Datos específicos del tipo
MX	casafx.dyndns.org.	10 dklab1.casafx.dyndns.org.
MX	casafx.dyndns.org.	10 dklab2.casafx.dyndns.org.
SPF	casafx.dyndns.org.	"v=spf1 mx/24 ?all"
TXT	casafx.dyndns.org.	"v=spf1 +mx/24 ?all"
TXT	exim4dkim._domainkey.casafx.dyndns.org.	"v=DKIM1; s=... ... p=..."
TXT	_domainkey.casafx.dyndns.org.	"t=y; o=~;"
TXT	_adsp._domainkey.casafx.dyndns.org.	"dkim=unknown"
SRV	_imap._tcp.casafx.dyndns.org.	0 0 143 dklab1.casafx.dyndns.org.
SRV	_imap._tcp.casafx.dyndns.org.	0 0 143 dklab2.casafx.dyndns.org.
SRV	_submission._tcp.casafx.dyndns.org.	0 0 587 dklab1.casafx.dyndns.org.
SRV	_submission._tcp.casafx.dyndns.org.	0 0 587 dklab2.casafx.dyndns.org.
SRV	_sieve._tcp.casafx.dyndns.org.	0 0 4190 dklab1.casafx.dyndns.org.
SRV	_sieve._tcp.casafx.dyndns.org.	0 0 4190 dklab2.casafx.dyndns.org.

El servicio de correo es el más complicado de todos puesto que está compuesto de varios servicios desarrollados a lo largo de la historia de Internet (IPv4 se estandarizó en 1981, SMTP en 1982).

Así pues los registros MX permitirán descubrir los intercambiadores SMTP (que podemos llamar MTA, "Mail Transfer Agent"). Ambos, dklab1 y dklab2, tendrán máxima e idéntica prioridad, 0. Al contrario que sucedía con AFS, a pesar de que los registros MX cumplen similar función a los nuevos SRV, todavía no se han definido éstos últimos para SMTP luego sólo registramos MX.

El registro SPF[42] permite al MTA verificar que la IP desde la que se envía el correo está autorizada a hacerlo. Es por tanto una lista blanca de IP en DNS, y "mx/24" implica que toda IP en el rango resultante de aplicar una máscara de 24 bits a la del registro MX está autorizada, si bien "?all" hace que, para dar un margen de tiempo en que probar el sistema, a las IP no autorizadas no se les deniegue el servicio aún. Puesto que no todos los

[42] http://www.openspf.org/

MTA soportan un registro SPF pero sí pueden interpretar un registro TXT, la información anterior se copia en uno TXT.

A continuación aparecen 3 registros TXT relacionados con DKIM. DKIM[43] es otro sistema de verificación, esta vez basado en firmas digitales. Básicamente, nuestro MTA dispondría de una llave privada con que firmar el correo saliente, y DNS publicaría la llave pública con la cual los MTA receptores de nuestro correo saliente utilizarían para verificar la firma y, con ello, el origen del correo. Precisamente el primero de los registros, prefijado con "exim4dkim", expone la llave pública que se ha de usar (variable "p"). Los dos siguientes establecen la política, el primero usa el formato obsoleto OSP, el segundo el nuevo ADSP. Básicamente, "o=~" y "dkim=unknown" expresan que no todo el correo estará firmado, de nuevo para dar un margen de tiempo en que probar el sistema.

Los registros SRV permiten descubrir los servidores IMAP, MANAGESIEVE (es decir, el servicio de administración remota de filtros de correo SIEVE) y LMTP (submission). Éste último corresponde en realidad a un dialecto de SMTP llamado LMTP[44], las diferencias son que el segundo se inicia con un LHLO (en lugar de EHLO) para poner al servidor sobreaviso de que no debe encolar el mensaje sino sólo emitirlo o fallar, de hecho tras el "." del comando DATA se ha de anunciar qué RCPT's han fallado. Puesto que el comportamiento es distinto al de un MTA, el puerto usado no es 25 sino 587.

En el anexo 9, el correspondiente al servicio de correo, se puede encontrar más información sobre las tecnologías esbozadas.

Registros A y CNAME En general los servicios se han definido a través de registros SRV que apuntaban a nombres de host. Ahora debemos crear los registros A que los resuelven a IP. Además, ahora debemos traducir con un registro A el nombre del servidor DNS de la zona que apuntamos en el registro NS.

Tipo (RR)	Nombre (FQDN)	Datos específicos del tipo
A	dklab1.casafx.dyndns.org.	10.168.1.1
A	dklab2.casafx.dyndns.org.	10.168.1.2
A	ns1.casafx.dyndns.org.	10.168.1.1
A	ns2.casafx.dyndns.org.	10.168.1.2

Se han creado registros CNAME como alias a los registros A anteriores. La ventaja de tener una "capa" de registros CNAME es que si cambiamos un servidor de máquina, en lugar de cambiar el nombre del host o cualquier otro cambio en la configuración, obtenemos el mismo resultado modificando a qué registro A apunta el CNAME correspondiente. Éste, que a primera vista puede representar una forma ventajosa de procecer, sin embargo, **no** es eficaz en nuestro despliegue.

La razón es que esos registros CNAME no se pueden usar donde los programas consultan, es decir en los registros SRV. Específicamente, ocurre que normalmente seguiría a esa resolución SRV una autenticación KERBEROS para poder utilizar el servicio, pero ésta

[43] http://www.dkim.org/

[44] http://tools.ietf.org/html/rfc2033

fallaría al no poder verificar el nombre de la máquina en los términos que se comentaron y, en definitiva, se haría imposible usar el servicio.

Así, los siguientes registros CNAME no tienen más relevancia y podríamos eliminarlos.

Tipo (RR)	Nombre (FQDN)	Datos específicos del tipo
CNAME	openvpn1.casafx.dyndns.org.	dklab1.casafx.dyndns.org.
CNAME	ntp1.casafx.dyndns.org.	dklab1.casafx.dyndns.org.
CNAME	ntp2.casafx.dyndns.org.	dklab2.casafx.dyndns.org.
CNAME	ldap1.casafx.dyndns.org.	dklab1.casafx.dyndns.org.
CNAME	ldap2.casafx.dyndns.org.	dklab2.casafx.dyndns.org.
CNAME	krb1.casafx.dyndns.org.	dklab1.casafx.dyndns.org.
CNAME	krb2.casafx.dyndns.org.	dklab2.casafx.dyndns.org.
CNAME	afs1.casafx.dyndns.org.	dklab1.casafx.dyndns.org.
CNAME	afs2.casafx.dyndns.org.	dklab2.casafx.dyndns.org.
CNAME	managesieve1.casafx.dyndns.org.	dklab1.casafx.dyndns.org.
CNAME	managesieve2.casafx.dyndns.org.	dklab2.casafx.dyndns.org.
CNAME	mx1.casafx.dyndns.org.	dklab1.casafx.dyndns.org.
CNAME	mx2.casafx.dyndns.org.	dklab2.casafx.dyndns.org.
CNAME	imap1.casafx.dyndns.org.	dklab1.casafx.dyndns.org.
CNAME	imap2.casafx.dyndns.org.	dklab2.casafx.dyndns.org.

Zona inversa En primer lugar anunciaremos el servidor de la zona para la traducción inversa. A continuación los registros PTR. Haremos que la relación sea inyectiva, y cada IP se relacione con un sólo nombre, por lo que sólo aparecen dos registros PTR, uno por cada máquina.

Tipo (RR)	Nombre (FQDN)	Datos específicos del tipo
NS	1.168.10.IN-ADDR.ARPA.	ns1.casafx.dyndns.org.
NS	1.168.10.IN-ADDR.ARPA.	ns2.casafx.dyndns.org.
PTR	1.1.168.10.IN-ADDR.ARPA.	dklab1.casafx.dyndns.org.
PTR	2.1.168.10.IN-ADDR.ARPA.	dklab2.casafx.dyndns.org.

El apartado 1.2.6 del anexo detalla cómo declarar las anteriores vistas y registros en ficheros de zonas de Bind9, así como la posibilidad de realizar comprobaciones sintácticas. El apartado 1.2.7 modifica ligeramente la configuración por defecto para la zona "localhost" (convenientemente, además, no sujeta a vistas).

Transferencia entre zonas

Puesto que DNS es en esencia un gestor de base de datos, las ideas manifestadas en el apartado "Justificación" de este documento deben ser cotejadas ahora para DNS. Por tanto, diremos que Bind9:

- Permite centralizar
- Replicar, si bien de forma asíncrona ("Zone Transfer")

- Bajo una topología Master-Slave

El último punto no es estrictamente necesario. Desde hace algún tiempo el proyecto Bind-DLZ[45] permite que named utilice a OpenLDAP como backend de almacenamiento (entre otras posibilidades). Los parches DLZ están aplicados en Debian por defecto, y presentan las siguientes características.

- Más sencillo y robusto de usar con LDAP que otras alternativas[46].
- Permite el uso o no de esta tecnología selectivamente por cada zona.
- Acceso flexible al árbol LDAP (el administrador lo declara utilizando LDAP URI's).
- Se puede beneficiar de las tecnologías de replicación de OpenLDAP, permitiendo abandonar el esquema Master-Slave por uno Multi-Master.

Pero también:

- No ofrece soporte para "DNS Dynamic Updates" en nuestra versión de Bind9.
- No soporta IXFR (transferencia incremental de zonas).
- Soporta el mecanismo simple y KERBEROS sólo v4 para autenticarse en OpenLDAP.
- Presenta ventajas concretas de eficiencia con zonas masivas, sobre todo en el momento de iniciar el servidor.
- Aunque incluye un esquema para LDAP, dlz.schema, la documentación es un tanto parca para mostrar su uso.

Los problemas con las actualizaciones dinámicas y la transferencia IXFR no son tales en nuestro caso dado que sería igualmente eficaz utilizar directamente a LDAP[47]. Sin embargo, nos parece complicado extraer una configuración válida a partir del esquema LDAP y los ejemplos de URI's ofrecidos en el sitio web del proyecto. Pensamos además que introduce una interdependencia importante entre los dos sistemas y que, en definitiva, parece más conveniente afrontar esta posibilidad como una línea de trabajo futuro.

Por tanto el despliegue de Bind9 corresponde al clásico esquema Master-Slave. El maestro estará en dklab1, el secundario estará en dklab2 y las zonas sujetas a transferencias serán "casafx.dyndns.org." tanto en la vista "vpn" como en "external".

Desafortunadamente, el apartado 1.3.7 del anexo describe cómo localizamos un bug que impide realizar la primera transferencia entre zonas. Como se detalla allí, named yerra al iterar sobre cada vista. Nuestra solución fue realizar la transferencia manualmente usando SSH.

[45] http://bind-dlz.sourceforge.net/

[46] http://trull.org/~alex/media/docs/altdnsbook.pdf

[47] Ésto no es válido en cualquier proyectohttp://blog.tridgell.net/?cat=5

Seguridad

Bind9 implementa los siguientes tipos de comunicaciones:

- Respuesta a consultas DNS (UDP/53)
- Transferencia entre zonas (TCP/53)
- Actualización dinámica de registros (TCP/53 o UDP/53)
- Control remoto de named, tal como recarga de los ficheros de zona etc (TCP/953)

Todas ellas admiten al menos dos mecanismos de seguridad: ACL por IP y firmas TSIG.

TSIG (de "Transaction Signatures"[48]) es un mecanismo de firma digital basado en criptografía simétrica que permite autenticar las comunicaciones en DNS. Las utilidades del proyecto proveen MAC (de "Message Authentication Code") basado en HASH, concretamente MD5. Ésto es un problema ya que desde 2004 se han probado ataques por colisión en MD5, pero Bind9 no provee alternativa. TSIG no se suele utilizar para la resolución DNS (primera de las comunicaciones mencionadas), debido a la dificultad en la distribución de la llave. En su lugar se desarrolló DNSSEC que aunque resuelve este problema, introduce otros cuando se intenta implementar a gran escala (Internet) y en nuestro caso queda pendiente su estudio como línea futura de trabajo.

Por tanto, a excepción de las consultas DNS, el resto de comunicaciones utilizan ACL por IP y TSIG. Los detalles se dan en el apartado 1.2.3 (creación de la llave), y 1.2.5 y 1.3.5 del anexo. Aunque en nuestro testbed es suficiente con la creación de una sóla llave TSIG para todas las comunicaciones, en un entorno en producción sería más conveniente mantener una llave independiente para cada una. Por otro lado, el apartado 1.3.9 muestra cómo realizar una actualización dinámica de registros y el 1.3.10 cómo utilizar la interfaz de control de named.

Cliente DNS

En SO basados en la librería C de GNU, se admiten varias fuentes para resolver información relativa a nombres, una de ellas sería DNS. Cuando hablemos sobre el servicio de Cuentas Centralizadas de Shell POSIX se detallará algo más ésto, ahora nos interesa decir que el fichero /etc/resolv.conf almacena tradicionalmente la localización de los servidores de nombres, pero que nosotros hemos interpuesto una capa intermedia que lo gestiona a través del software "resolvconf" de Debian[49]. Ésta nos permite hacer dependiente la configuración de los resolvedores DNS de distintas variables, y realizar cambios automáticos cuando éstas tomen otros valores (por ejemplo, al cambir de red).

Ésto debe entenderse como que, si del lado del servidor teníamos las vistas, del lado del cliente tenemos al gestor resolvconf, siendo ambos mecanismos de distinta naturaleza pero que permiten para ganar control sobre la resolución de nombres. El apartado 1.2.2 y 1.3.2 tiene los detalles.

[48] http://tools.ietf.org/html/rfc2845
[49] http://packages.debian.org/squeeze/resolvconf

1.3.7. NTP

Anexo:	3

Nuestra infraestructura necesita una fuente de tiempo que permita a las máquinas mantenerse sincronizadas. Ésto se debe a que todas participan de algún subsistema que precisa indexar en el tiempo algunas de sus variables:

- La replicación entre bases de datos necesita una forma de ordenación absoluta de las versiones de cada objeto presente en el clúster, de forma que puedan ser etiquetados como antiguos o nuevos y actuar en consecuencia.
- Los tickets KERBEROS ofrecen memoria de una autenticación anterior, pero (en general) sólo durante un intervalo de tiempo a partir de la fecha de expedición. Para que el cliente pueda predecir cuándo dejan de ser válidas sus credenciales necesita una forma de estar de acuerdo con los servidores sobre esa fecha e intervalo.

Efectivamente en ambos casos se necesita una fuente de tiempo. Una implementación robusta de esa fuente de tiempo podría incluir:

- Varios servidores que aumentasen la disponibilidad del servicio a la vez que velasen por mantener única la fuente de tiempo de una forma segura.
- Idealmente éstos estarían sincronizados con la fuente usada internacionalmente.
- A modo de alternativa ante problemas con la fuente internacional, podrían conmutar para ser simplemente una fuente local que continuase permitiendo la sincronización a los participantes de nuestra infraestructura.
- La información temporal sería pública pero adicionalmente nuestro parque informático podría acceder a ella de forma autenticada[50].

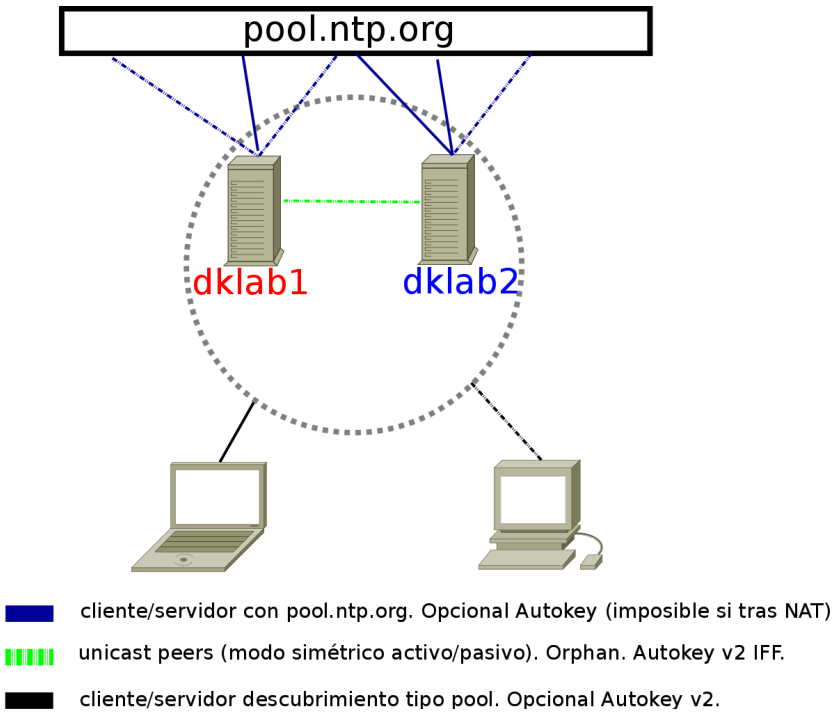
El protocolo NTP[51] y su implementación de Ntp.org[52] permiten desplegar un sistema como el esbozado. Danto respuesta punto por punto a las especificaciones anteriores:

- Para aumentar la disponibilidad y resistencia de una forma segura: Ntp.org permite que varios servidores propios se sincronicen entre sí gracias a su modo unicast peers (simétrico activo/pasivo) en asociación permanente. Además provee mecanismos de seguridad basados en Autokey[53] v2 IFF.
- Para sincronización con fuente internacional: Ntp.org provee un modo cliente/servidor en asociación permanente al clúster mundial pool.ntp.org[54] gestionado por el mismo proyecto.

[50] Nos referimos a que los servidores se autentican ante los clientes para probar su identidad.
[51] En NTP, el tiempo se representa con 64 bits como un número en coma flotante.
http://www.ntp.org/ntpfaq/NTP-s-algo.htm#AEN1895
[52] http://www.ntp.org
[53] http://tools.ietf.org/html/rfc5906
[54] http://www.pool.ntp.org/en/

- Ante problemas con pool.ntp.org: Ntp.org permite conmutación automática a un modo local: Orphan.
- El parque informático hace un descubrimiento tipo pool de los anteriores servidores NTP, Autokey v2 no es obligatorio para autorizarse y poder utilizar el servicio.

El siguiente diagrama muestra los tres tipos de comunicaciones que llevan a cabo los servidores NTP: entre sí, con el pool, con los clientes finales. En nuestro caso, dklab1 y dklab2 serán nuestro pool local de servidores NTP. Los siguientes apartados detallan su funcionamiento al usar los modos esbozados anteriormente.



Stratums y modo Orphan

A un nivel abstracto, NTP organiza las fuentes de hora a nivel planetario de forma jerárquica donde cada nivel de la jerarquía es llamado "stratum n", siendo n un número natural entre 0 y 16.

Stratum 0 es un dispositivo que utilizando técnicas como GPS conoce la hora UTC ("Universal Time Coordinated"). Stratum 1 es una computadora que, sin red intermediaria sino de forma directa (por ejemplo porque el dispositivo stratum 0 esté conectado a su placa base) recibe la información temporal del Stratum 0. Stratum 2 y subsiguientes implican saltos en una red como IP para sincronizarse los unos (en general stratum n) con los otros (en general stratum n-1)[55].

[55]En el caso de España la hora legal es dictada por un real observatorio de la armada en San Fernando, disponiendo de una Time Stamping Authority asociada. Allí existe un servidor NTP stratum 1 asociado a ese centro. Otro stratum 1 en Andalucía (basado en GPS) es hora.cica.es, presente en el pool 3.debian.pool.ntp.org que viene por defecto en la configuración de ntpd en Debian.

En nuestro despliegue, los servidores NTP en dklab1 y dklab2 intentarán sincronizarse con otros servidores en stratum 1 hasta 4 pertenecientes al pool "pool.ntp.org". Si pierden la conexión con el exterior o bajan de stratum 4, se activa el modo Orphan por el cual se ponen de acuerdo para ofrecer una fuente independiente pero única de tiempo a la infraestructura que depende de ellos.

Modo unicast peers (simétrico activo/pasivo) en asociación permanente

Este modo de operación consigue que los servidores negocien dinámicamente quién se sincroniza de quién con el fin acoplarse lo más estrechamente posible.

En nuestro caso dklab1, dklab2 y las fuentes de pool.ntp.org a las que se conecten forman un grafo acíclico parcialmente dirigido que cambia con el tiempo para maximizar la precisión. Por "parcialmente dirigido" nos referimos a que todos los servidores mantienen permanentemente un conjunto de conexiones que periódicamente supervisan y de las que sólo una se va eligiendo como fuente de sincronización. Además (siguiendo con la representación en grafo) en el caso de la arista dklab1-dklab2 puede no estar dirigida o estarlo, pero en ese caso puede dirigirse tanto en un sentido como en el otro porque al ser "unicast peer's" ambos pueden actuar activa o pasivamente entre sí.

Descubrimiento del servicio tipo "pool"

El tercer grupo de actores son los clientes finales que se comportan pasivamente, es decir inician una conexión como clientes a los servidores para que éstos les sincronicen. De forma abstracta serían los nodos hoja del árbol que forman las aristas dirigidas en el grafo mencionado anteriormente.

El descubrimiento del servicio se efectúa a través de DNS. Como se indicó al hablar sobre nuestra política de nombres, NTP define registros SRV pero la implementación de Ntp.org no implementa su uso. El mecanismo, entonces, tradicional es que a los clientes se les indique un registro A múltiple (es decir varias direcciones corresponden suprayectivamente a un sólo nombre) de forma que los clientes, configurados en modo "pool", descubren todos los servidores posibles a través de ese nombre e inician una conexión a todos o algunos de ellos.

En nuestro caso el pool estará tras el nombre "ntp.casafx.dyndns.org." y las IP a las que resuelve serán las de dklab1 y dklab2.

Otro anexo, el 7, en sus apartados 1.6.2 y 1.6.3 (epígrafes dedicados a NTP) detalla cómo configurar hipotéticos clientes Debian, Windows y MAC OSX para utilizar el pool de nuestra infraestructura.

Esquema de auntenticación "Autokey v2 IFF"

Autokey representa en síntesis un servicio de autenticación del servidor al cliente, basado en criptografía asimétrica en el que además se negocia una llave compartida que usar en un algoritmo rápido. Funciona ligeramente distinto en cada modo (server, peer's,

broadcast) y además se han definido varios subesquemas: IFF, GQ, MV, otros. Nosotros utilizaremos IFF porque es flexible hasta adaptarse al despliegue que estamos exponiendo[56]. De hecho, MV es el subesquema más flexible, pero esa flexibilidad se muestra en escenarios distintos al nuestro[57].

Autokey v2 IFF parte de un grupo de servidores que se sincronizan entre sí. Para identificarse los unos con los otros, comparten una llave de grupo que constituyen los parámetros IFF (son mapeables 1:1 con una estructura DSA) y pueden verificar una cadena de certificados X509v3.

Concretando algo más el uso de los certificados, cada servidor es lanzado con un certificado autofirmado, sólo que aquellos servidores del grupo con menor stratum añaden la extensión X509v3 "Extended Key Usage" con el valor "Trust Root"[58] y se consideran TA (de "Trusted Authority", también expresado a veces como TH). Otro detalle del certificado de los TA es que los atributos Subject e Issuer tienen por CN el nombre del grupo[59], en lugar de otro nombre (los no TA tendrán un nombre por cada servidor, de forma que se suelen usar los nombres del sistema DNS, que no el protocolo DNS)[60]. Puede haber varias TA en el grupo.

Pongamos ahora que uno de los servidores del grupo (llamémoslo no-TA) se sincroniza con uno que es TA. Para ello se inicia, carga su certificado autofirmado y al comprobar que no lleva "Trusted Root", pide al TA que se lo firme. Si todo va bien, acaban completando la autenticación y se sincronizan. Imaginemos entonces que, a continuación, un tercer servidor NTP del grupo (que tampoco es TA luego lo llamaremos otro-no-TA) contacta con "no-TA" e intenta sincronizarse con él previa autenticación. Tenemos por tanto una jerarquía de 3 niveles: TA, no-TA y otro-no-TA. Éste último pide entonces a no-TA su certificado y, a cambio no-TA le manda su certificado firmado por la TA así como el certificado de la TA. Al recibirlos, otro-no-TA puede comprobar que aunque no es "Trusted Root" sí ha sido firmado por uno que lo es. Por ello, otro-no-TA ya considera confiable a no-TA y, claro, si es confiable, puede pedirle que firme su certificado autofirmado.

Extendiendo el proceso hacia stratums más alejados de los TA, se genera un camino de certificaciones que se puede verificar. Si cada host además de demostrar que es confiable demuestra que tiene la llave IFF de grupo, pertenece al grupo y puede, finalmente, ser usado como fuente de hora.

Podemos anotar además que cuando un TA de un grupo se conecta a otro grupo (ésto ya simplemente en modo cliente), surgen las jerarquías de grupo.

Conocido el funcionamiento básico de IFF, podemos plantearnos ahora sobre su pertinencia. Ciertamente, si se tiene en cuenta que las asociaciones jerárquicas entre servidores NTP de un grupo o jerarquía ocurren ad hoc en la mayoría de los escenarios, este esquema de autenticación puede ser más conveniente que, por ejemplo, el de PKIX. La clave está

[56]Ésto podría incluir también la posibilidad de reusar nuestra CA. Se ha reservado para más adelante una discusión al respecto.

[57]Por ejemplo, MV puede considerar que los clientes no sean confiables y habrían de probar su identidad para utilizar el servicio. No es nuestro caso.

[58]Como se detalla en el anexo, cuando se utiliza la utilidad ntp-keygen ésto se consigue al pasar la opción -T.

[59]De nuevo, por ilustrar, ésto se consigue con el argumento del flag -i de ntp-keygen.

[60]Con el argumento de otro flag de ntp-keygen, -s.

en las ventajas o desventajas que extrae cada sistema de las relaciones temporales entre las partes. Efectivamente, en el caso de la PKIX los certificados proveen memoria de un acontecimiento pasado, la firma por la CA, pero a cambio obligan a que en el momento de la autenticación, la parte que intenta verificar la identidad del autenticado tenga que estar en posesión del certificado de aquella CA. En los sistemas en que se utiliza PKIX ésto es ventajoso. Autokey v2, teniendo en cuenta el dinamismo en las comunicaciones NTP del que hablábamos, considera que en NTP no tiene sentido suponer que todas las partes posibles disponen del certificado de la CA. Entonces toma otra dirección.

Restrospectivamente, ese certificado representaba la memoria de la acreditación de la CA, pero esa memoria del pasado no es necesaria porque en un servicio sistema-sistema como es el de tiempo, la autoridad sobre la información recae en otro servidor, el de stratum 0 (o el más cercano) y éste actúa concurrentemente a los demás, y de ahí la estrategia de crear "al vuelo" las cadenas de certificación. Así, en lo que respecta a las relaciones temporales, Autokey v2 toma como ventaja la no necesidad de memoria, mientras que PKIX toma ventaja de la memoria que aportan las firmas digitales de la CA en el pasado.

Podemos añadir finalmente que, además, el diseño de Autokey v2 consigue que los servidores NTP se comporten "Stateless" (sin necesidad de guardar variables por cada conexión) conservando las críticas cualidades temporales del servicio[6].

Tras la discusión anterior, estamos en condiciones de llevar Autokey a nuestro despliegue: dklab1 y dklab2 serán, bajo el esquema IFF, TA del grupo "casafx.dyndns.org" y, por lo reducido del despliegue, sólo dejaremos indicaciones para que otros hipotéticos servidores que pudiésemos incorporar en un futuro pudieran actuar como peer's no-TA que establecieran las cadenas de certificación mencionadas. El resto de nuestro hipotético parque informático actuaría como clientes de los anteriores.

Éstos últimos pueden usar opcionalmente Autokey v2 para autenticar al grupo IFF. En su caso el par criptográfico no llevará, evidentemente, la extensión "Extended Key Usage" a "Trusted Root". Además no tienen la clave de grupo, pero pueden saber si el otro extremo sí la tiene. Puesto que el cliente puede verificar que el servidor tiene la llave sin que el cliente conozca ésta, Autokey v2 IFF es un esquema "Zero-knowledge Proof" y lo hace más resistente ante ataques MitM. Finalmente diremos que, cuando el cliente comprueba que su servidor tanto pertenece al grupo como su cadena de certificados hasta el TA es verificable, puede confiar en él y utilizarlo como fuente de hora.

Posibilidad de reusar nuestra CA Queremos abordar ahora el hecho de que el subesquema IFF fue concebido para permitir que librerías criptográficas como las de OpenSSL pudiesen generar el material criptográfico[61]. Conocemos adicionalmente que[6]:

- Los certificados X509v3 que se crean para cada servidor llevan la extensión "Basic Constraints" a "CA:TRUE" (porque son válidos, efectivamente, para firmar). Los de la TA llevan la extensión "Extended Key Usage" a "Trusted Root".

- Respecto a los certificados "al vuelo", diremos que en cada firma el servidor NTP dependiente le pasa al servidor de menor stratum su certificado autofirmado. Allí es

[61] http://www.eecis.udel.edu/~mills/ident.html

entendido éste como una petición de certificado y, tras extraer tanto la clave pública como "Issuer", "Subject" y "Extension Fields", lo firma con su certificado y le copia además el atributo "Serial Number".

- Los parámetros IFF son mapeables a DSA luego se puede usar directamente la implementación DSA de OpenSSL.

En general todos esos requisitos e indicaciones nos hacen pensar que el nivel de conocimiento que pusimos en práctica al crear nuestra CA para OpenVPN es suficiente como para adaptar ésta ahora y utilizarla para administrar el material criptográfico también en Autokey v2 IFF. Sin embargo tres hechos nos hacen rechazar esta opción:

- Conocemos avisos de posibles dificultades: "Certificates can be generated by the OpenSSL library or an external public certificate authority, but conveying an arbitrary public value in a certificate extension field might not be possible"[6].
- No existe información paso por paso para utilizar OpenSSL (no la hemos encontrado). Autokey v2 incluye varios subesquemas, es difícil encontrar información para IFF y cuando la hay es para la utilidad ntp-keygen de Ntp.org.
- El proceso por el cual completamos el despliegue fue, ciertamente, insidioso.

Por tanto las dificultades que tuvimos usando ntp-keygen, la falta de documentación para llevarlo a cabo con OpenSSL y la previsión de problemas irresolubles nos desanimaron a intentar utilizar nuestra CA.

Autorización

La política por defecto es que cualquiera, previa autenticación o sin ella, puede consultar la hora y es el comportamiento por defecto.

Anotación TSP

Un concepto relacionado con NTP es el protocolo TSP (de "Time Stamp Protocol").

Básicamente, el autor de cierta información electrónica utiliza un par criptográfico para generar una petición de "timestamp" (sellado de tiempo) dependiente del documento (HASH) y la envía a una TSA ("Time Stamp Authority") utilizando el protocolo TSP. La firma digital de la TSA permitirá asegurar ante terceros que la información electrónica guarda una relación temporal con algún instante de tiempo. El sistema está modelado en torno a PKIX y utiliza certificados X.509.

Evidentemente, una TSA necesita una fuente segura de tiempo, pudiendo emplear hardware especial directamente, o bien, efectivamente, NTP y Autokey v2.

No hemos encontrado ninguna implementación de TSP en Debian 6 "Squeeze". El proyecto OpenTSA (activo al menos hasta 2006) implementaba TSP (cliente y servidor) utilizando de backend a OpenSSL modificado. Los parches que proveían hacían que

"openssl" admitiese un subcomando "ts"[62] pero ésto no ocurre en el binario del que disponemos en Debian luego, inicialmente, no tenemos soporte de ningún tipo y el despliegue de TSP queda pendiente para una reevaluación futura.

Diremos finalmente que, una vez superada la dificultad de evaluar los distintos modos de operación, roles o lograr crear el material criptográfico, la configuración de "ntpd" es sencilla y se realiza en un único fichero. Las secciones 1.1.1 y 1.2.1 detallan el proceso. Adicionalmente se expone un ejemplo de uso de las utilidades de monitorización y depuración en la sección 1.3.

[62] http://www.openssl.org/docs/apps/ts.html

1.3.8. KERBEROS y LDAP

Anexo:	4

Abordamos ahora el servicio base para el almacenamiento de metadatos en nuestra infraestructura. Esas necesidades de almacenamiento podemos resumirlas en:

1. Principals KERBEROS.
2. Identidades y ACL relativas a AFS.
3. Nombres del sistema DNS, ya expuesto anteriormente.
4. Metadatos relativos a las Cuentas Centralizadas de Shell POSIX[63].
5. Información adicional de las cuentas independiente de POSIX[64].
6. Metadatos relativos a la gestión del correo[65].
7. Metadatos relativos a la gestión del servicio de presencia y otros relativos al servicio de mensajería instantánea y VoIP que no se incluyen en el punto 5.

Podemos agrupar los metadatos anteriores tomando como criterios el patrón de uso y las interrelaciones con otros metadatos. Entonces estaremos en disposición de utilizar una u otra tecnología para su gestión.

El grupo que reuniría a los puntos 1, 4, 5, 6 (en color **verde**) son metadatos que suelen ser infrecuentemente modificados y que tienden a agruparse de forma heterogénea y difícil de definir a priori con otro tipo de datos. Puesto que un sistema de directorio efectivamente está orientado a lecturas más que escrituras, y puesto que un sistema de directorio estructura la información de forma arbórea donde cada nodo permite almacenar datos de cualquier conjunto de atributos (predefinidos), esos metadatos estarán gestionados usando LDAP, el protocolo ligero de acceso a directorios.

El grupo que reuniría los puntos 2 y 3 (en color **negro**) corresponde a información que, presentando las mismas cualidades que los anteriores y por tanto siendo susceptibles de ser almacenados en LDAP, no se hace así. En el caso de los metadatos relativos a AFS, porque no está implementado el soporte de almacenamiento usando LDAP. En el caso de nombres del sistema DNS, a pesar de estar implementado, no lo utilizamos por los motivos que se discutieron al hablar de "Transferencia de Zonas" en DNS.

El punto 7 (color **azul**), representa metadatos que son frecuentemente modificados y cuyas relaciones entre sí son fácilmente definibles a priori de forma relacional. Por tanto estarán gestionados por un gestor relacional SQL[66]. Podemos añadir, además, la consideración de que en nuestro despliegue no existen necesidades masivas de almacenamiento de este tipo de metadatos. Por tanto, no se hace necesario utilizar otras tecnologías que rompan el modelo ACID[67] para ganar en escalabilidad (sería el caso de Hadoop, MapRe-

[63] Como UID, GID, home...
[64] Como un avatar gráfico, una dirección web...
[65] Como aliases, direcciones de reenvío, cuotas, preferencias ante Spam...
[66] En nuestro despliegue, podemos adelantar ya que será Jabberd2 (una implementación de servidor XMPP) el software que haga uso del RDBMS SQL.
[67] De transacciones atómicas, consistentes, aisladas y durables.

duce o las tecnologías agrupadas recientemente bajo la denominación de mercadotecnia "NoSQL": Memcached, CouchDB, MongoDB y otras muchas).

En la presente sección expondremos las particularidades encontradas al usar LDAP, y dejaremos SQL para más tarde.

LDAP[68] es la versión ligera y sobre TCP/IP de DAP ("Directory Access Protocol", interfaz al sistema X.500 basado en redes OSI). Por ligera entendemos que no soporta funcionalidades raramente usadas, con lo que se facilita su implementación, despliegue y posterior uso: efectivamente ha sido ampliamente adoptado. Como otros protocolos de nuestro despliegue (KERBEROS, AFS, SQL...) permite a las implementaciones delegar partes de la información, de forma que por ejemplo cada departamento de una organización pudiera hacerse cargo de sus datos.

Necesitamos un gestor LDAP que cumpla el estándar y cubra las siguientes funcionalidades:

- Permita centralización, replicación de forma pseudo-síncrona y topología de replicación Multi-Master.
- Actualización dinámica sin necesidad de reinicio.
- Backend de almacenamiento transaccional jerárquico con posibilidad de renombrado de subramas.
- Subprotocolo SASL de negociación de mecanismo de autenticación, así como, específicamente, mecanismo de autenticación SASL-GSSAPI (basado en KERBEROS).
- Adicionalmente, posibilidad de autenticación anónima.
- Autorización basada en ACL de alta granularidad.

OpenLDAP[69] es una implementación que cumple todos los anteriores requisitos.

Efectivamente nos permite centralizar y redundar los metadatos. El estándar LDAP se extendió[70] con un esquema de replicación como una ampliación de las búsquedas LDAP. Éste permitie tanto actuar por sondeo (modo "refreshOnly") como por interrupción (modo "refreshAndPersist") y no sigue un modelo transaccional. OpenLDAP ha implementado y extendido el estándar, incluyendo soporte para una topología de replicación Multi-Master.

El servidor se distribuye como el binario "slapd", y desde la versión 2.4 la interfaz para configurar su comportamiento es también LDAP. Es por ello que cualquier cambio, afecte a la configuración del servicio o al resto, se hace disponible inmediatamente sin necesidad de reinicio.

OpenLDAP no implementa aún la extensión para transacciones LDAP en operaciones que afecten a varias entradas[71]. Sí utiliza como soporte de almacenamiento a una versión

[68] http://tools.ietf.org/html/rfc1777
[69] http://www.openldap.org
[70] http://tools.ietf.org/html/rfc4533
[71] http://tools.ietf.org/html/rfc5805

jerárquica de la base de datos embebida y transaccional Berckeley DB. En terminología de OpenLDAP, es llamado "backend HDB" (donde la "H" viene de "hierarchical", jerárquica).

OpenLDAP ofrece servicios de seguridad como autenticación, autorización, integridad y confidencialidad. El mecanismo de autenticación se puede negociar a través del subprotocolo SASL[72], caso en el que está disponible el mecanismo SASL-GSSAPI. Éste, definido en el rfc4752[73], permite por un lado utilizar la API de alto nivel GSS-API[74] para usar el mecanismo de autenticación KERBEROS y hacer SSO, y por otro lado, adicionalmente, permite negociar una capa de seguridad que ofrezca integridad y confidencialidad a la comunicación posterior. Esta capa de seguridad puede hacer innecesaria la proveída por el también soportado TLS[75]. Alternativamente a los servicios anteriores, OpenLDAP permite autenticación no SASL basada en contraseña ("simple bind") y anónima (sin credencial alguna).

Finalmente, OpenLDAP soporta un servicio de autorización basado en ACL de alta granularidad ("brutally complex"[76]). Éstas pueden restringir qué (nombres distinguidos, atributos, ...), quién (self, anonymous, identidades autenticadas, nombre distinguido, IP, DNS...) y cómo (si TLS, algoritmo, longitud de clave, ...), o con qué nivel de acceso (read, write, search, compare, ...).

Llevado a nuestro despliegue, dklab1 y dklab2 albergarán cada uno una instancia de "slapd" que, a excepción de la rama destinada a albergar su configuración, se sincronizarán entre sí en modo "refreshAndPersist", constituyendo una topología Master-Master pero extensible a Multi-Master si se incorporasen otros gestores OpenLDAP. Ambos utilizarán HDB como backend de almacenamiento, sin que hayamos previsto para éste ajustes[77] distintos a los que ocurren por omisión (a excepción, ciertamente, de la creación de índices).

Configuraremos slapd para que acepte SASL-GSSAPI como mecanismo de autenticación, pero no lo usaremos en la rama correspondiente a los metadatos de las cuentas centralizadas de nuestra organización. Ésto es una decisión de relieve en nuestro despliegue, veamos pues:

La idea[7] es que en la rama para las cuentas no tenemos información sensible ya que todas las credenciales (principals KERBEROS) se almacenan en otro subárbol distinto. Por tanto, siempre que podamos mantener esta premisa, el acceso a la rama de las cuentas

[72] http://tools.ietf.org/html/rfc4422
http://tools.ietf.org/html/rfc4752

[73] http://tools.ietf.org/html/rfc4752

[74] http://tools.ietf.org/html/rfc2743

[75] Aunque en teoría no es necesario usar TLS cuando usamos SASL-GSSAPI, lo cierto es que de las tres implementaciones que conocemos de SASL (Cyrus, Gsasl, Dovecot-Sasl) sólo la primera implementa la parte del estándar dedicada a la capa de seguridad. OpenLDAP utiliza Cyrus SASL, luego no es necesario usar TLS. No obstante si lo fuese por otra razón, recomendamos recompilar OpenLDAP contra OpenSSL, puesto que por razones legales Debian lo distribuye enlazado contra GnuTLS y, durante nuestras pruebas, tuvimos muchos problemas (que se añaden a lo especialmente complicado que es indicar a slapd que lo use a través de sus ACL).

[76] http://www.zytrax.com/books/ldap/ch6/
"LDAP for Rocket Scientists".

[77] http://www.openldap.org/doc/admin24/tuning.html

puede ser anónimo y, con ello, el resto del despliegue se facilita en muy alto grado. Si en un futuro surge la necesidad de eliminar dicha restricción y la rama para cuentas almacena información sensible, podemos inmediatamente utilizar un sistema robusto como KERBEROS puesto que el servicio, como dijimos, se deja kerberizado.

Modificaciones del árbol LDAP; diseño de nuestro DIT

La información se estructura de forma arbórea, donde cada base de datos independiente es denominada DIT ("Directory Information Tree") y está representada por los nodos que nacen, en general, del primer nivel de la jerarquía.

Cada DIT es, haciendo un símil, como el espacio de nombres en un sistema de ficheros POSIX, pero donde:

- El nombre del recurso se expresa con el formato <tipo de atributo>=<nombre>(no sólo con <nombre>), llamado RDN ("Relative Distinguished Name", véase DN a continuación).
- La ruta se expresa retrógradamente hacia el raiz (al contrario que en POSIX), y constituye su DN ("Distinguished Name").
- El separador es el símbolo "," (coma, no "/").

Por ejemplo las siguientes dos líneas expresan el mismo camino en el árbol, la primera como sería en un sistema de ficheros POSIX, la segunda (dados unos <tipo de atributo>en cada nodo) como sería en LDAP DN:

- /org/dyndns/casafx/accounts/users/umea
- uid=umea,ou=users,ou=accounts,dc=dyndns,dc=casafx,dc=org

Los atributos que contiene cada nodo se pueden considerar instancias de clases de objetos predefinidos y conocidas por el sistema. Algunas de esas clases de objetos son "Structural", y cada nodo debe instanciar atributos de una y sólo una de esa clase especial de objetos; las demás clases (llamadas "Auxiliary") no están sujetas a esas condiciones. Cuando el RDN se expresa con el formato <tipo de atributo>=<nombre>, <tipo de atributo>es alguno de los permitidos por las clases de objetos de las que instancia ese nodo, y debe ser alguno que impida duplicidades en el mismo nivel jerárquico.

Es el administrador quien carga los esquemas con clases de objetos en OpenLDAP[78], el que diseña qué clase de objetos de las disponibles usa cada nodo, el que define a qué atributos de esas clases de objetos usadas por el nodo se les da un valor y a cúales no, y

[78] Algunos esquemas vienen precargados por el proyecto OpenLDAP por ser estándar, otros los provee cada software que use LDAP, y otros los diseñamos nosotros. Durante el despliegue del servicio de correo, se muestra un ejemplo de diseño de estas clases de objetos al extender un esquema, ejemplificando así el uso del espacio de OID's (identificadores de objetos) privado asignado por la IANA, relaciones de herencia entre clases de objetos, atributos de instanciación obligatoria y opcional, etc.

el que escoje cuál de esos atributos se usa para formar el RDN. Los RFC 2377[79] y 4519[80] describen algunas recomendaciones a la hora de diseñar un espacio de nombres LDAP (también referido como "LDAP Name Planning"), incluyendo qué <tipo de atributo>usar como RDN en ciertas situaciones comunes.

Por ejemplo, es habitual que el nombre del DIT para una organización se construya aprovechando el sistema de nombres de dominio, sustituyendo el "." en ese sistema por el atributo dc (de "Domain Component") proveído en el objeto "dcObject" del esquema "core". Otro ejemplo, en este caso para nodos que actúan como los directorios de un sistema de ficheros (es decir que sirven básicamente para aunar otras entidades en un mismo nivel de la jerarquía); en LDAP son nodos que instancian el atributo ou (de "Organizational Unit") del objeto "organizationalUnit" del esquema "core". Finalmente nodos que representan cuentas de aplicaciones, suelen[81] usar para construir su RDN el atributo uid del objeto "posixAccount" de la clase "nis". Observando una vez más el ejemplo de DN anterior:

- uid=umea, ou=users, ou=accounts, dc=dyndns, dc=casafx, dc=org

... es fácil leer que la metainformación del dominio casafx.dyndns.org. estará almacenada en el DIT LDAP dc=dyndns,dc=casafx,dc=org. Hay una rama especial, ou=accounts, para albergar metainformación específica de las cuentas de ese dominio. Y dentro de ésta hay otra rama ou=users para metainformación que siendo de cuentas es sólo relativa a usuarios (en oposición a que habrá otra rama para información de cuentas pero sólo relativa a grupos por ejemplo). Por último, una de esas cuentas a las que hace referencia este DN particular es llamada "umea" gracias a su atributo uid.

Otro atributo muy frecuente que puede aparecer es cn, de "Common Name". Como veremos a continuación puede instanciarse de varios objetos estructurales, pero uno especialmente importante es "organizationalRole". Éste se usa para definir nodos que corresponden con roles en una organización y, por tanto, su DN es susceptible de ser usados como identidad de autenticación y autorización en LDAP. Si la autenticación está basada en contraseña, se suelen usar junto al objeto "simpleSecurityObject" porque éste permite instanciar un atributo que represente una contraseña. Sin embargo, cuando el nodo ha de representar tanto un rol como, específicamente, una cuenta POSIX, se sustituye "organizationalRole" y su atributo "cn", por "posixAccount" y su atributo "uid" que ya hemos comentado en el párrafo precedente. Por su lado "simpleSecurityObject" se sustituiría por "shadowAccount" pues provee atributos para contraseñas y para su administración acorde a POSIX.

Cuando instalamos OpenLDAP, efectivamente sólo hay un DIT. Su DN es "cn=config" y está dedicado a almacenar la configuración de slapd. Uno de los nodos que cuelgan de este DIT "cn=config" es "cn=schema, cn=config", cuyo cometido es almacenar los esquemas con clases de objetos y atributos que luego usa el resto del árbol. Efectivamente ya conocemos a éstos pero no sabíamos dónde residían.

[79] http://tools.ietf.org/html/rfc2377
[80] http://tools.ietf.org/html/rfc4519
[81] Es decir, tal como sugiere la recomendación rfc2307.

Otro nodo que cuelga de cn=config es "olcDatabase={-1}config, cn=config" y contiene la configuración por defecto del *comportamiento* de OpenLDAP para todos los DIT. Si cambiamos "-1" por "0", tenemos el DN del nodo que almacena la configuración no por defecto del propio DIT cn=config. Ésto implica que por cada DIT que nosotros definamos a continuación, en cn=config se creará a la par una rama para contener la configuración no por defecto relativa a ese DIT. Por ejemplo si nosotros, como venimos anunciando, vamos a crear el DIT dc=casafx,dc=dyndns,dc=org entonces automáticamente a la creación de éste se creará también "olcDatabase={1}hdb, cn=config". Aunque el sistema prefiere crear el RDN utilizando el índice y el tipo de backend, si lo inspeccionamos utilizando las herramientas de OpenLDAP veríamos que otro de los atributos definidos para ese nodo es olcSuffix de valor, efectivamente, el string "dc=casafx,dc=dyndns,dc=org" luego a modo de puntero indica el DN de nuestro DIT al que se refiere esa configuración específica en el DIT "cn=config".

Una de las variables de configuración de cada DIT y que se define en los nodos bajo los DN que acabamos de mencionar son las que controlan el proceso de autenticación. Por ejemplo se pueden especificar qué mecanismos están permitidos. Además, en mecanismos como KERBEROS hace falta declarar el mapeo de la identidad KERBEROS a la identidad LDAP, es decir el mapeo del principal KERBEROS a el DN del nodo[82] que representa al rol en LDAP (por ejemplo alguno que instancie de "organizationalRole" como se comentó) y con el que se hace efectiva la autenticación y autorización. Estos mapeos se hacen, sin embargo, directamente en cn=config a través de atributos olcAuthRegexp.

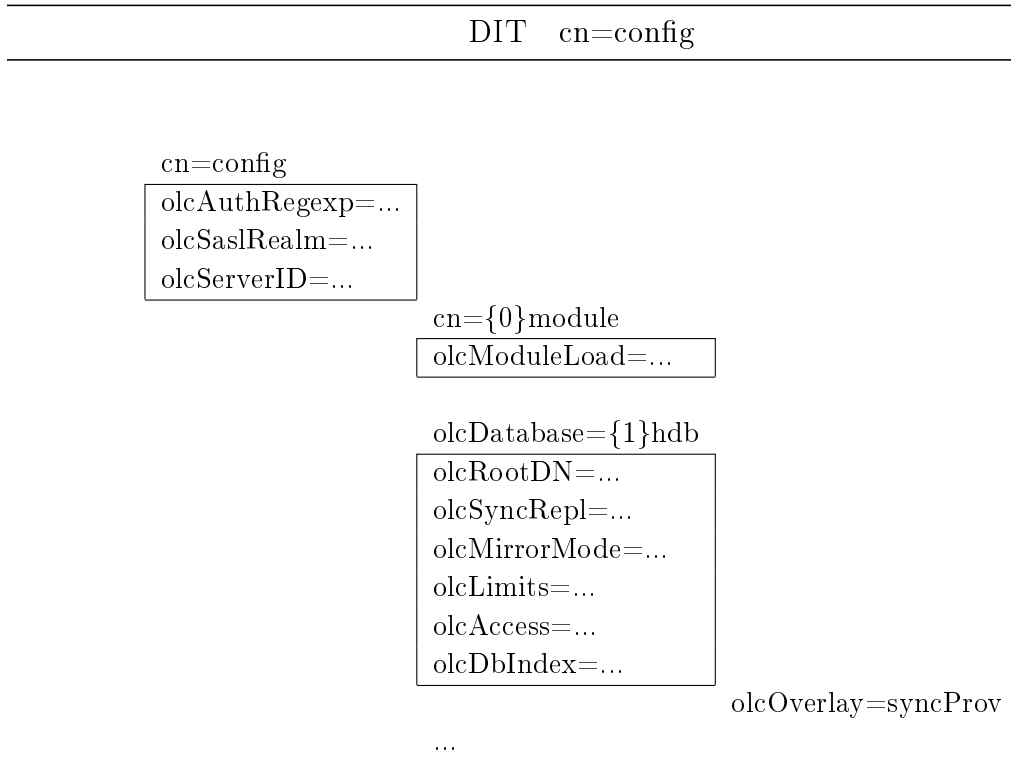
Respecto a la autorización que acabamos de mencionar , efectivamente otra de las variables de configuración de cada DIT son los atributos olcAccess, que corresponden a las ACL de LDAP y de las que nos ocuparemos luego.

Una tercera variable que representa un caso especial de autenticación y autorización es el atributo olcRootDN del objeto "olcDatabaseConfig". El valor del olcRootDN se interpreta como la identidad que estará plenamente autorizada para realizar cualquier acción en ese DIT, ya que no se le aplican las ACL de ese DIT. Existen dos posibilidades, que el valor de olcRootDN apunte a un nodo del árbol (una vez más, por ejemplo alguno que instancie de "organizationalRole" como se comentó), o que no. Éste último caso se aprovecha para tener una identidad que pueda administrar cualquier DIT de cualquier instancia de OpenLDAP, sin necesidad de que esté definido un nodo local para ello, en su lugar está definido siempre el mismo olcRootDN.

Estamos ya en condiciones de describir las modificaciones efectuadas en el DIT cn=config y en el de nueva creación para nuestra organización.

[82]No es estrictamente necesario, para la validez de la autenticación, la existencia de una entidad en el DIT a la que haga referencia el mapeo del principal (puede consultarse "man slapd-config" al respecto del atributo "olcAuthzRegexp"). No obstante, nos será útil al definir las ACL.

El siguiente árbol representa de izquierda a derecha las ramas de cn=config donde hemos introducido atributos especiales, los cuales encuadramos bajo cada rama.



Efectivamente el nodo cn=config contendrá atributos olcAuthRegexp y olcSaslRealm para mapear principals KERBEROS a DN LDAP. El tercer atributo, olcServerID asigna un identificador útil para diferenciar a los replicadores en la topología Multi-Master.

Un nodo hijo del anterior, "cn={0}module" instancia un atributo olcModuleLoad que indica a slapd que debe cargar la librería que implementa las funcionalidades de replicación.

Otro nodo hermano del anterior, "olcDatabase={1}hdb", representa como sabemos la configuración del comportamiento de slapd para el DIT "dc=casafx, dc=dyndns, dc=org" y era registrado automáticamente a la creación de éste. Nosotros le añadiremos un nodo hijo, "olcOverlay=syncProv". Efectivamente ese nodo hijo junto a los atributos olcSyncRepl y olcMirrormode indican a slapd que se utilicen para aquel DIT los modos que describimos del sistema de replicación: el rol de proveedor, el rol de consumidor y la posibilidad de escrituras completando el esquema Multi-Master.

Habrá varios atributos olcAccess que representan las ACL[83]. Como se introdujo, lo fundamental es que la rama con la información de las cuentas centralizadas podrá ser

[83]El crédito de tales ACL corresponde, con alguna excepción adaptada a nuestro despliegue, a Jaap Winius. Así ocurre en general con los detalles de integración entre MIT Kerberos y OpenLDAP. Puede consultarse su magnífica documentación en:
http://rjsystems.nl/en/2100.php

leída por cualquiera[7]. No así otras como la dedicada a almacenar los principals en LDAP excepto para las identidades LDAP creadas a tal fin o las responsables de la replicación.

Otro orden distinto de limitación lo controla el atributo olcLimits. Indicará a slapd que no debe restringir el tamaño de datos transferido en una conexión. Es el valor recomendado durante el período de pruebas pero que más tarde puede ser reconfigurado para ajustarse al tipo de entorno en producción. Los atributos olcDbIndex, por su lado declaran qué atributos debieran ser indexados para hacer más eficientes las búsquedas[84]. Su mantenimiento requiere algunos recursos, de forma que sólo indexamos los atributos de los que tengamos conocimiento que son frecuentemente usados. El propio slapd muestra en ocasiones, a través de syslog, mensajes de aviso sobre la conveniencia de indexar cierto atributo.

Respecto a nuestra política para el olcRootDN, declarará una identidad que no corresponde a ningún nodo del árbol sino que es el resultado de mapear cierto principal KERBEROS. La intención es que existirá una identidad que puede realizar labores de administración en todos los servidores (los de MIT Kerberos, OpenLDAP, OpenAFS, Jabberd2...) y para ello tenemos que definirla para KERBEROS primero[85] y mapearla a las identidades específicas de cada sistema después. El siguiente cuadro revela el principal KERBEROS elegido y su traducción en el caso que nos ocupa ahora, un DN LDAP acorde a las reglas de mapeo en el atributo olcAuthRegexp mencionado dos párrafos antes.

Identidad global de administración

Principal	DN
root/admin@CASAFX.DYNDNS.ORG	uid=root/admin, ou=users, ou=accounts, dc=casafx, dc=dyndns, dc=org

[84] http://www.openldap.org/faq/data/cache/42.html

[85] El crédito a esta forma de actuar se debe en nuestro caso a Davor Ocellic, cuya magnífica documentación nos presentó por primera vez los despliegues basados en MIT Kerberos, OpenLDAP y OpenAFS. http://techpubs.spinlocksolutions.com/dklar/kerberos.html#krb-adduser-priv

El siguiente árbol es creado íntegramente para almacenar la metainformación de nuestra organización:

DIT **dc=casafx, dc=dyndns, dc=org**

```
ou=krb5
                cn=kdc-srv
                cn=adm-srv
ou=consumers
                cn=dklab1
                cn=dklab2
ou=accounts
                ou=users
                                uid=umea
                                uid=aemu
                ou=groups
                                gid=umea
                                gid=aemu
```

Efectivamente, nuestro DIT contendrá tres subramas principales. La primera, ou=krb5, almacenará todos los metadatos relativos a MIT Kerberos, tal como los principals. Esas estructuras no aparecen en el árbol anterior porque no son administradas directamente por nosotros sino por MIT Kerberos. Los dos nodos hijos, cn=adm-srv y cn=kdc-srv, instancian atributos de organizationalRole. Constituyen los DN con que los servidores de MIT Kerberos, kadmind y krb5kdc, se autenticarán y autorizarán para modificar o leer en esa rama.

La rama ou=consumers, de forma similar crea dos nodos que instancian de organizationalRole y que constituyen los DN con que dklab1 se autentica y autoriza en dklab2 y a la inversa, en un contexto refreshAndPersist de replicación LDAP.

La tercera rama, ou=accounts, como hemos venido repitiendo hasta ahora, representa el subárbol donde se almacenan los metadatos de las cuentas centralizadas POSIX. Ésto incluye metainformación relativa a usuarios pero también a grupos, de ahí los dos nodos hijos ou=users y ou=groups. Finalmente, bajo ou=users presentamos las dos cuentas que serán creadas a lo largo del despliegue de forma que estén disponibles cuando se realicen comunicaciones en servicios persona-persona[86]. Ambos nodos instancian atributos de posixAccount entre otros[87].

[86]El nombre "umea" viene de Umeå, ciudad del norte de Suecia donde algunos alumnos de la Universidad de Jaén disfrutan de una beca Erasmus cada año. Por su lado "aemu" corresponde simplemente a la lectura en sentido contrario de umea, haciendo referencia a que ha sido creada para dar la réplica a ésta en los servicios persona-persona.

[87]La sección 1.7.2 del anexo 7 tenía, como dijimos, los detalles.

LDAP permite distribuir de forma disjunta el modelo de datos y dejar que cada instancia de slapd haga de proxy para las divisiones de las que no se haga cargo. En nuestro despliegue, como sabemos, no utilizamos esa funcionalidad y el árbol presentado es común en dklab1 y dklab2.

Todas las modificaciones anteriores se realizan indicándole a la utilidad ldapmodify que establezca una comunicación con slapd por un lado, y por el otro pasándole un fichero en formato LDIF que detalle las acciones que debe ordenar hacer. LDIF (de "LDAP Data Interchange Format") es un formato de tipo texto simple algo verborreico pero preciso definido en el RFC 2849[88]. Durante la comunicación, el protocolo LDAP es usado y por tanto la información se transforma y presenta como un lenguaje de mensajes ASN.1 codificados en BER.

Todos los detalles del despliegue pueden encontrarse en las secciones 1.1.3 y 1.2.3 del anexo. En general, la labor es complicada[89]. Además es crítico revisar muy cuidadosamente y adaptar las ACL antes de desplegar en un sistema en producción. Por su lado la sección 1.1.2 comenta algunos aspectos de la configuración de los clientes, es decir del software que se enlaza contra las librerías del proyecto OpenLDAP[90].

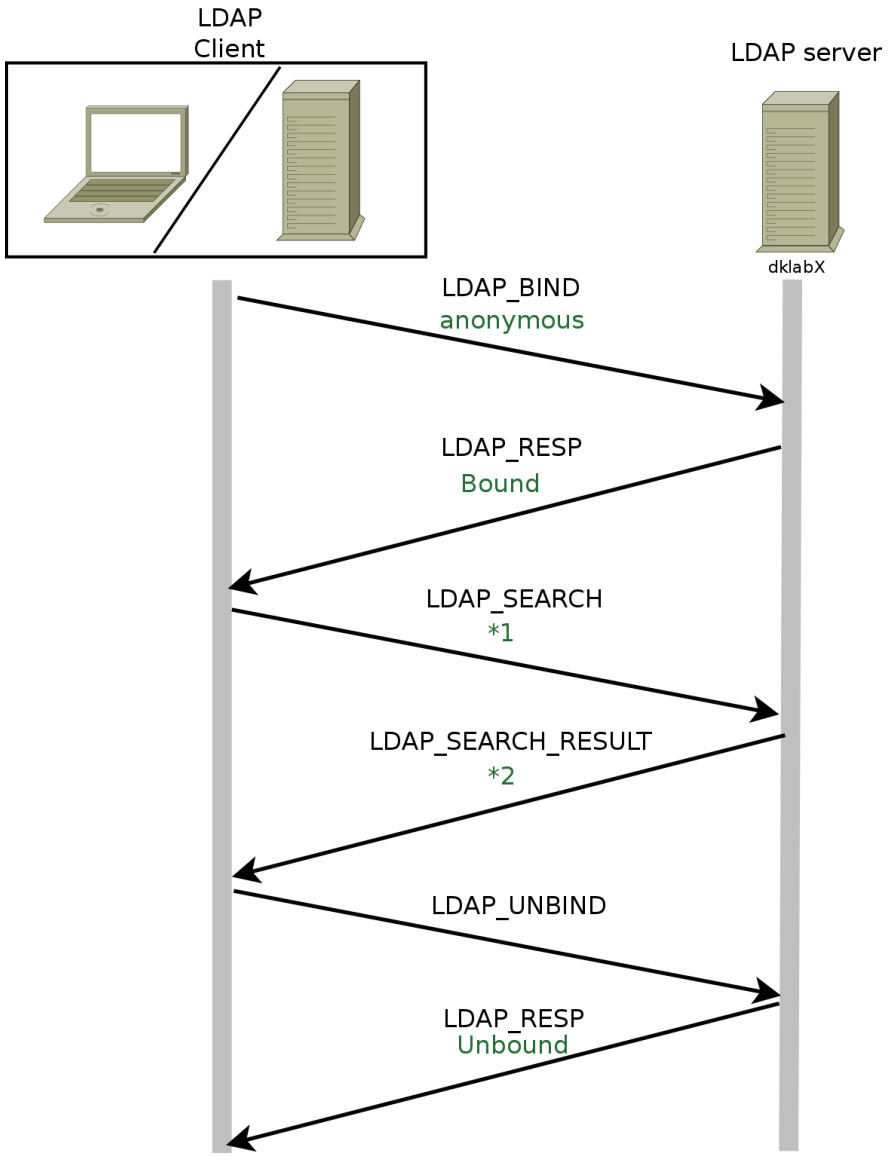
Ejemplo de comunicación LDAP

El siguiente diagrama intenta ilustrar cómo los clientes, ya sean otros servidores o los programas de usuario, consultan el árbol LDAP. Hasta ahora sabemos que el DN "ou=accounts, dc=casafx, dc=dyndns, dc=org" representa la rama bajo la que se encuentran los nodos con información sobre cuentas. Además, nuestra política de acceso (ACL's) permitía lanzar búsquedas con autenticación simple anónima.

[88] http://tools.ietf.org/html/rfc2849

[89] En general, la tasa de cambio de la curva de aprendizaje para LDAP es lenta, es necesario dominar varios conceptos y ponerlos en práctica en un lenguaje algo complicado. Existen utilidades para encapsular la complejidad del intercambio a través de LDIF. Nosotros hemos probado algunas como la aplicación web "phpLDAPadmin", si bien ésta hasta ahora tiene un bug que le impide utilizar SASL-GSSAPI. Existen otras como "GOSA" para, en particular, administración masiva de cuentas.

[90] Tal como libldap.so



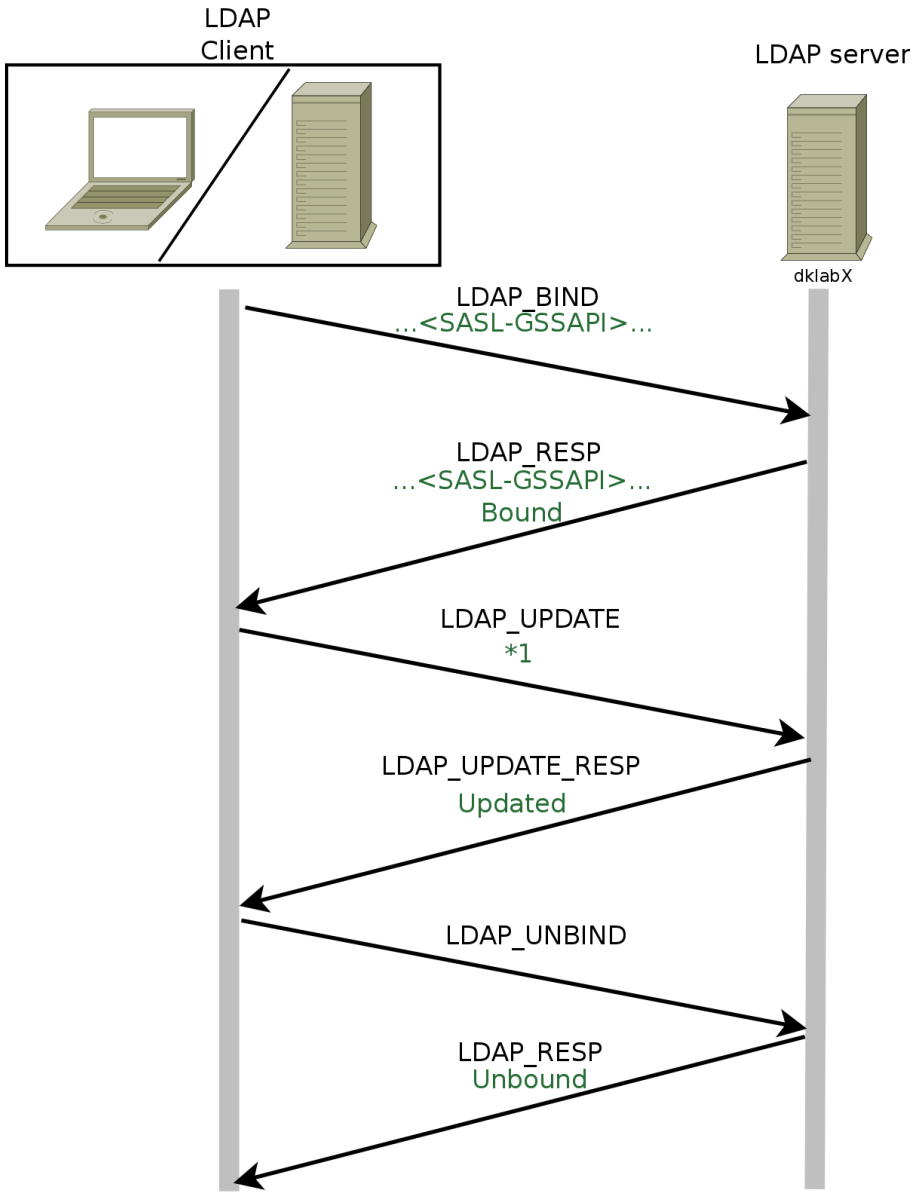
*1
```
base="ou=accounts,dc=casafx,dc=dyndns,dc=org" scope=2
filter="(&(objectclass=posixaccount)(uid=umea))"
attr=uid uidNumber gidNumber homeDirectory loginShell
```

*2
```
uid             umea
uidNumber       31000
gidNumber       31000
homeDirectory   /afs/casafx.dyndns.org/user/u/um/umea
loginShell      /bin/bash
```

Así pues el cliente descubrió el servicio (por ejemplo a partir de su registro SRV en DNS), se inicia la conexión y el cliente se autenticará (en terminología de LDAP, acción "bind") anónimamente. Si todo fue bien lanzará una consulta (en terminología LDAP, acción "search"). El motivo será conocer el valor de algunos atributos típicos POSIX,

como el número uid, el número gid, el home o la shell por defecto. La consulta se aplicará a todos los nodos que, colgando de la rama de cuentas (scope=2), instancien atributos del objeto posixAccount y tengan un atributo "uid" de valor "umea". Esta condición constituye un ejemplo de filtro LDAP. Si todo fue correctamente recibirá la respuesta ("result") y opcionalmente avisará de su desconexión ("unbind").

A continuación se muestra el mismo diagrama pero para lanzar una modificación en el árbol (en terminología LDAP, acción "update"). La única diferencia es que, dadas nuestras ACL, ahora el LDAP binding no puede ser anónimo, sino autenticado utilizando el mecanismo SASL-GSSAPI (y por tanto, KERBEROS). Más adelante se darán los detalles sobre el funcionamiento de KERBEROS, ahora nos interesa solamente que tras una autenticación exitosa, el cliente puede ordenar un "update". Por ejemplo la modificación puede consistir en crear una nueva cuenta a partir de la creación de un nodo que instancie atributos del objeto posixAccount etc bajo la rama users. Si todo fue bien, el cliente avisará de su desconexión ("unbind").



*1

```
dn: uid=umea,ou=users,ou=accounts,<DIT>
objectClass: top
objectClass: POSIXAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
uid: umea
loginShell: /bin/bash
homeDirectory: /afs/<cell>/user/u/um/umea
uidNumber: 31000
gidNumber: 31000
...
```

Interdependencias OpenLDAP/MIT Kerberos

Nuestro despliegue presenta efectivamente ciertas dependencias entre OpenLDAP y MIT Kerberos que nos obligan en el anexo 4 a presentarlos de forma conjunta. Ciertamente, de otra forma es fácil perder el curso lógico del despliegue y no entender la necesidad de cada paso.

Las dependencias entre ambos son:

- MIT Kerberos utiliza a OpenLDAP como su almacén de metadatos, como se ha puesto de manifiesto en el apartado anterior.
- OpenLDAP utiliza a MIT Kerberos para la autenticación SASL-GSSAPI, obligatoria en el caso de las comunicaciones para la replicación LDAP[91].

Rol de MIT Kerberos en el despliegue

Todas las comunicaciones en nuestra infraestructura proveen varios subservicios de seguridad. Entre ellos destaca el de autenticación, es decir el acto de confirmar la identidad de una entidad referida ésta a través de algún dato. Su importancia es central puesto que se desarrolla en los primeros pasos de la comunicación, en la "puerta de entrada" al servicio.

A su vez, como se describió en el apartado "Justificación" de este documento, el administrador debe identificar qué parte del proceso de una comunicación es especialmente reelevante para su trabajo, aislar esa parte e intentar implementarla de la forma más eficaz posible.

Siendo por tanto la autenticación una parte especialmente sensible, nos preguntamos entonces qué características debiera cumplir su implementación para ser robusta y fiable:

- Si usase algún tipo de credencial, ésta no debería almacenarse en la máquina del usuario sino que éste debería recordarla.
- Debería además ser única, independientemente del número de servicios.
- Idealmente sólo se debería interrumpir una vez al usuario para que diese prueba de conocer la credencial, independientemente del número de servicios y ocasiones de uso.
- Sin perjuicio de lo anterior, la contraseña no debería transmitirse por la red.
- Si hubiese de ser almacenada, sería encriptada y tras un servicio especial. Ésto adicionalmente facilitaría su administración al estar centralizada.
- Para no ser sensible a ataques MitM, la autenticación debería ser mutua: el servidor autentica al cliente, el cliente autentica al servidor.

[91] Concretamente era obligatoria en replicación, imposible en las de administración de los principals por parte de MIT Kerberos y opcional en el resto de comunicaciones en nuestro testbed.

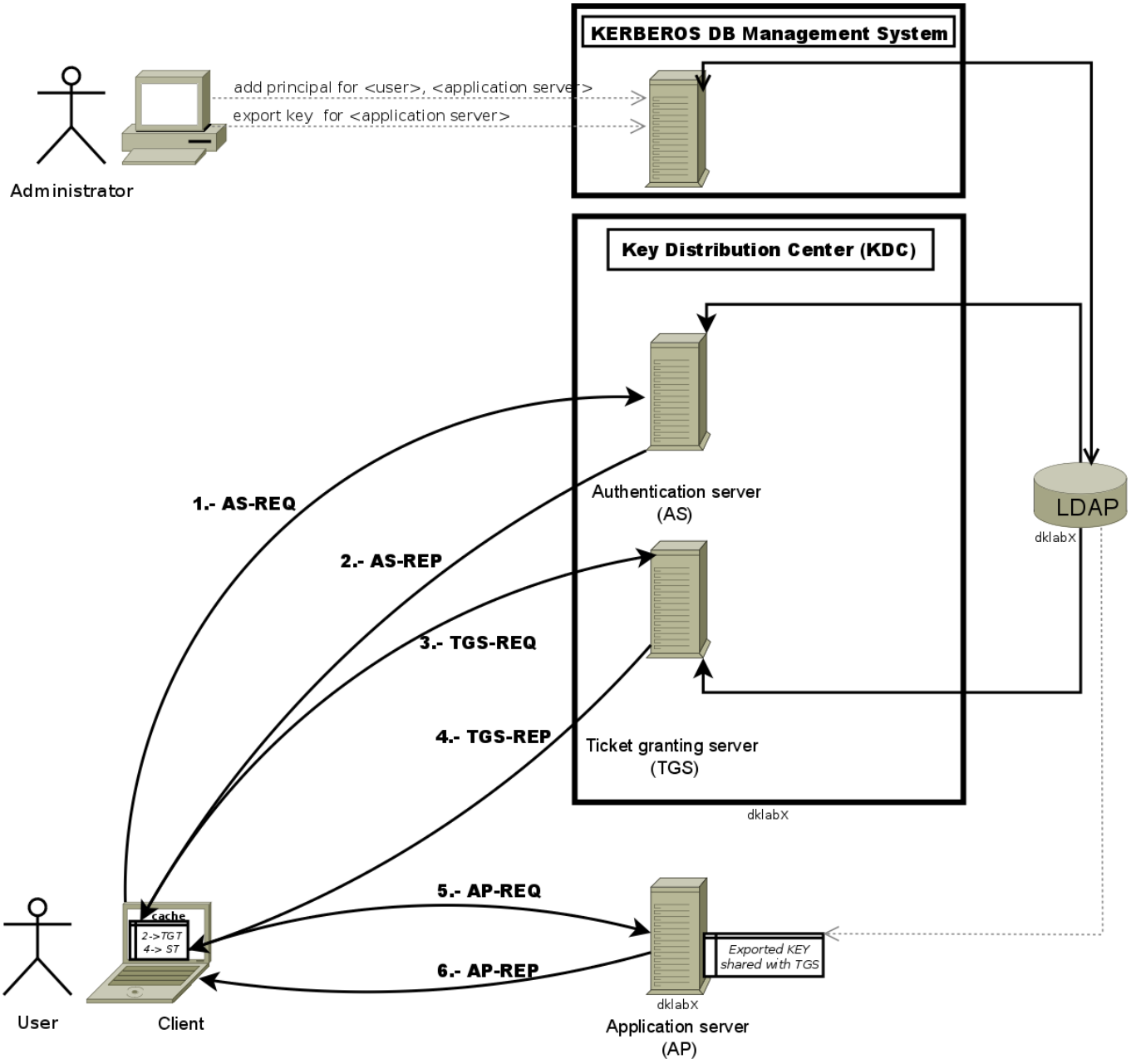
- Para facilitar las condiciones anteriores, debería implementarse como un protocolo específico, independiente, que se utilizase sólo durante la fase de autenticación.
- Idealmente, si la autenticación se produce exitosamente, el protocolo podría extenderse para permitir inmediatamente la negociación de un mecanismo y recursos para encriptar la comunicación posterior, proveyendo a ésta de confidencialidad e integridad si se considera necesaria.

KERBEROS es un protocolo de autenticación que cumple todos los requisitos anteriores. Si los hosts (clientes, servidores de KERBEROS, servidores de servicios finales) son seguros, sus resultados son fiables a pesar incluso de que la red subyacente sea considerada insegura.

KERBEROS se basa en el protocolo de Needham-Schroeder y utiliza criptografía simétrica y notaría, dos aspectos que conforman su arquitectura: efectivamente está compuesto por dos componentes. El primero es una base de datos y su correspondiente gestor. Almacena los principal KERBEROS, es decir el dato que refiere la identidad que se autentica pero, adicionalmente al principal, almacenan la llave criptográfica asociada a éste y cuyo valor depende de una contraseña. También guarda otra información asociada como fecha de caducidad de la contraseña, o del propio principal, etc. El segundo componente es el KDC ("Key Distribution Center") y su función es, utilizando la base de datos anterior, actuar como tercero de confianza expidiendo tickets a clientes y servidores que avalen la identidad de unos a los otros y viceversa.

El KDC se divide a su vez en dos subcomponentes: el AC ("Authentication Server") y el TGS ("Ticket Granting Server"). El AC es el servicio contra el que el usuario contacta cuando aún no está autenticado, e intenta demostrar su identidad en un proceso en el que interviene su contraseña. Si tiene éxito el AC le entrega un ticket especial TGT ("Ticket Granting Ticket") que se almacena en una caché de credenciales del lado del usuario y que queda a disposición para que los programas clientes de servicios finales lo puedan usar transparentemente al usuario. Efectivamente, cuando el usuario lance un software cliente de algún servicio final y la comunicación entre dicho cliente y su servidor avance hasta la fase de autenticación, el software cliente utilizará el TGT para, contactando ahora con el subcomponente TGS de KERBEROS, obtener un ticket ST ("Service Ticket"). A continuación, el cliente entrega este ST al servidor del servicio final para demostrar la identidad del usuario y satisfacer la fase de autenticación de ese servicio, y todo ello sin interrumpir al usuario. Gracias a la información almacenada en el ST por un lado, y por otro a que el servidor del servicio final también comparte una llave con el TGS, se demostrará también ante el cliente la identidad del servidor. Por tanto, autenticados cliente y servidor del servicio final entre sí, se dará paso al uso del servicio o incluso a la negociación de una capa de seguridad que ofrezca integridad y confidencialidad al resto de la comunicación. Además, el ST no se desecha sino que, en previsión de un uso futuro del servicio e independiente del actual, pasa a engrosar la caché de credenciales que había inaugurado el TGT. Esta situación se mantiene hasta que el TGT caduca, de forma que el usuario puede iniciar un proceso de renovación antes de esa fecha, o comenzar desde el principio en otro caso.

El siguiente gráfico ilustra el proceso de autenticación que acabamos de describir:



En la parte superior se añade un paso previo, la creación de los principal e información asociada por parte del administrador. Efectivamente con la línea gris punteada queremos expresar que son acciones que se dieron en algún punto del pasado. Ésto incluye también la exportación de la llave que comparten el TGS y el servidor del servicio final y su transmisión a éste último a través de un canal seguro.

El resto del diagrama corresponde a las comunicaciones con el AS (paso 1 y 2) y TGS (paso 3 y 4) junto a la expedición de los tickets TGT y ST y su almacenamiento en una caché de credenciales durante los pasos 2 y 4 respectivamente. Finalmente en horizontal, los pasos 5 y 6 representan la autenticación ante el servidor del servicio final gracias al ST.

El servidor también se autenticará ante el cliente gracias al ST y a su llave exportada[92].

MIT Kerberos[93] es la implementación de KERBEROS seleccionada. Corresponde a la implementación original, es la más avanzada y está bien soportada por las aplicaciones (a este respecto, diremos que la fase AP_REQ - AP_REP no es estándar sino que es flexible y varía según la aplicación). El gestor de la base de datos en MIT Kerberos es llamado "kadmind" mientras que "kadmin" implementa el lado del cliente. Además existe "kadmin.local" que permite hacer modificaciones offline. Por otro lado, el KDC se implementa como un sólo programa llamado "krb5kdc" y, efectivamente, implementa tanto al AS como al TGS. Los dos componentes anteriores admiten tener a OpenLDAP como backend de almacenamiento, si bien krb5kdc sólo puede usarlo en modo lectura al no ser el gestor.

Estamos ya en condiciones de especificar cómo se desplegará MIT Kerberos en nuestra infraestructura: como viene siendo habitual, kadmind y krb5kdc estarán instanciados tanto en dklab1 como en dklab2. En la siguiente sección discutiremos la forma que tendrán los principals.

Política de nombres asociada a KERBEROS

Cada principal sigue la estructura <primary>/<instance>@<REALM>. En este apartado discutiremos la política de valores que recibe cada parte en nuestra infraestructura, y qué principals serán necesarios.

KERBEROS permite discriminar varios grupos de autenticación independientes total o parcialmente. Para ello el principal se sufija con "@<REALM>", donde <REALM>suele corresponder al nombre del dominio DNS en mayúsculas. Así pues, en nuestro despliegue:

Dominio DNS	REALM KERBEROS
casafx.dyndns.org	CASAFX.DYNDNS.ORG

Por su lado, <primary>/<instance>representan la identidad en ese reino de autenticación. La parte "/<instance>" es opcional, pero si se utiliza permite emplear <instance>como un metadato susceptible de ser empleado en labores de autorización. En el caso de principals que identifican a humanos, <instance>no se utiliza para usuarios comunes mientras que para administradores sí, por ejemplo permite interpretar el tipo de permisos que disfruta ese administrador en el gestor de la base de datos KERBEROS.

Por otro lado, en el caso de principals que identifican autómatas como, por ejemplo, un servidor de un servicio final, <instance>suele aparecer y corresponderse con el FQDN del host que alberga esa instancia del servidor; además <primary>corresponde a un nombre que por convenio represente el servicio, normalmente el nombre del protocolo en minúsculas.

[92]Puede consultarse el proceso con mayor detalle en:
http://kerberos.org/software/tutorial.html#1.4
La especificación completa y sus implicaciones pueden consultarse en los papers referenciados en:
http://web.mit.edu/kerberos/papers.html
[93]http://web.mit.edu/kerberos/

Los siguientes principal revelan las identidades KERBEROS de administradores, usuarios y servidores en nuestra infraestructura. Comenzamos con los de administrador:

Principal de administrador
root/admin@CASAFX.DYNDNS.ORG

Como sabemos desde nuestra exposición sobre LDAP, la identidad anterior será mapeada en cada sistema a la identidad, en la sintaxis de dicho sistema, autorizada para realizar cualquier tipo de labor administrativa.

Pasamos ya a los principal de usuario:

Principal de usuario
umea@CASAFX.DYNDNS.ORG
aemu@CASAFX.DYNDNS.ORG

De forma parecida, las dos identidades anteriores serán mapeadas a las identidades con que el usuario sea representado en los servicios finales, así que como resultado el usuario disfrute de una única identidad, su cuenta.

Pasamos ya a los principal de los servidores:

Principal de servidor
ldap/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
ldap/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG
postgres/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
postgres/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG
bucardo/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
bucardo/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG
xmpp/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
xmpp/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG
jabberd2-sm/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
jabberd2-sm/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG
smtp/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
smtp/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG
imap/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
imap/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG
sieve/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
sieve/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG
host/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
host/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG
afs/casafx.dyndns.org@CASAFX.DYNDNS.ORG

Algunos de los anteriores <primary>precisan algún comentario adicional. El valor "host" se utiliza en servicios de log-in remoto tal como SSH. Por su lado el valor "jabberd2-sm" se utiliza para que el componente sm ("Session Manager") de Jabberd2 (nuestra implementación escogida para servidor XMPP) pueda autenticarse usando KERBEROS ante PostgreSQL (nuestra implementación de SQL RDBMS). De una forma parecida, el <primary>de valor "bucardo" se utiliza para que (en este caso un software específico para la replicación del modelo de datos tras SQL) pueda autenticarse usando KERBEROS ante el SQL RDBMS. Éste último servicio, PostgreSQL, debe tener necesariamente otro principal puesto que las dos entidades anteriores pretenden utilizar KERBEROS como mecanismo de autenticación; dicho principal es, evidentemente, el que tiene por <primary>a "postgres".

El último principal, "afs/casafx.dyndns.org@CASAFX.DYNDNS.ORG" corresponde al usado en AFS, y el <instance>no corresponde en este caso a un nombre de host sino al de la célula AFS (el nombre de la unidad administrativa en ese sistema, y que normalmente corresponde al nombre del dominio en minúsculas).

Estrategia de uso de KERBEROS entre servidores

En la sección anterior mencionábamos que un componente del servicio XMPP necesitaba autenticarse ante el SQL RDBMS, y era KERBEROS el mecanismo utilizado. Ésto es algo generalizado.

Efectivamente, nuestro despliegue incluye una infraestructura de servicios base que los servicios finales utilizan. Entonces, KERBEROS es el protocolo que intermedia, además de la autenticación entre usuarios y servicios finales, las autenticaciones en esas comunicaciones entre servidores.

Sin embargo ésto plantea dos dificultades que debemos resolver:

- El protocolo contemplaba una fase interactiva en la que el AS expedía un TGT previa demostración por el cliente de conocer la contraseña del principal requerido.
- El TGT caduca. Ésto implica un problema de automatización, en este caso para renovar el TGT antes de que sea inválido.

El primer problema se resuelve exportando la llave. Efectivamente sabíamos que el software que actúa como servidor en una comunicación KERBEROS no tiene una fase interactiva sino que directamente almacena en su máquina la llave que comparte con el TGS. Del lado del cliente KERBEROS ésto no era así: por razones de seguridad ese almacenamiento se evita y en su lugar se obliga al usuario a memorizar una contraseña. Sin embargo, si ello no es posible como ahora, podemos hacer caso omiso a esa restricción. Entonces, almacenaremos la llave también para el rol de cliente, y ésta debería ser utilizada para conseguir el TGT.

Al omitir el prompt para la contraseña, nos ahorramos el problema de la interactividad pero lo sustituímos por otro, en concreto por otro problema de automatización. Así pues

tenemos que automatizar la expedición del TGT a partir del keytab[94], y, como vimos, su periódica renovación.

Llamaremos "monitorizador del TGT" al programa que vele por la existencia y validez del TGT y, eventualmente, lo gestione también. Una implementación de este monitorizador sería la combinación del software "k5start"[95] e "init"[96]. Por un lado k5start supervisa el TGT, por otro init supervisa que k5start esté cargado en memoria y operativo (que no se encuentre en estado zombie, por ejemplo). Adicionalmente, k5start puede facilitar otras automatizaciones, como la de utilizar el TGT para conseguir un ticket de servicio para AFS y, como es forzoso en ese sistema para completar la autenticación, obtener un token AFS.

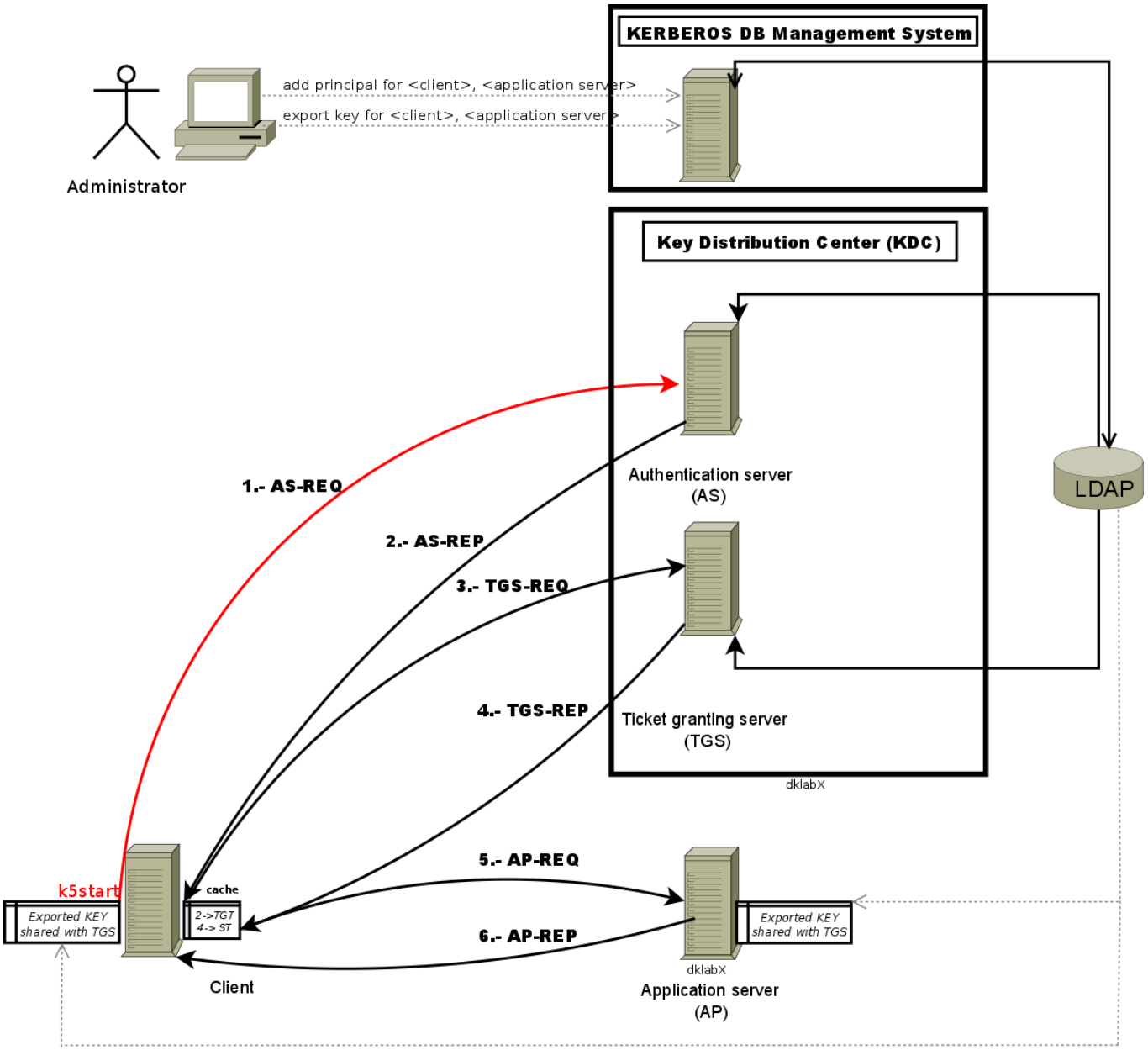
El siguiente diagrama muestra cómo se modifica el proceso de autenticación cuando el rol de cliente KERBEROS es desarrollado por un autómata tal como un servidor:

[94]Es decir, del fichero que almacene la llave.

[95]http://www.eyrie.org/~eagle/software/kstart/

[96]Nos referimos efectivamente al típico proceso init que referencia POSIX y que es parte integrante de cualquier instalación de Debian en particular.

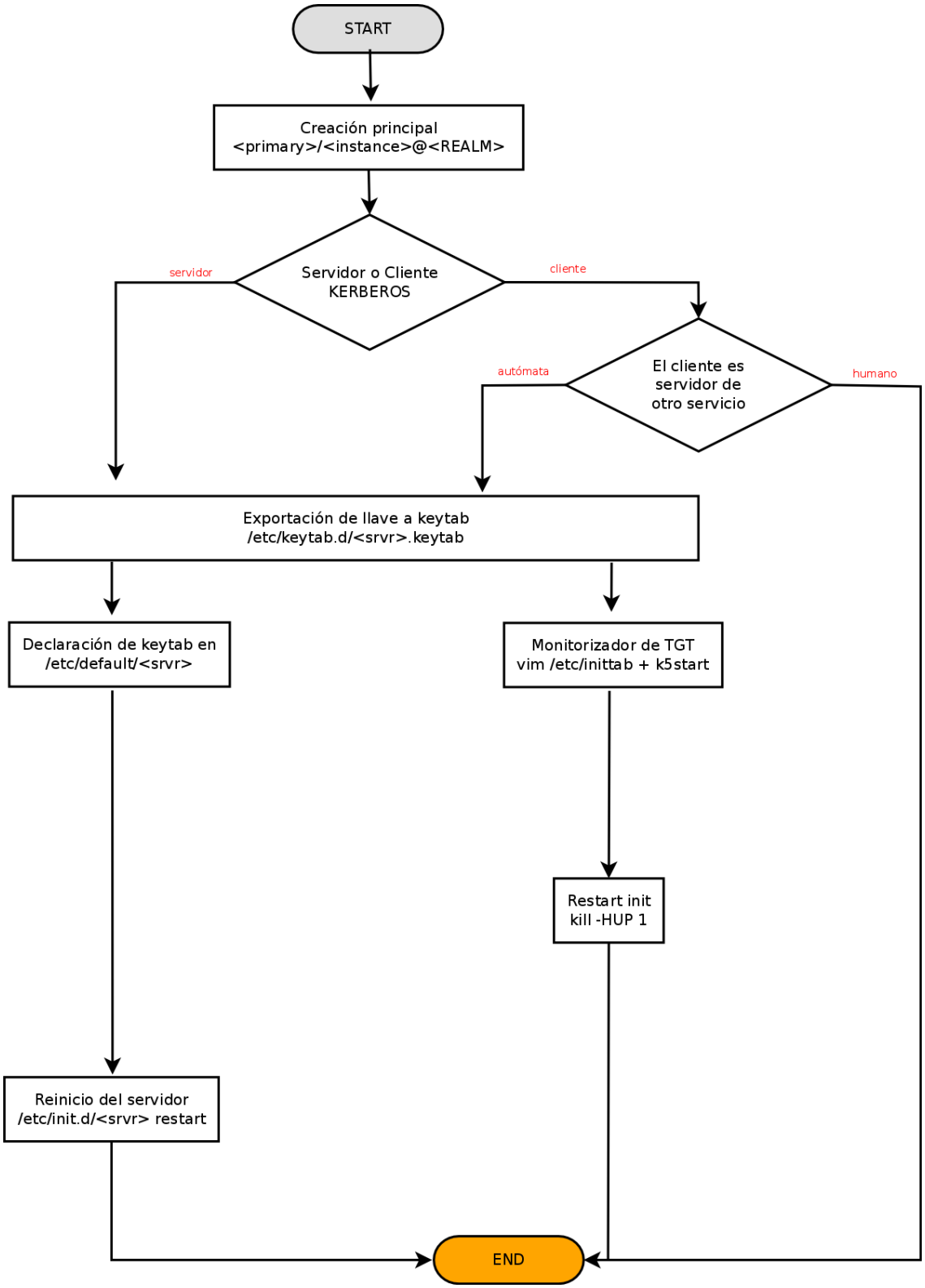
http://arxiv.org/abs/0706.2748



Lo fundamental es, efectivamente, que la llave del cliente KERBEROS también se exporta, y que el proceso k5start se encarga de automatizar la obtención y renovación del TGT. En el diagrama, además, es notoria la ausencia de usuario alguno en esa máquina.

Flujo de trabajo del administrador en relación a KERBEROS

En los dos diagramas que hemos mostrado para representar la comunicación KERBEROS, mencionábamos que, con anterioridad, el administrador del sistema debería crear los recursos necesarios (principal). Ahora añadimos que posiblemente sea además necesario realizar modificaciones en algún fichero. El siguiente diagrama de flujo muestra las acciones que tomar en cada caso:



Efectivamente el diagrama comienza con una acción necesaria en todos los casos: la creación del principal (a través de una conexión con kadmind, por ejemplo).

Si el principal era para una entidad que jugará el rol de servidor de servicio en el protocolo KERBEROS (es decir, queremos kerberizar este servicio), entonces debemos exportar su llave en un fichero (llamado genéricamente keytab) y, a continuación, debemos anunciar a ese servidor dónde está aquel. Para ello podemos declarar la variable KRB5_KTNAME con valor la ruta al keytab en el fichero de configuración del script de inicio del servidor, /etc/default/<srvr>.

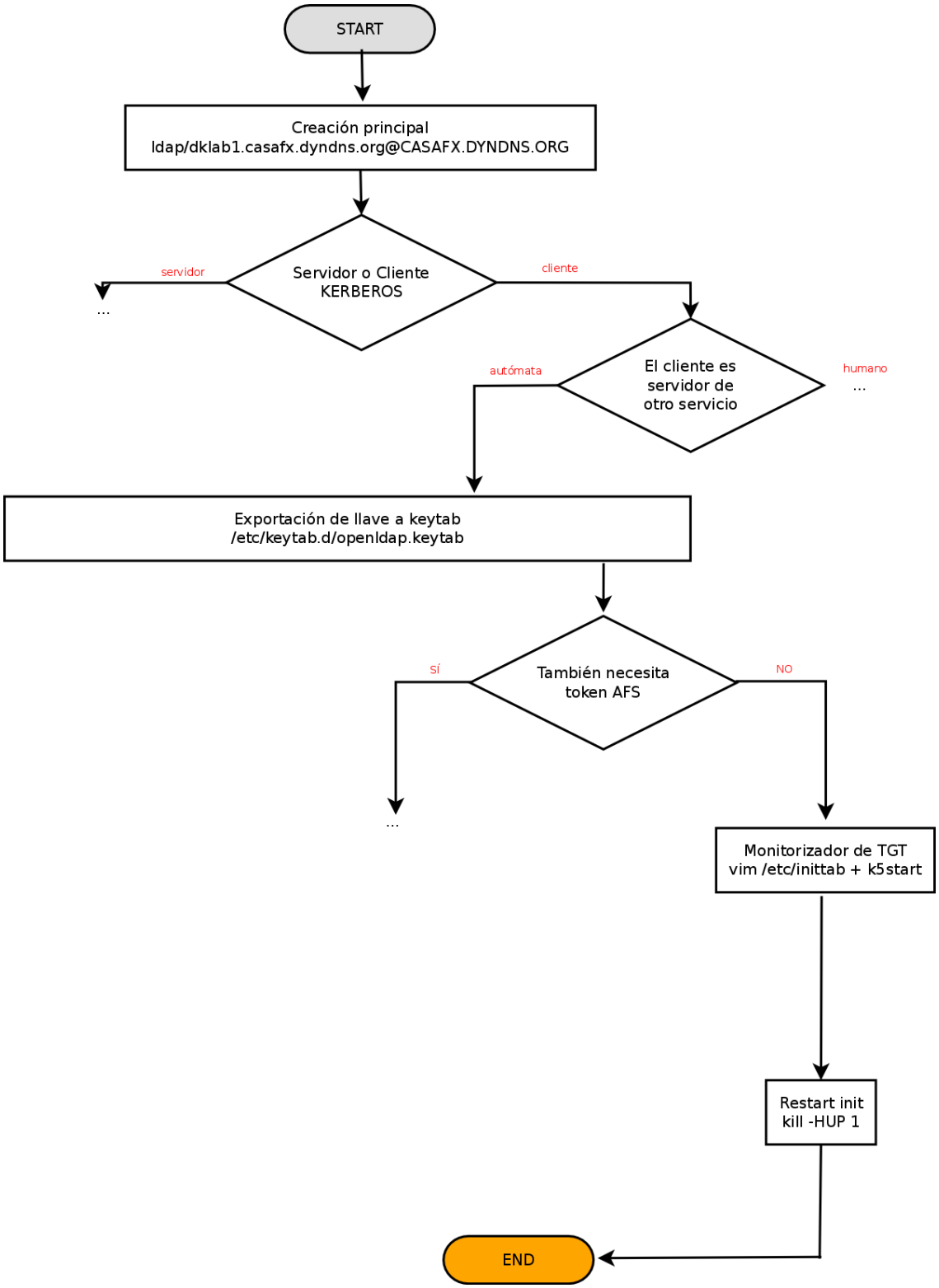
Si el principal era para una entidad que jugará el rol de cliente KERBEROS y corresponde a una persona, hemos acabado (si bien esa persona debería conocer y memorizar la contraseña utilizada en el proceso).

Si el principal era para una entidad que jugará el rol de cliente KERBEROS nuevamente, pero corresponde a un autómata tal como un servidor de algún servicio que necesita autenticarse ante cualquier otro, entonces debemos exportar su llave en un keytab como primer paso. Como segundo paso debemos modificar a init (editando /etc/inittab en Debian 6) para que lance y monitorice a k5start, el cual recibirá como argumento entre otros el keytab. Entonces hacemos que init relea su configuración mandándole la señal HUP. A partir de ése momento, cuando el autómata necesite autenticarse como decíamos, encontrará que su caché de credenciales no está vacía, sino que contiene el TGT, obtenido y renovado puntualmente por k5start.

Anotamos que, en el capítulo dedicado a AFS, se añadirán dos subcasos para la situación que acabamos de explicar (cliente KERBEROS autómata) que distingan si el TGT es para obtener un token AFS (porque el autómata lo que necesita es simplemente acceder a AFS) o no (es otro servicio kerberizado distinto de AFS al que pretende acceder).

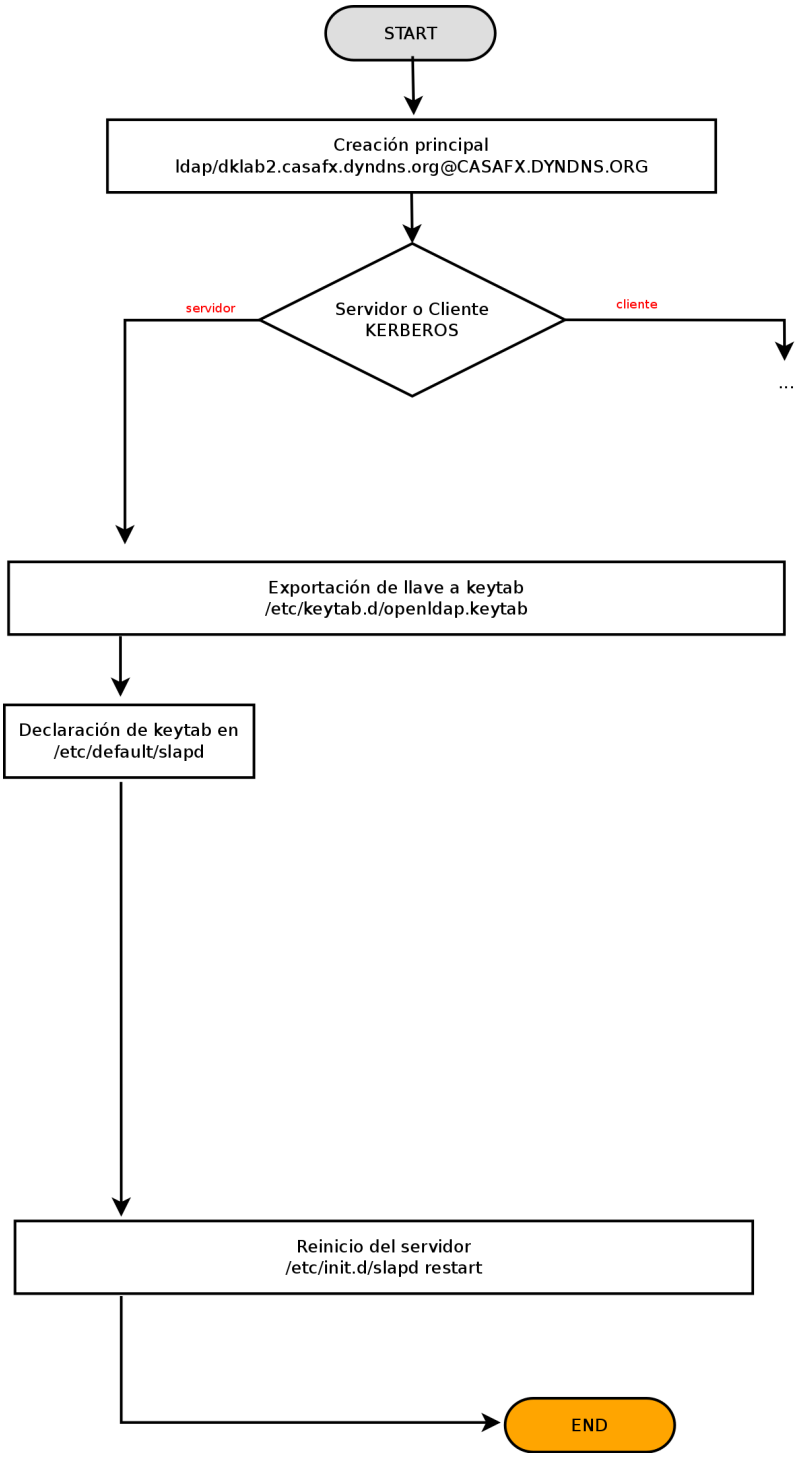
Kerberización de un servicio: replicación LDAP

Acorde al flujo de trabajo descrito, los recursos que crear y las decisiones que tomar son:



Ésto asegura que, durante la replicación, slapd en dklab1 pueda autenticarse con el rol de cliente de KERBEROS en dklab2. Recíprocamente, aún tenemos por tanto que kerberizar slapd en dklab2 para que éste pueda actuar con el rol de servidor KERBEROS

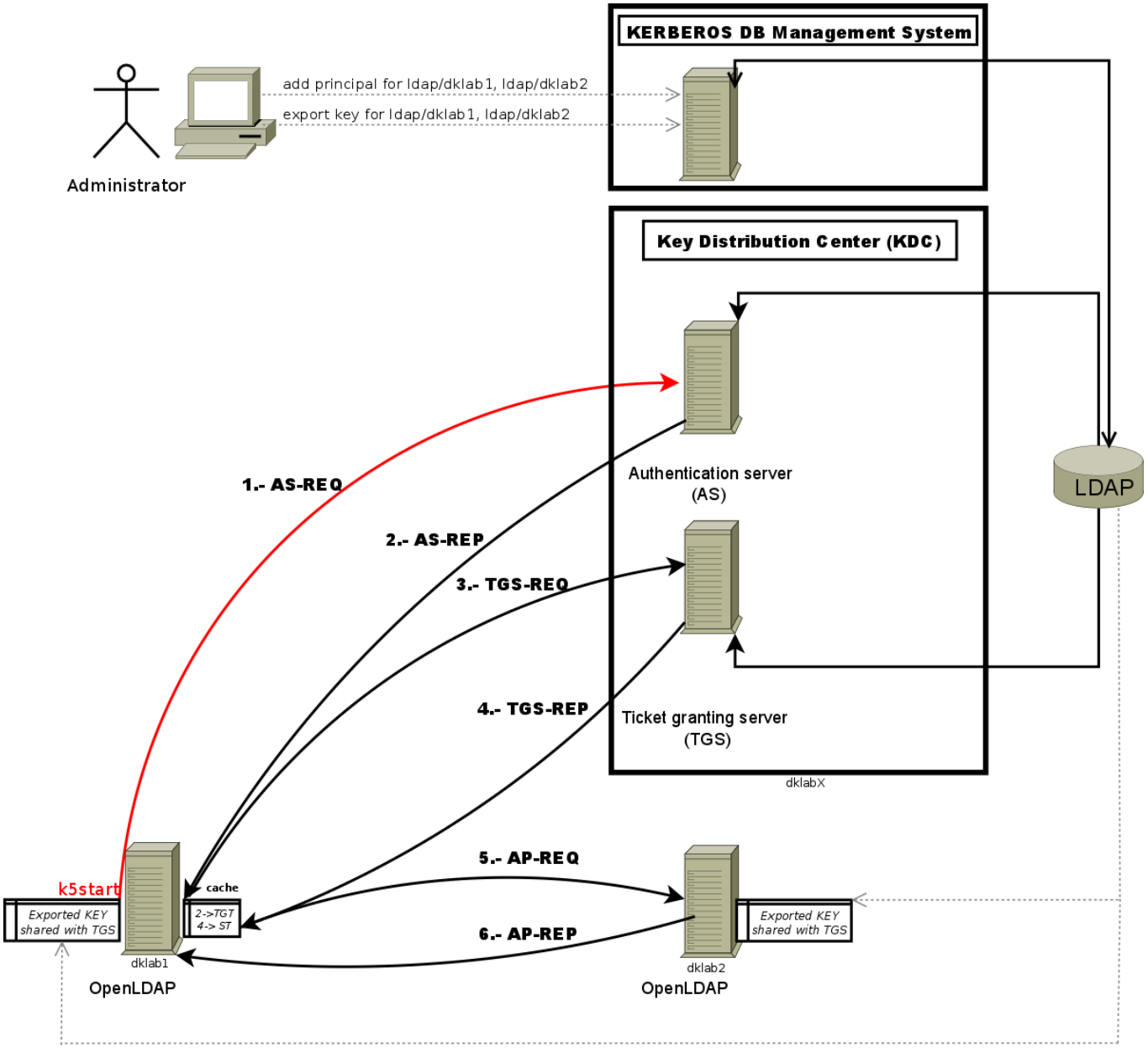
durante la autenticación.



Similar proceder se ha de efectuar en dklab1 y dklab2 para invertir los papeles, sólo que los principal ya están creados y exportados.

Para que la librería libsasl sepa que ha de utilizar SASL-GSSAPI y no otro mecanismo de autenticación, se puede explicitar éste en su fichero de configuración (/etc/ldap/ldap.conf).

Ésto es suficiente para asegurar la replicación que configuramos para LDAP. La fase de autenticación se producirá, tal como se expuso genéricamente cuando comentamos el despliegue de MIT Kerberos:



1.3.9. SQL

Anexo:	5

En el apartado anterior dedicado a LDAP se expuso y justificó la presencia de un gestor de bases de datos relacionales. Específicamente se comentó que sería Jabberd2 su

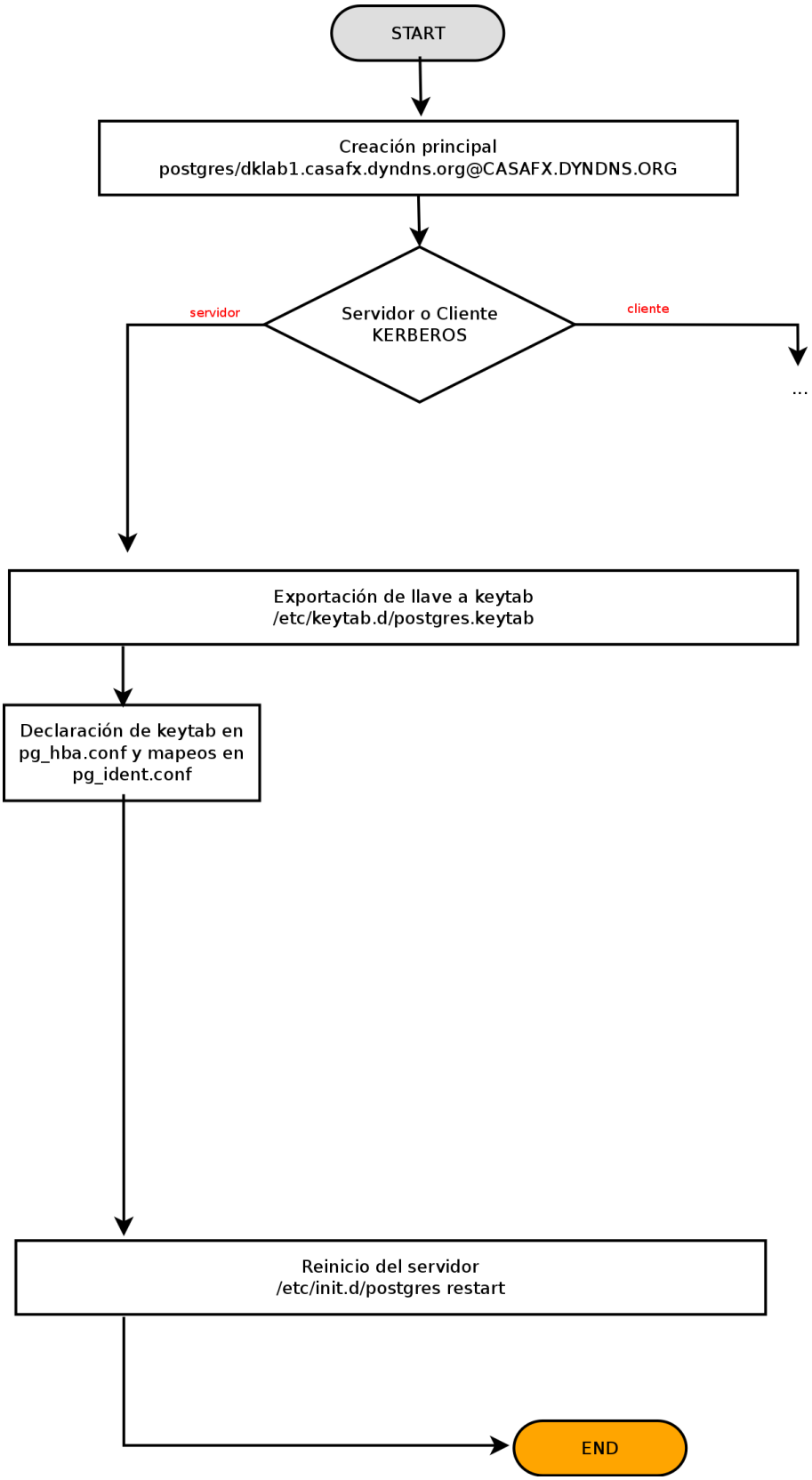
único usuario, en concreto para un subconjunto especialmente volátil de sus metadatos.

No vamos a extendernos mucho más en éste. La implementación escogida fue PostgreSQL[97] porque, efectivamente es el SQL RDBMS más avanzado dentro del Software Libre y soporta nuestro esquema de autenticación KERBEROS.

PostgreSQL ofrece desde hace poco un sistema nativo de replicación síncrona pero, sin embargo, de topología Master-Slave. La implementación de esta funcionalidad es una tarea prioritaria para el proyecto pero aún no está conseguido. Es por ello que instalamos un software adicional, Bucardo, que permite hacer una replicación asíncrona Master-Master. Una discusión sobre la idoneidad de Bucardo para nuestra situación concreta (sólo Jabberd2 usa PostgreSQL) puede encontrarse en la sección 1.3 del anexo. Bucardo, ciertamente, no es de nuestro interés, pero con una implementación nativa Multi-Master a la vuelta de la esquina, lo usaremos temporalmente puesto que es sencillo de desplegar. Todos los detalles al respecto en la sección 1.3.1 y 1.3.2.

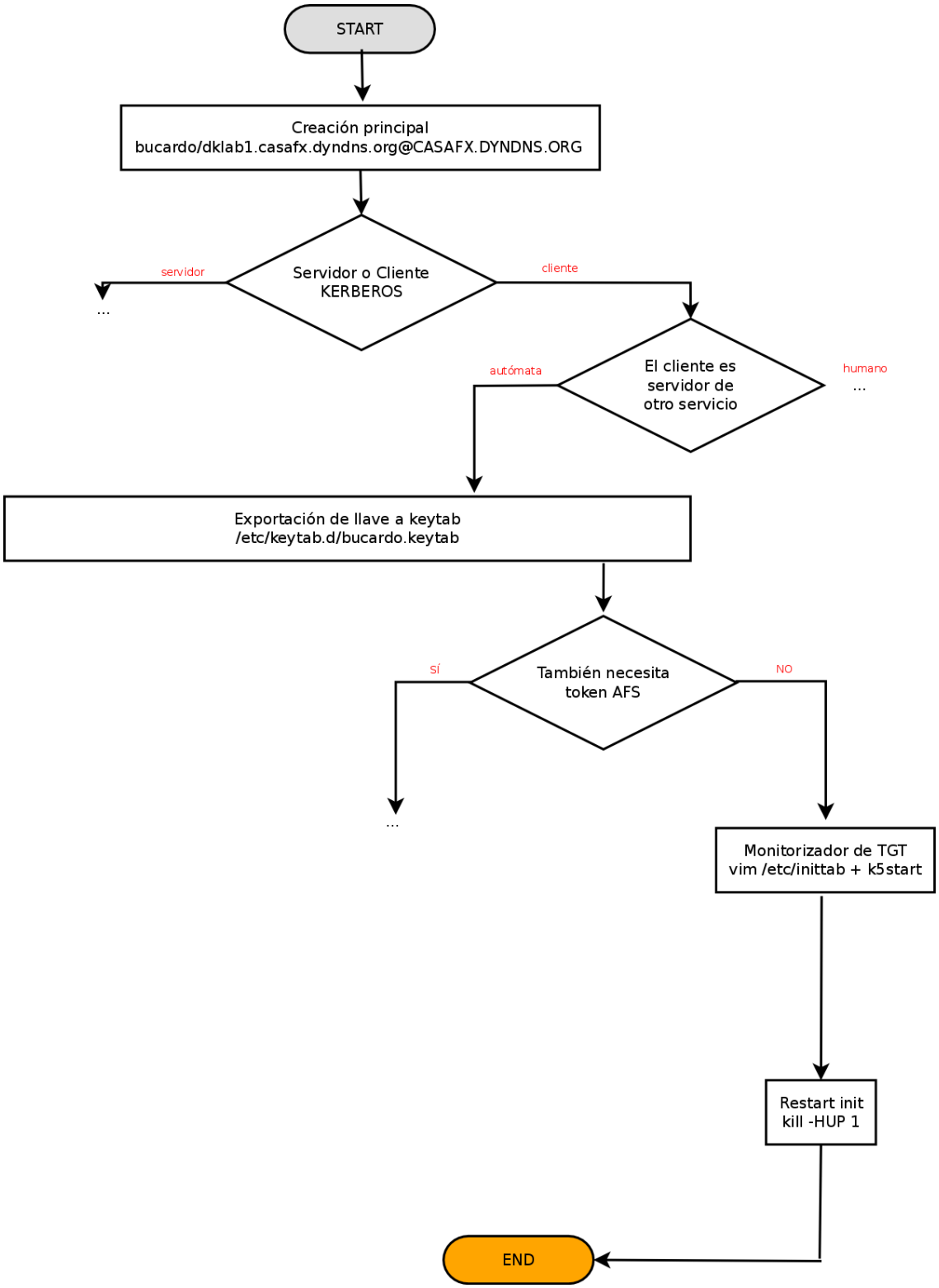
Sí es de nuestro interés detallar cómo kerberizar el servicio. Haciendo uso de nuestro diagrama para el flujo de trabajo del administrador:

[97] http://www.postgresql.org/


START
Creación principal
postgres/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
Servidor o Cliente
KERBEROS
servidor
cliente
...
Exportación de llave a keytab
/etc/keytab.d/postgres.keytab
Declaración de keytab en
pg_hba.conf y mapeos en
pg_ident.conf
Reinicio del servidor
/etc/init.d/postgres restart
END

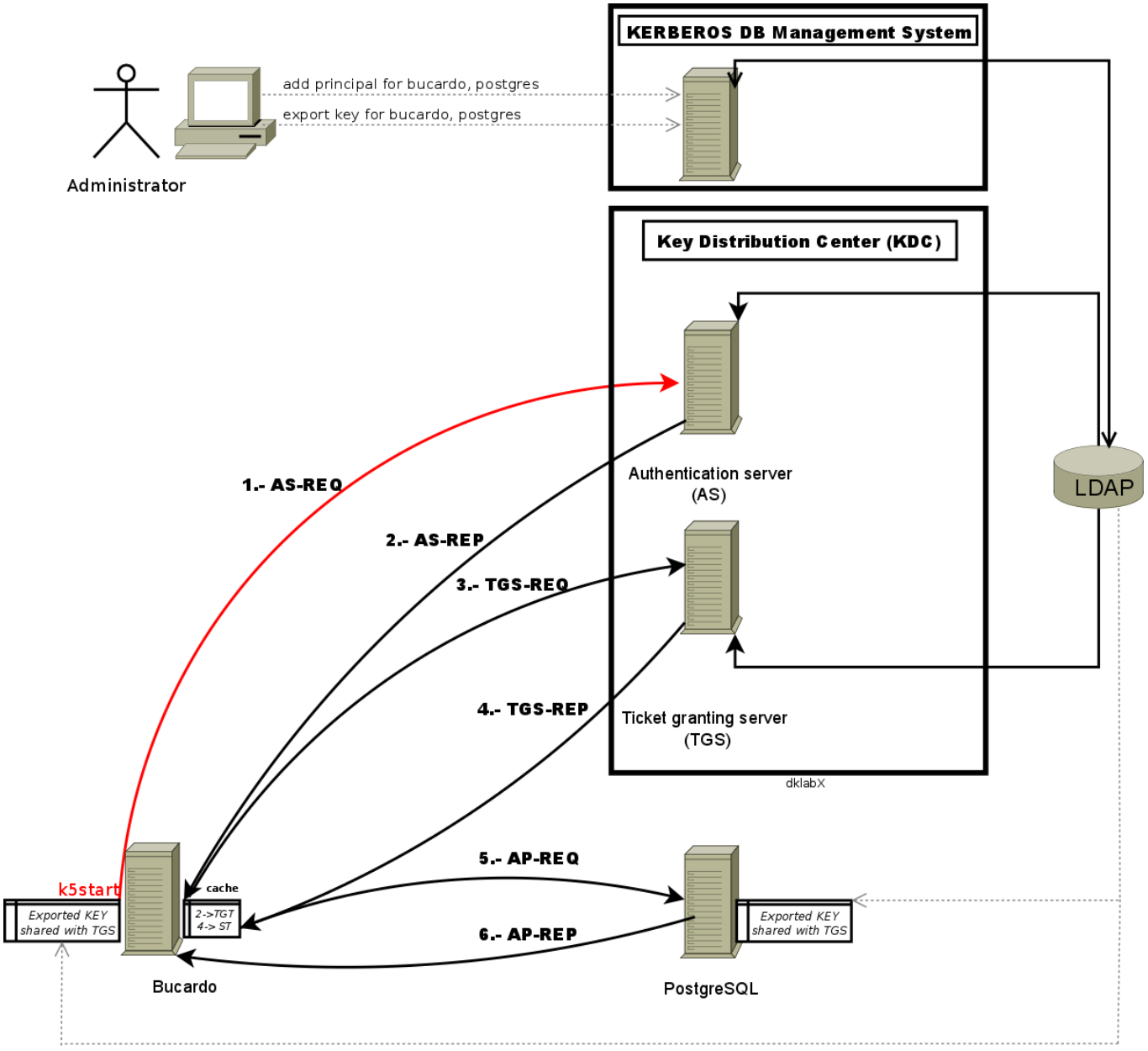
Ésto permite a Postgres actuar como servidor kerberizado. Al contrario que en otras ocasiones, destaca el hecho de que PostgreSQL dispone de una variable de configuración en su fichero "pg_hba.conf" para indicar el keytab. Además en otro fichero, "pg_ident.conf" se hacen explícitos los mapeos de, por ejemplo, los principal de Jabberd2 a sus roles de PostgreSQL. U, otro ejemplo de mapeo, permite que Bucardo se conecte autenticado gracias a KERBEROS.

Efectivamente hemos de dar soporte a Bucardo para actuar como cliente KERBEROS. Suponiendo que ya hayamos editado pg_ident.conf para mapear su rol de PostgreSQL a su futuro principal:



Ni Bucardo ni Jabberd2 pueden utilizar registros SRV para descubrir a los servidores PostgreSQL porque éstos no existen. En cualquier caso, el administrador configurará a mano la localización de PostgreSQL y el proceso de autenticación será, como se discutión

genéricamente en la sección dedicada a MIT Kerberos:



1.3.10. AFS

Anexo:	6

Nuestra infraestructura tiene necesidad de almacenar recursos diversos de tamaño muy variable y sobre los que poder efectuar acciones tales como creación, lectura, escritura, traslado, enlazado o agrupamiento. Dichos recursos corresponden a los gestionados por los usuarios finales desde el servicio de Shell, pero también pueden corresponder a otros como

los gestionados por el sistema de correo, es decir los mensajes de correo electrónico[98]. El tipo de sistema de almacenamiento que se adapta a estas premisas iniciales es el de un sistema de ficheros ("FS", del inglés File System).

Nuestro FS debiera cumplir las siguientes especificaciones:

- Operar en red
- Con espacio de nombres único, común a todos los clientes.
- Multiplataforma. En relación a ésto, sería deseable que pudiese automáticamente mapear directorios para binarios adaptados a la arquitectura de microprocesador y SO del cliente.
- Que pueda usar tickets TGT KERBEROS como mecanismo de autenticación; mapeos.
- Excelente historial de fiabilidad y escalabilidad.
- Adaptado a un patrón de uso en el que predominan las lecturas.

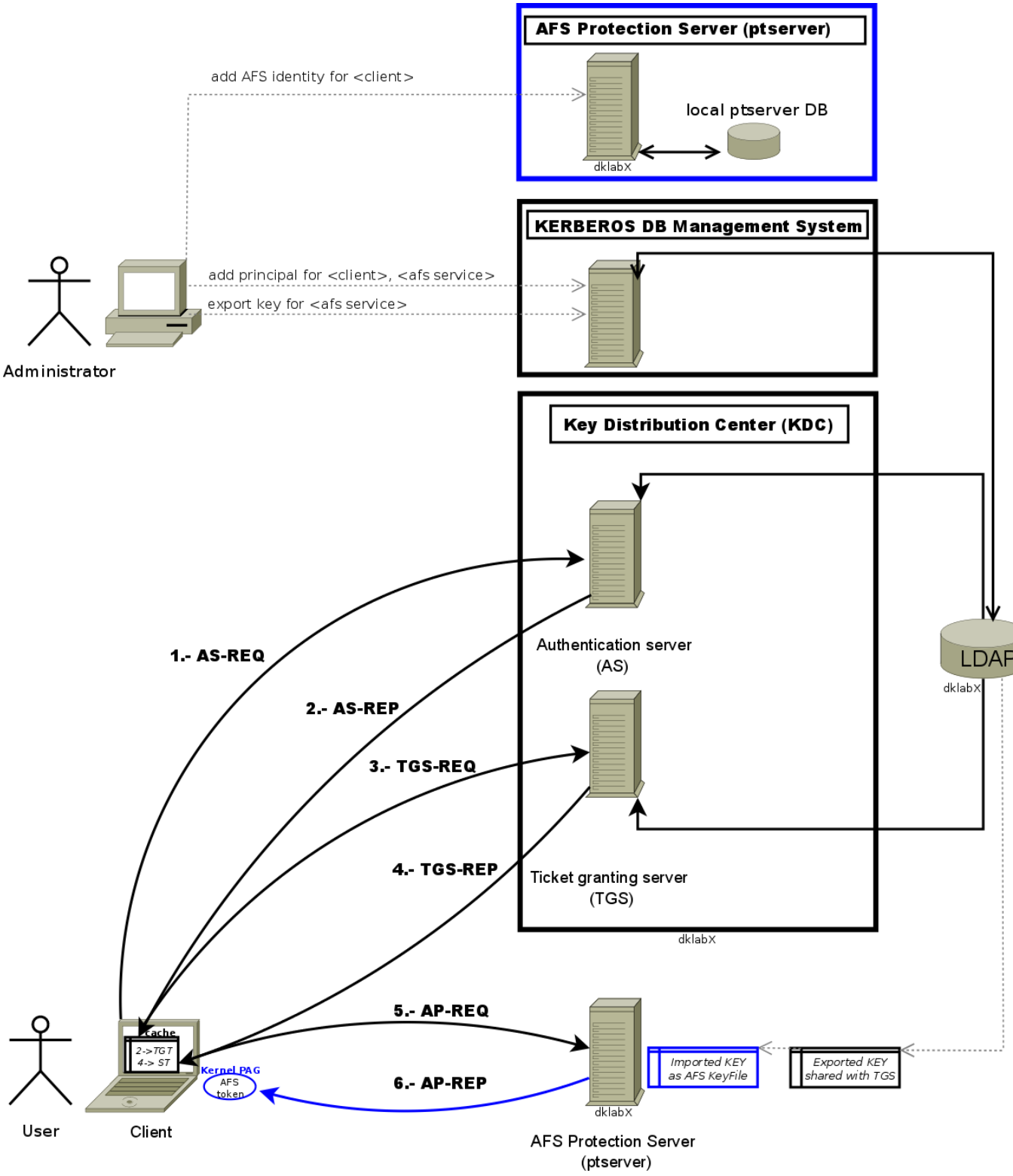
Efectivamente, AFS es un reputado sistema de ficheros que cumple con los anteriores requisitos, entre otros[99].

OpenAFS[100] constituye la implementación de AFS en nuestra infraestructura (y la única implementación Software Libre - al menos completa, ARLA[101] no lo está). Puede considerarse como servicio base (caso del correo), aunque normalmente lo usamos como un servicio final más. Es por ello que vamos a centrarnos en su funcionamiento como servicio y en algunas peculiaridades que presenta la autenticación.

El apartado 1.1 del anexo detalla brevemente su arquitectura de componentes. El resto del anexo da buena cuenta de todos los detalles para desplegar todos sus componentes tanto en dklab1 como dklab2. El software es ciertamente complejo pero también robusto. Añadiremos que tiene su propio servicio de replicación Multi-Master.

Como comentábamos, veamos cómo modifica el esquema de autenticación KERBEROS:

[98] http://tools.ietf.org/html/rfc5322
[99] http://blog.endpoint.com/2009/01/why-not-openafs.html
[100] http://www.openafs.org/
[101] http://www.stacken.kth.se/project/arla/



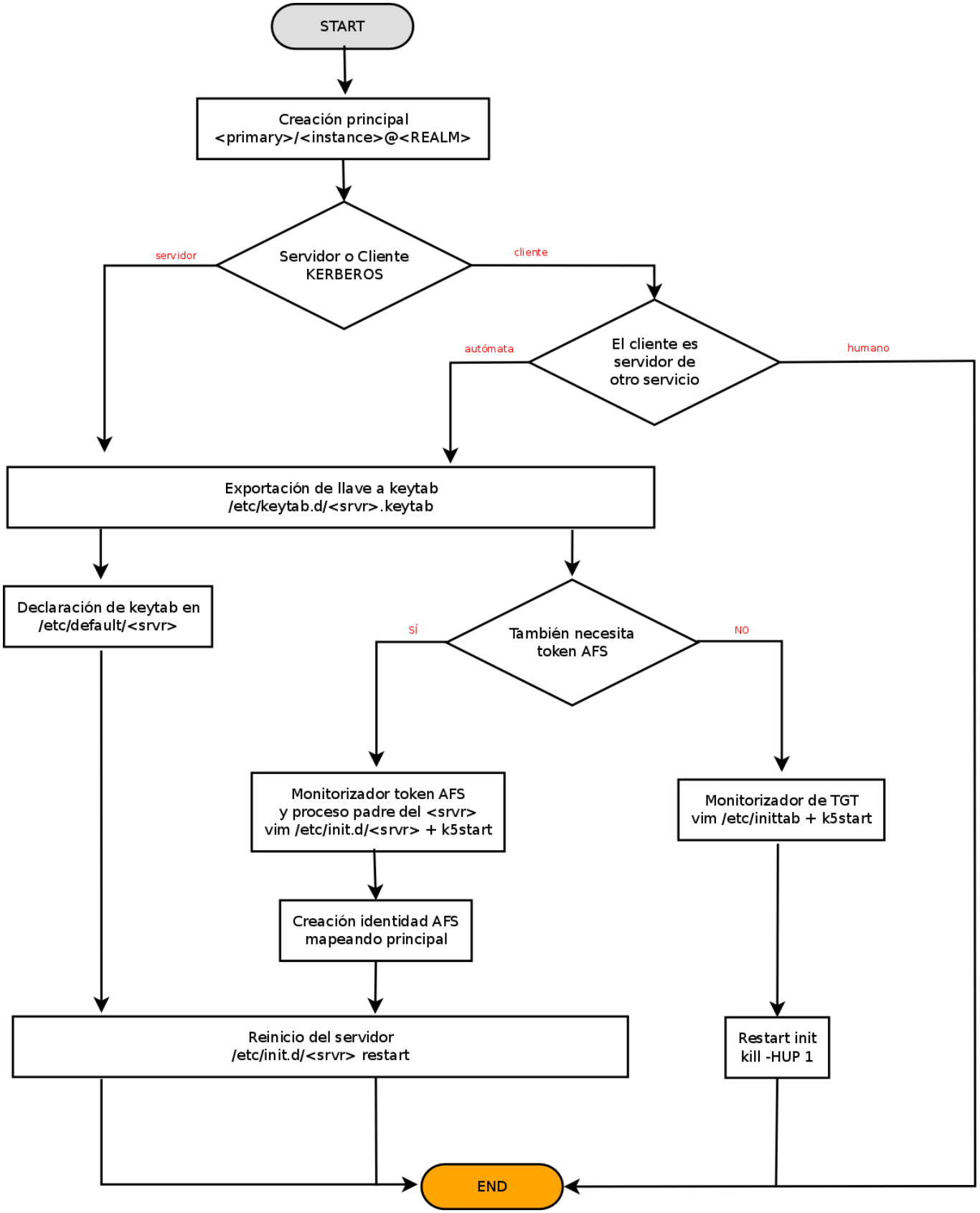
Efectivamente, tres son las principales particularidades cuando el servicio ante el que se autentica el usuario es AFS. Utilizamos el color azul para marcarlas. La primera es,

como se ve en la parte superior, la necesidad de crear una identidad en el sistema AFS, ésto es adicional a la creación del principal KERBEROS y, además, dependiente de ello puesto que el principal ha de poder mapearse a la identidad AFS.

En segundo lugar, el componente kerberizado que autentica en AFS al cliente es pt-server (el servidor de la "Protection DB" vinculada con las identidades AFS y las ACL del sistema de ficheros). Ocurre que ptserver no usa directamente el keytab donde está exportada la llave que comparte con el el KDC, sino que el administrador ha de importar ésta en un formato KeyFile apto para ptserver.

Por último, cuando el cliente se autentica en el ptserver, éste le entrega en su respuesta una credencial, el token AFS. La forma de la caché para el token en el diagrama es ahora elíptica porque se almacena en memoria, al contrario que la caché de credenciales KERBEROS que es de forma rectangular porque se almacena en el sistema de ficheros. Con el token AFS, la aplicación cliente puede ya usar a AFS, como se mostrará más adelante.

Veamos ahora cómo se modifica el flujo de trabajo del administrador cuando AFS se utiliza como servicio base, y los servidores (autómatas) necesitan acceder a AFS:

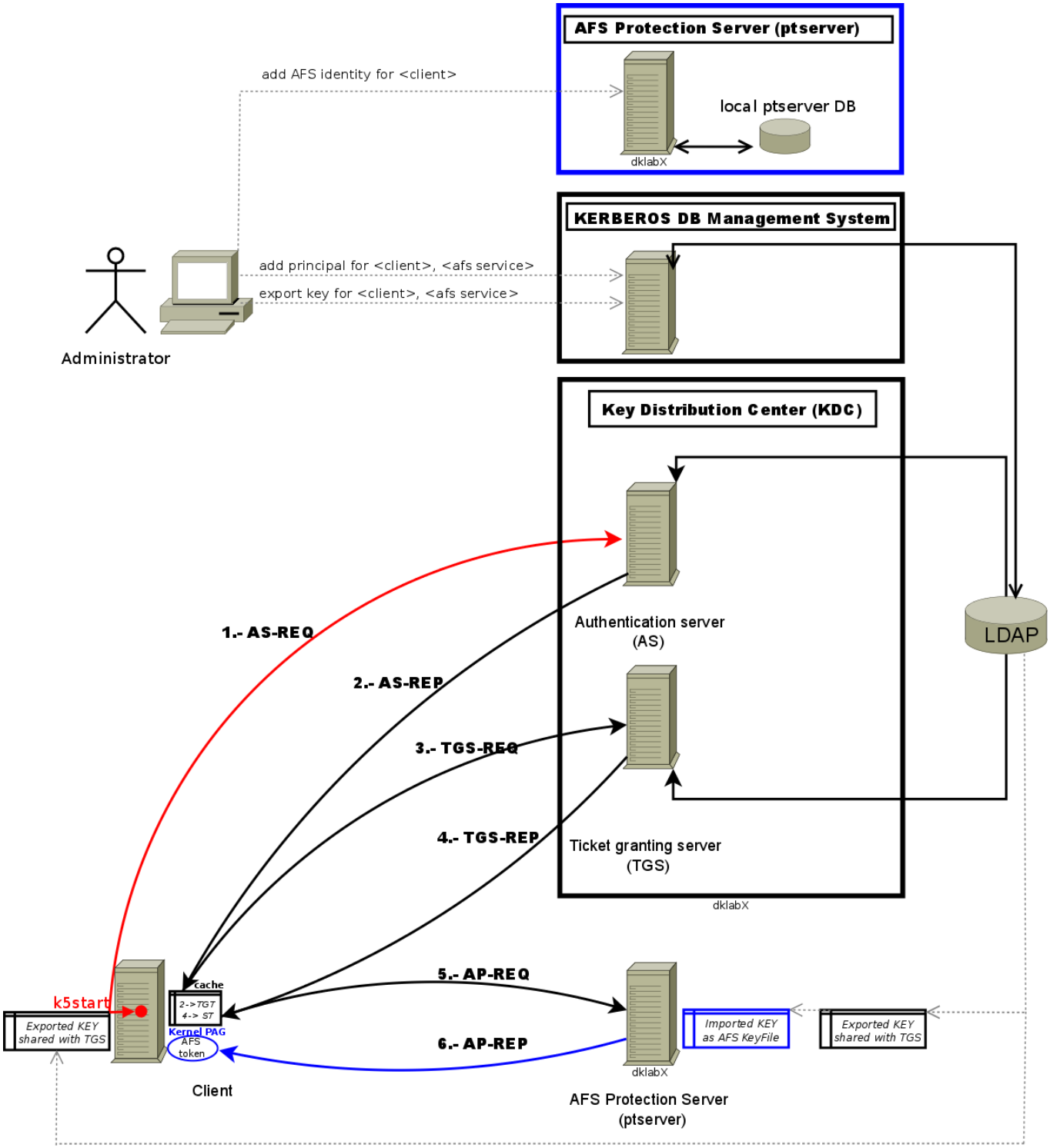

START
Creación principal
<primary>/<instance>@<REALM>
Servidor o Cliente
KERBEROS
servidor
cliente
El cliente es
servidor de
otro servicio
autómata
humano
Exportación de llave a keytab
/etc/keytab.d/<srvr>.keytab
Declaración de keytab en
/etc/default/<srvr>
También necesita
token AFS
SÍ
NO
Monitorizador token AFS
y proceso padre del <srvr>
vim /etc/init.d/<srvr> + k5start
Monitorizador de TGT
vim /etc/inittab + k5start
Creación identidad AFS
mapeando principal
Reinicio del servidor
/etc/init.d/<srvr> restart
Restart init
kill -HUP 1
END

Ceñiéndonos a lo novedoso, la rama final situada en segundo lugar por la izquierda, viene a indicar que cuando un autómata necesita el TGT simplemente para poder autenticarse en el servicio AFS, obtener el token AFS y usar el sistema de ficheros, el esquema de monitorizar el TGT cambia. La razón es que el modelo de diseño para la caché de credenciales en AFS difiere sensiblemente de la de KERBEROS. En este último, cualquier proceso puede acceder a la caché si puede relacionarse (leer) con una parte del árbol de ficheros (es decir, un fichero - normalmente /tmp/krb5cc_<uid>- donde se localiza la caché de credenciales. En el caso de AFS, un proceso sólo puede acceder a la caché si puede relacionarse con una parte del árbol de procesos, no de ficheros. Efectivamente sólo los procesos hijos del aquel que obtuviese el token (y ése mismo) pueden usar el token AFS. Ésto se relaciona con que el token no se almacena en el sistema de ficheros sino que el cache manager (el módulo openafs.ko del kernel) lo almacena en memoria, concretamente en una estructura PAG (de "Process Authentication Group"[102]). Por tanto, un nuevo esquema de monitorización de credenciales y lanzamiento del servidor es necesario.

Una posibilidad es, como señala el diagrama, modificar el script de inicio (/etc/init.d/<srvr>) para que lance a k5start, éste consiga el TGT (a través de la localización del keytab, que ha de pasársele con el flab "-f"), a continuación registre una PAG y obtenga el token AFS, y por último lance al servidor como proceso hijo, asegurándole a éste y a sus otros posibles hijos el acceso al token AFS y, por tanto, al sistema de ficheros AFS. Como siempre, k5start renovará puntualmente el TGT. Una vez modificado el script de inicio con las directrices descritas, reiniciaríamos el servicio para hacerlas efectivas pero, antes, debemos asegurarnos de crear una identidad AFS que sea mapeable desde el principal KERBEROS, en otro caso no será posible obtener ningún token.

El siguiente diagrama vuelve a presentar el proceso de autenticación KERBEROS pero ahora incluyendo las modificaciones y particularidades cuando el cliente AFS es, en lugar de un usuario como ya se expuso, un autómata.

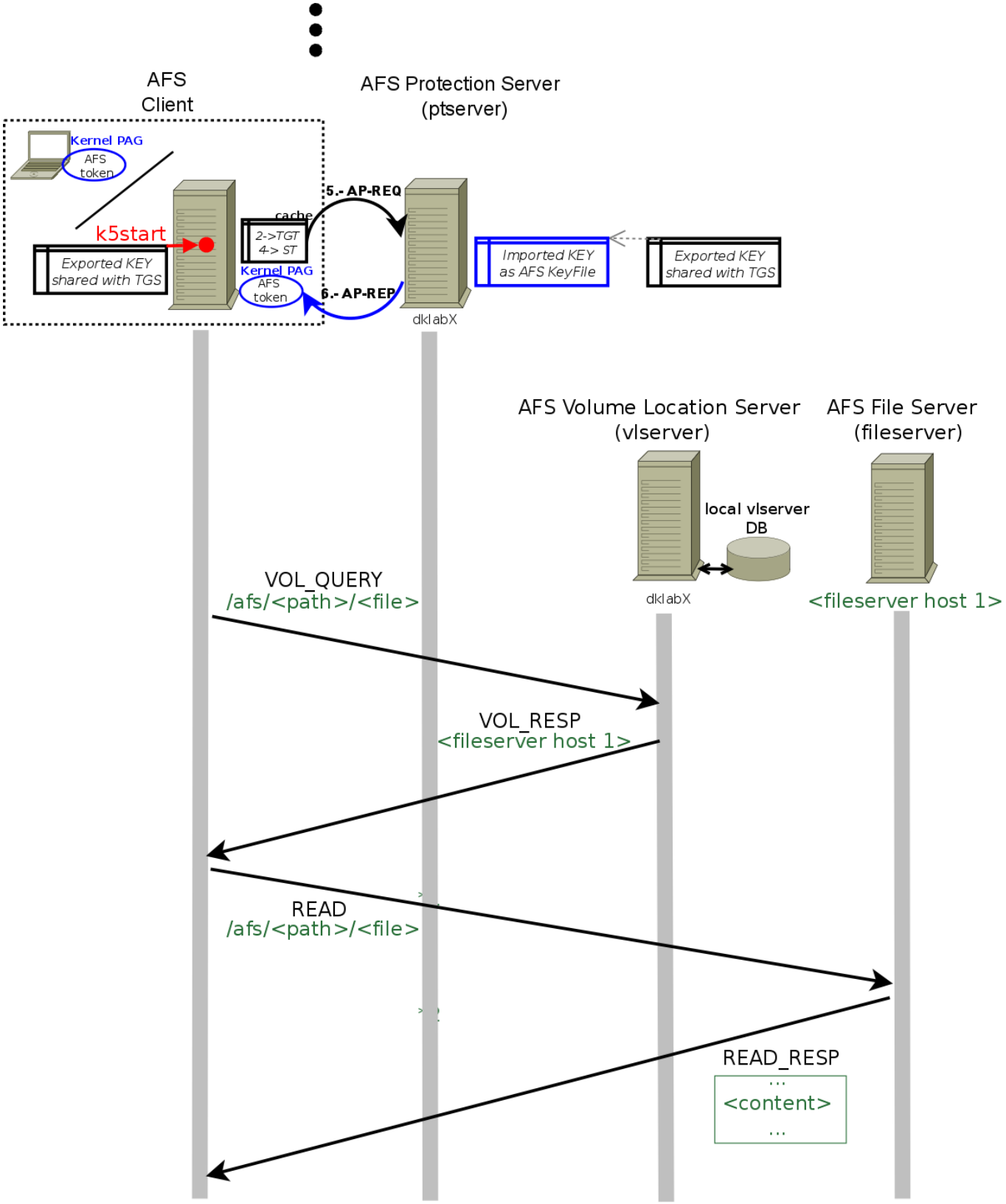
[102] http://docs.openafs.org/AdminGuide/ch02s10.html#HDRWQ64



Efectivamente, el diagrama simplemente incluye, en la autenticación Kerberos para autómatas, las tres particularidades que presentamos cuando el servicio kerberizado es AFS. El único detalle adicional tiene que ver con la necesidad de que el autómata sea lanzado como proceso hijo de aquel que obtiene el token AFS. Efectivamente k5start

lanza al autómata como proceso hijo y ésto se simboliza en el diagrama con la flecha roja que parte de k5start y empuja (a la memoria) el círculo rojo que representaría por tanto al autómata.

El siguiente diagrama presenta un ejemplo de comunicación AFS. El primer paso ya se expuso y consistía en la autenticación del cliente por ptserver con la expedición de un token AFS. Los puntos suspensivos en la parte superior vienen a reforzar esa idea de que ese primer paso viene de lo que se expuso más arriba.


AFS
Client
AFS Protection Server
(ptserver)
Kernel PAG
AFS
token
k5start
Exported KEY
shared with TGS
cache
2->TGT
4-> ST
Kernel PAG
AFS
token
5.- AP-REQ
6.- AP-REP
dklabX
Imported KEY
as AFS KeyFile
Exported KEY
shared with TGS
AFS Volume Location Server
(vlserver)
AFS File Server
(fileserver)
local vlserver
DB
dklabX
<fileserver host 1>
VOL_QUERY
/afs/<path>/<file>
VOL_RESP
<fileserver host 1>
READ
/afs/<path>/<file>
READ_RESP
...
<content>
...

Una vez que el cliente AFS (ya sea un autómata o una aplicación lanzada por un usuario) tiene un token AFS válido, inicia una comunicación con el componente vlserver (AFS "Volume Location Server") para consultar qué fileserver (AFS "File Server") almacena el volumen que a su vez almacena el contenido referenciado por la ruta /afs/<path>/<file>. Entonces vlserver consulta su base de datos de localización de volúmenes y le devuelve una respuesta adecuada a las interfaces de red que, según tenga conocimiento vlserver, dispone el cliente. Por fin, gracias a la información recabada del vlserver y el token AFS, el cliente puede contactar con el fileserver adecuado, pedirle la lectura de un fichero y recibir su contenido.

Las reglas de mapeado entre principal KERBEROS e identidades AFS son más bien rígidas. Básicamente la identidad AFS coincide con el <primary>del principal cuando no existe <instance>. En otro caso, la identidad AFS se forma como <primary>.<one_dot_instance>y donde <one_dot_instance>es el instance hasta y sin incluir el primer punto al ser leída de izquierda a derecha. Adicionalmente, tras el mapeo la identidad debiera tener menos de 63 caracteres (idealmente alrededor de 8) y no utilizar símbolos como: @,:;.<space><newline>.

A continuación detallamos las identidades AFS de nuestra infraestructura:

Principal	Identidad AFS
umea@CASAFX.DYNDNS.ORG	umea
aemu@CASAFX.DYNDNS.ORG	aemu
imap/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG	imap.dklab1
imap/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG	imap.dklab2

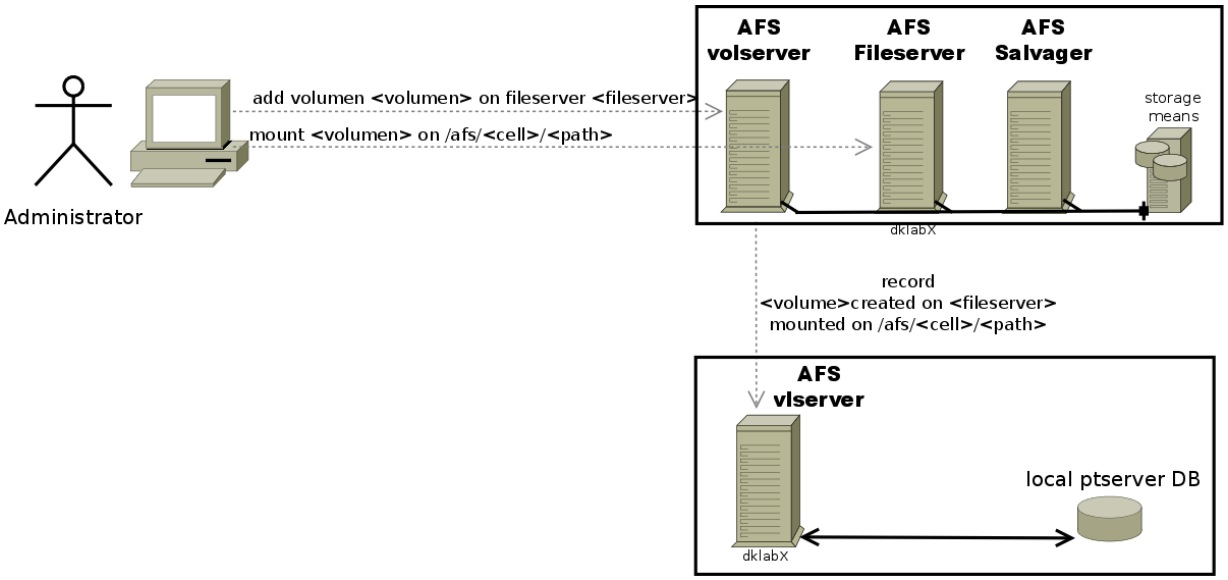
Como se aprecia, los servidores del sistema de correo tendrán necesidad de hacer de clientes AFS, ya que el correo se almacena allí. La sección 1.2.5 explica por qué, a pesar de que son varios los componentes del sistema de correo que accedea a AFS (el que hace la entrega, el servidor IMAP, el servidor MANAGESIEVE), sólo es necesaria una identidad AFS por cada host, siendo elegida la obtenible a partir del principal indicado en la tabla, el del servidor IMAP.

Tanto los servidores de correo mencionados, como los usuarios mediante su home, necesitan espacio de almacenamiento en AFS según nuestro diseño del sistema de correo y de las cuentas centralizadas de shell. Ésto implica que debemos crear volúmenes AFS dedicados a tal fin.

Efectivamente, el administrador durante su despliegue de AFS reserva al menos una partición de su medio de almacenamiento para el trío "fs" de AFS. El trío "fs" corresponde al componente filserver que servirá los volúmenes, al componente volserver que permite su gestión remota y al componente salvager que permite su comprobación remota. Los volúmenes son las unidades en que divide AFS el sistema de ficheros, donde cada volumen almacenaría por tanto un subárbol (con o sin nodos hoja) del árbol bajo /afs/<cellname>/.

Por tanto el administrador debe contactar con volserver en aquellas máquinas que, albergando un trío "fs", considere necesario crear los volúmenes de home o almacenes de correo. Un buen criterio de elección es la cercanía geográfica media entre los usuarios de esa parte del árbol y el servidor donde estarán alojados los volúmenes que lo almacenan.

Adicionalmente a la creación, deberá montarlos en algún punto del árbol proporcionado por otros volúmenes previamente montados.



Como se comentó, AFS es un software complejo, el anexo procura ser especialmente explicativo y por tanto recomendamos la lectura de las discusiones que se dan allí.

1.3.11. Introducción a los servicios finales de usuario

Comenzamos ya con los servicios finales de usuario. Este cambio reorienta nuestro planteamiento de exposición y nos preguntamos fundamentalmente por dos cuestiones.

En primer lugar, desde el punto de vista del usuario qué ofrece nuestro despliegue particular.

En segundo lugar, desde el punto de vista del administrador qué facilidades de las ofrecidas por la infraestructura de servicios base son usadas por estos servicios finales de usuario.

Adicionalmente, diremos que la identidad en estos servicios es de la forma <user>@<dominio>. Como sabemos es fundamental conocer el tipo de mapeo que desde el principal nos lleva a ese formato de cuenta:º

Principal KERBEROS	Identidad Servicios Finales (Cuenta)
<primary>/<instance>@<REALM>	<primary>@<dominio>

Donde es claro que los servidores conocen el <dominio>al que pertenecen.

El mapeo en el servicio de Shell es especial. Puesto que la autenticación con el KERBEROS AS se inicia en nuestro caso al log-in POSIX, el mapeo en este caso es desde la identidad POSIX, la cuenta de Shell, al principal.

Cuenta Shell Centralizada	Principal KERBEROS
<primary>	<primary>/<instance>@<REALM>

1.3.12. Shell a Cuentas Centralizadas

Anexo:	7

En primer lugar, analizamos qué ofrece al usuario. Introductoriamente diremos que el servicio de Shell a cuentas centralizadas representa nuestra implementación genérica a los servicios de tipo persona-sistema.

Funcionalidad para el usuario y referencias a su despliegue

El usuario obtiene:

1. Posibilidad de utilizar completamente los recursos de las computadoras gracias a la interfaz de Shell.
2. Posibilidad de utilizar todas las computadoras de la misma forma: misma contraseña, mismo home, mismos documentos, mismo software.
3. Posibilidad de cambiar de una máquina a otra buscando mayor capacidad de cómputo o alguna otra característica hardware.
4. Asegura SSO posteriormente.

Para conseguir las funcionalidades 2 y 4 es necesario integrar los servicios de la infraestructura base con la secuencia de log-in del SO. Respecto a esta secuencia y tomando como modelo a SO POSIX, específicamente Debian, la sección 1.2 describe el sistema PAM de interfaz a distintos métodos de autenticación (como el de KERBEROS AS) y establecimiento de variables de sesión (como la obtención de un token AFS). La sección 1.3 describe el subsistema NSS para recabar metadatos desde bases de datos centralizadas como DNS o LDAP (por ejemplo el uid, gid, home, etc).

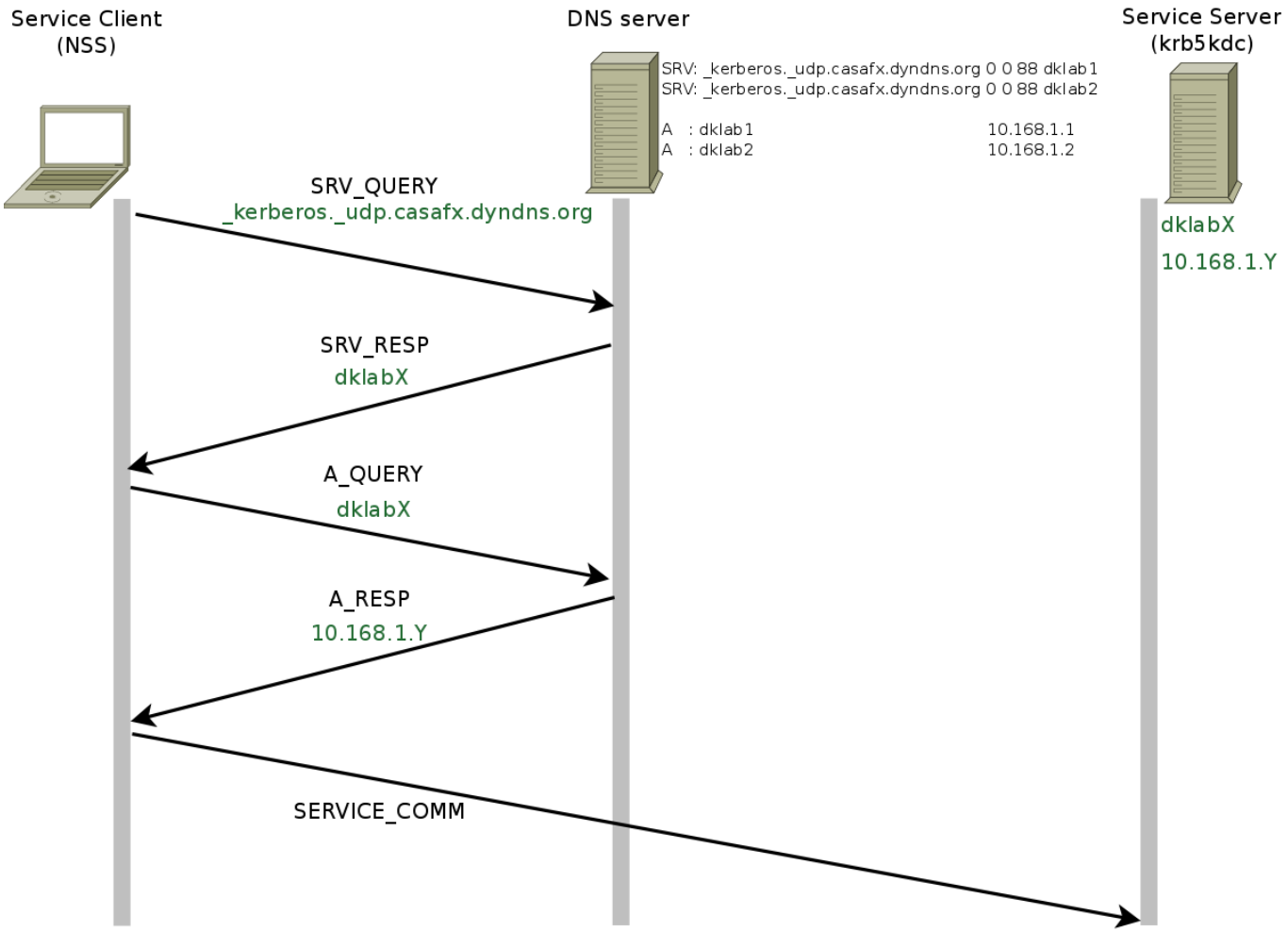
Así pues gracias a NSS por un lado, y PAM apoyado en NSS por otro, el log-in permite que las cuentas puedan estar centralizadas (al recabar metadatos desde LDAP), asegura SSO posteriormente (al obtener el TGT desde el KERBEROS AS) y sitúa el home en AFS (al obtener el token AFS tras el TGT). Las secciones 1.4.1 a 1.4.5 y 1.5.1 a 1.5.5 detallan el proceso concreto.

La funcionalidad 3 se consigue in situ (sin cambiar físicamente de máquina) utilizando SSH. La sección 1.4.7 del anexo detalla cómo desplegar OpenSSH y configurarlo para hacer log-in en el sistema (contactando con el AS), así como el mencionado cambio de máquina SSO. También se ofrece una breve discusión sobre en qué direcciones se puede extender esa forma de trabajo remota (posibilidad de gráficos, sonido).

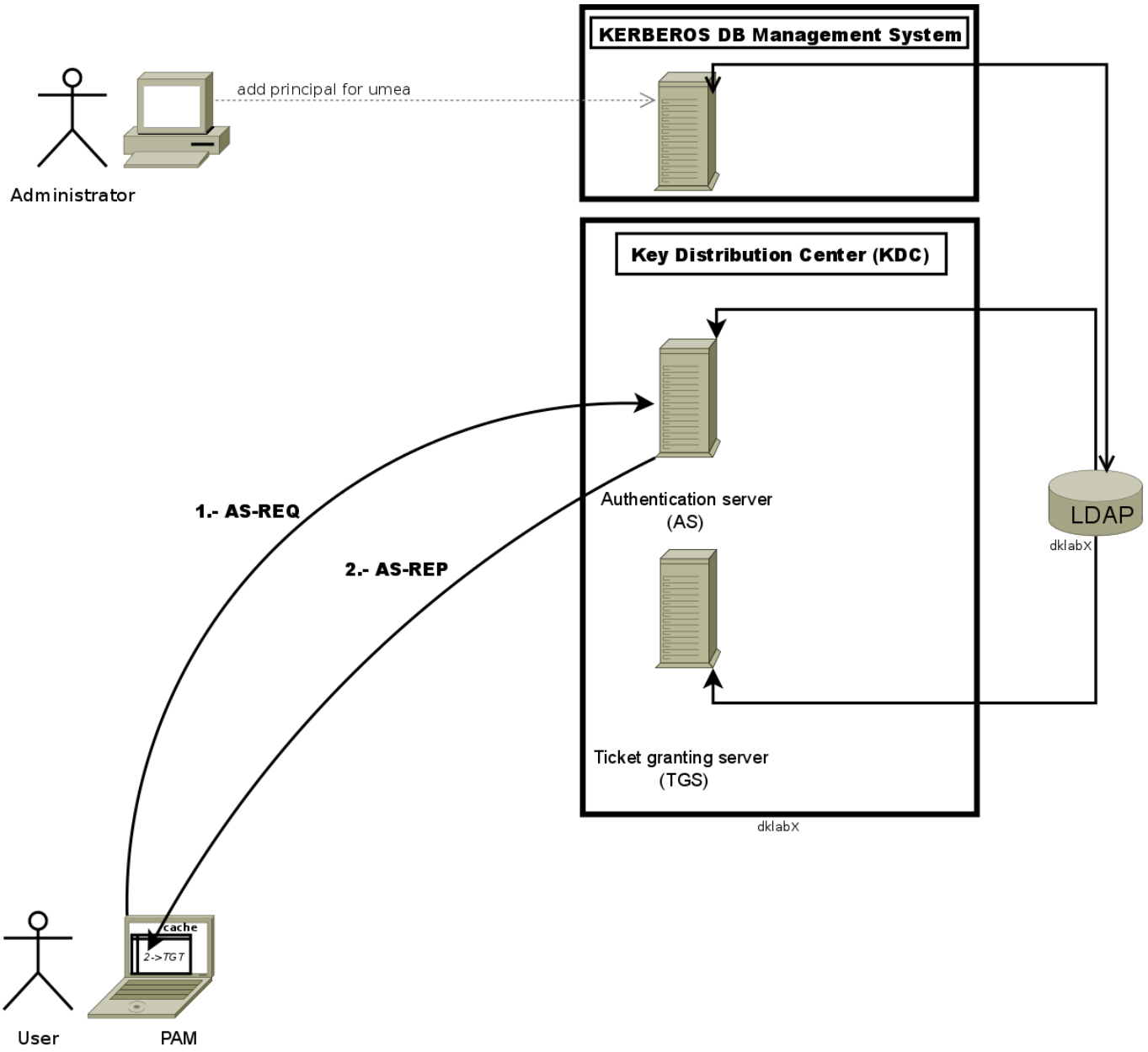
Explotación de la Infraestructura de Servicios Base por el servicio de Shell

Supongamos que el usuario referido por la cuenta umea escoge una computadora del parque informático y comienza la secuencia de log-in. El usuario pone su nombre de usuario (umea) y contraseña.

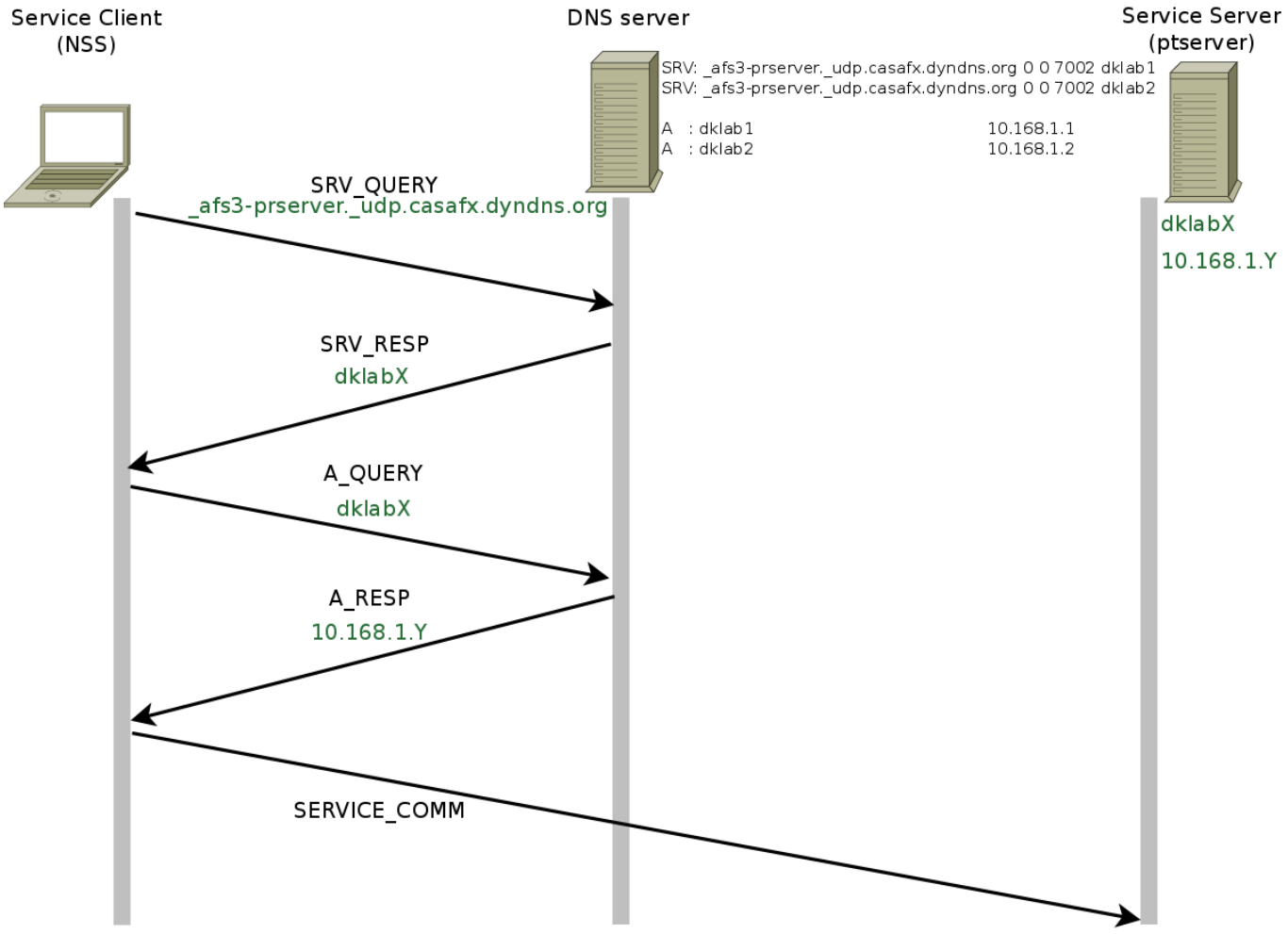
En primer lugar, se recoge la información del prompt y el sistema PAM intenta la autenticación a través de KERBEROS. Inicialmente busca los servidores KERBEROS a través de una consulta SRV:



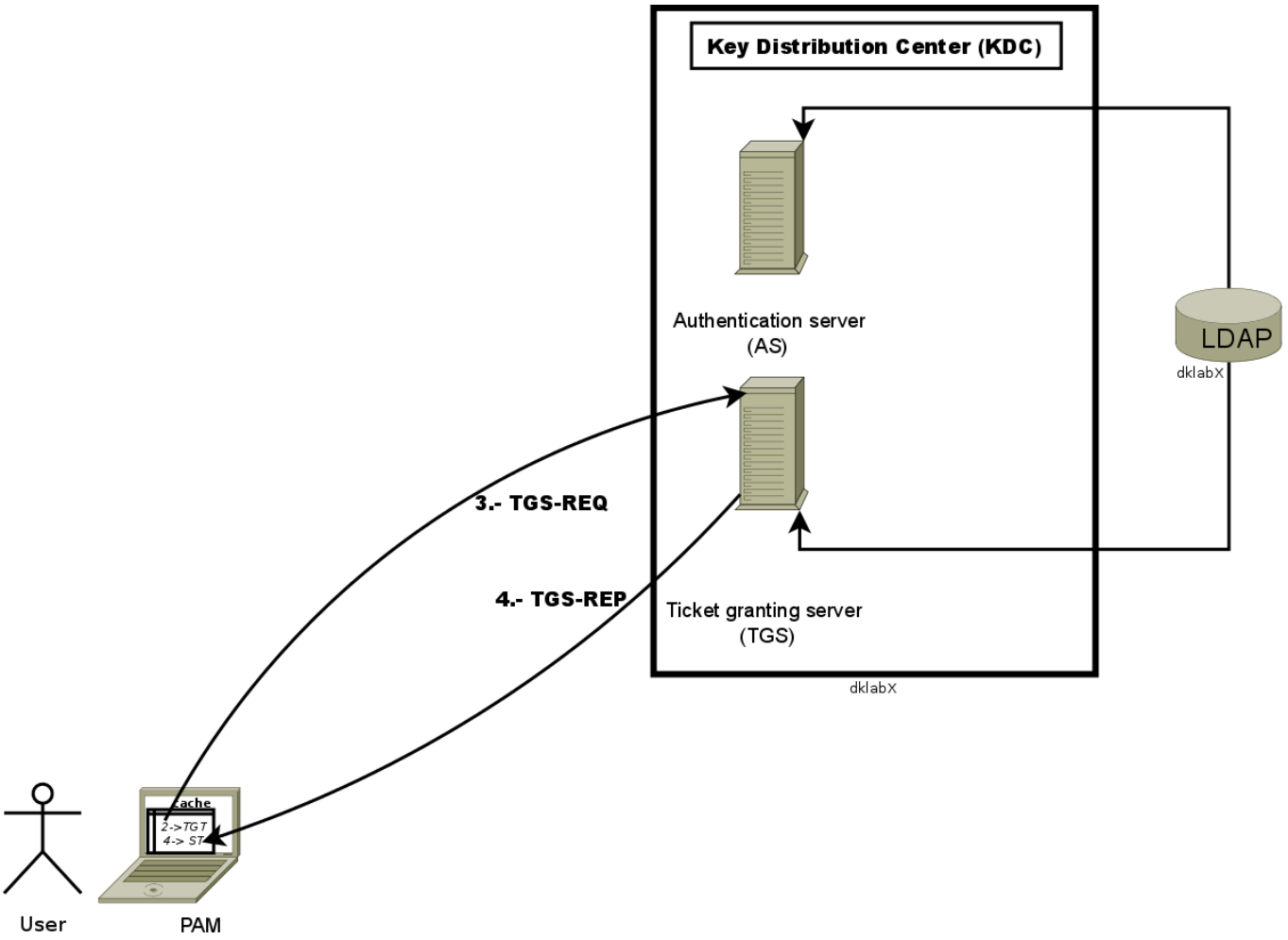
Se utiliza la contraseña para autenticarse contra el KERBEROS AS. Si todo va bien se devuelve el TGS que es almacenado en la caché de credenciales (normalmente el fichero /tmp/krb5cc_<uid>).



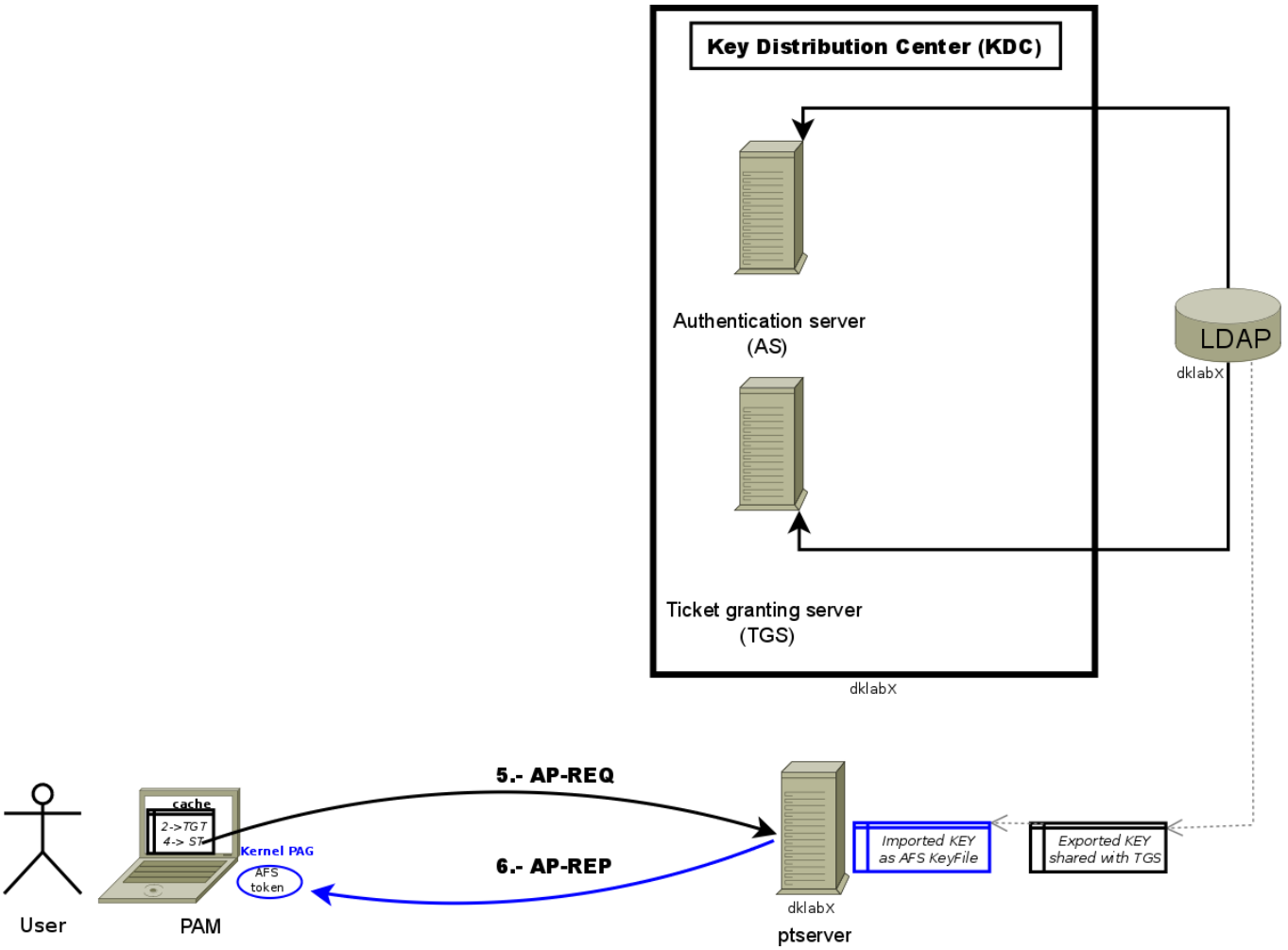
Entonces PAM se ocupa de conseguir el token AFS. Ésto requiere descubrir la localización del AFS ptserver, de nuevo a través de una consulta DNal registro SRV:



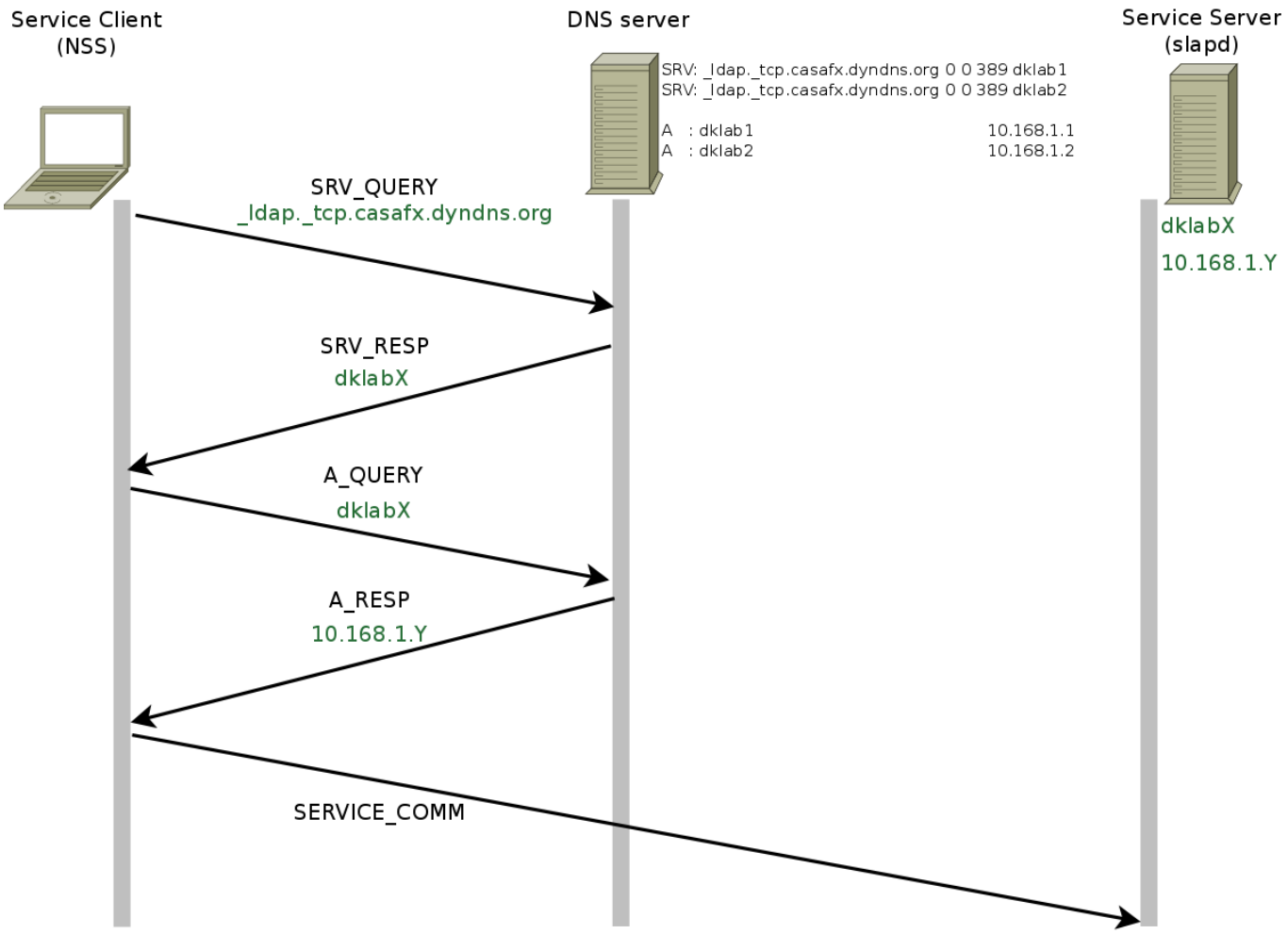
Una vez localizado el ptserver, PAM utiliza el TGS para conseguir el ticket ST:



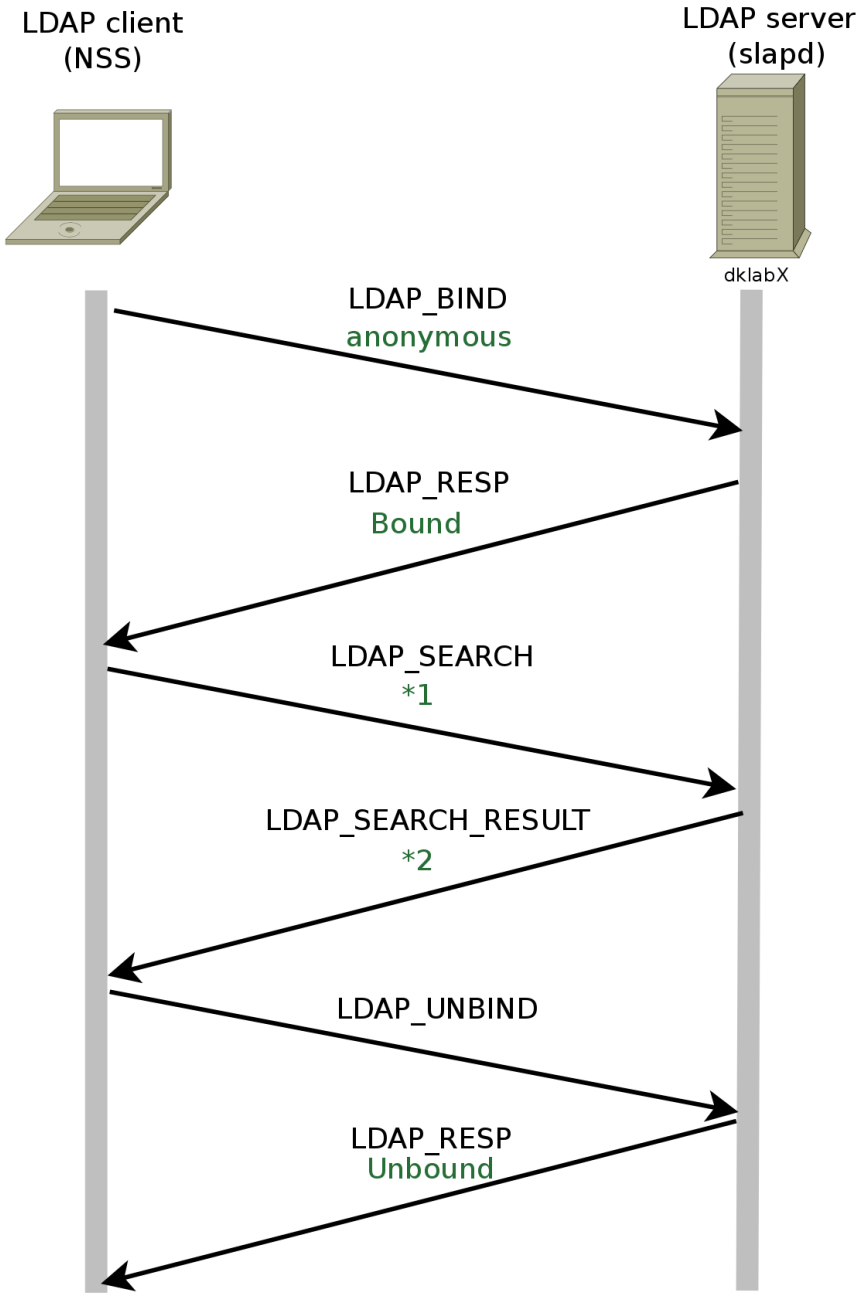
Y a continuación se lo presenta a ptserver y, si toda va bien y ptserver es igualmente identificado, ptserver expide finalmente el token AFS.



Puesto que umea ha conseguido su token AFS, ya puede situarse su home en AFS, y hacer que la shell de usuario pueda ejecutar sus scripts de inicialización (profile etc). Sólo hay un problema: aún no sabe cuál es su home, ni su shell, ni su uid etc. Entra en juego por tanto el susbsistema NSS y sus consultas a LDAP para proveer esa información.



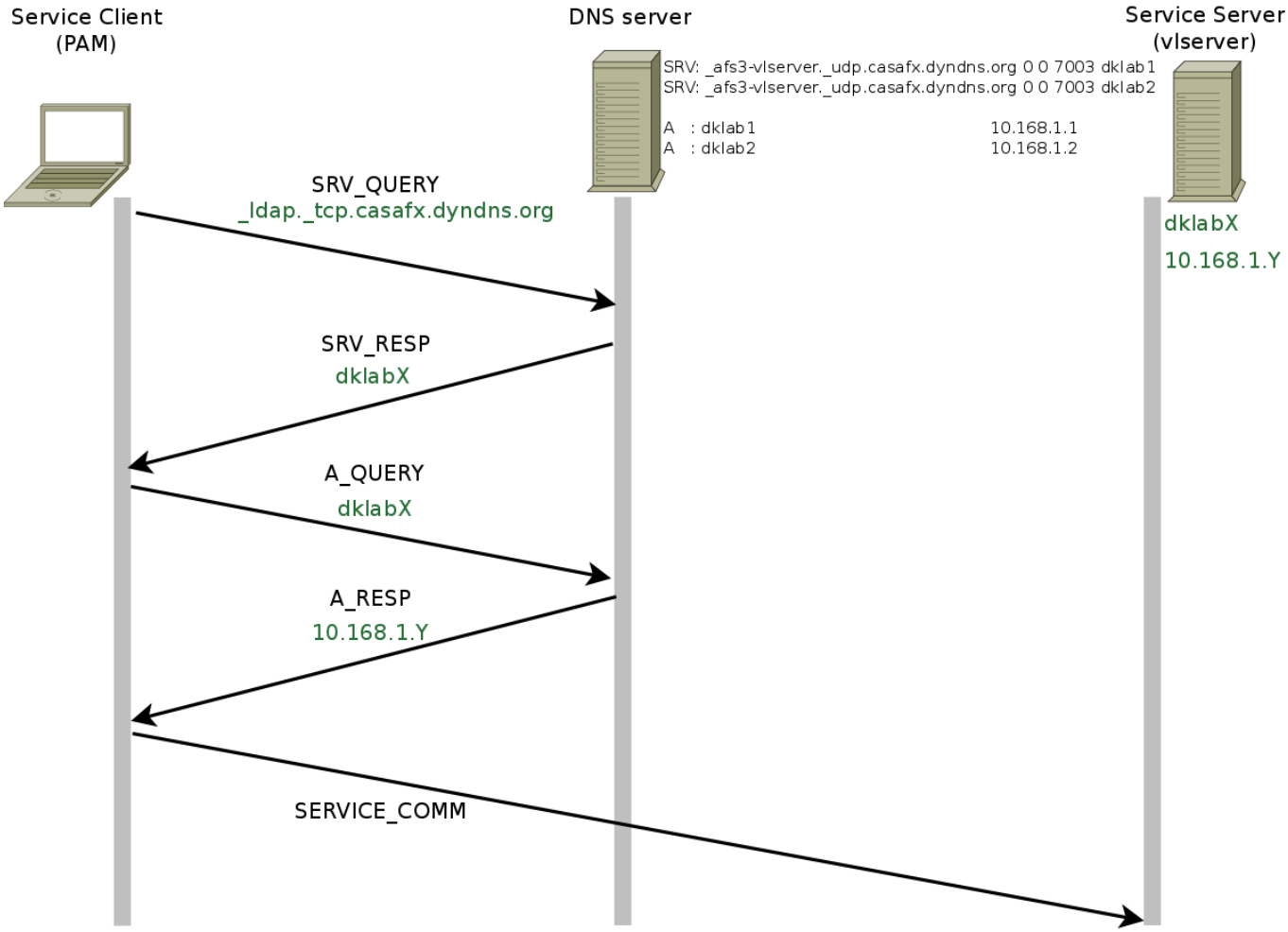
Localizado el servidor LDAP, una consulta LDAP ("search") permite recabar la información de la cuenta centralizada POSIX de umea:



*1
```
base="ou=accounts,dc=casafx,dc=dyndns,dc=org" scope=2
filter="(&(objectclass=posixaccount)(uid=umea))"
attr=uid uidNumber gidNumber homeDirectory loginShell
```

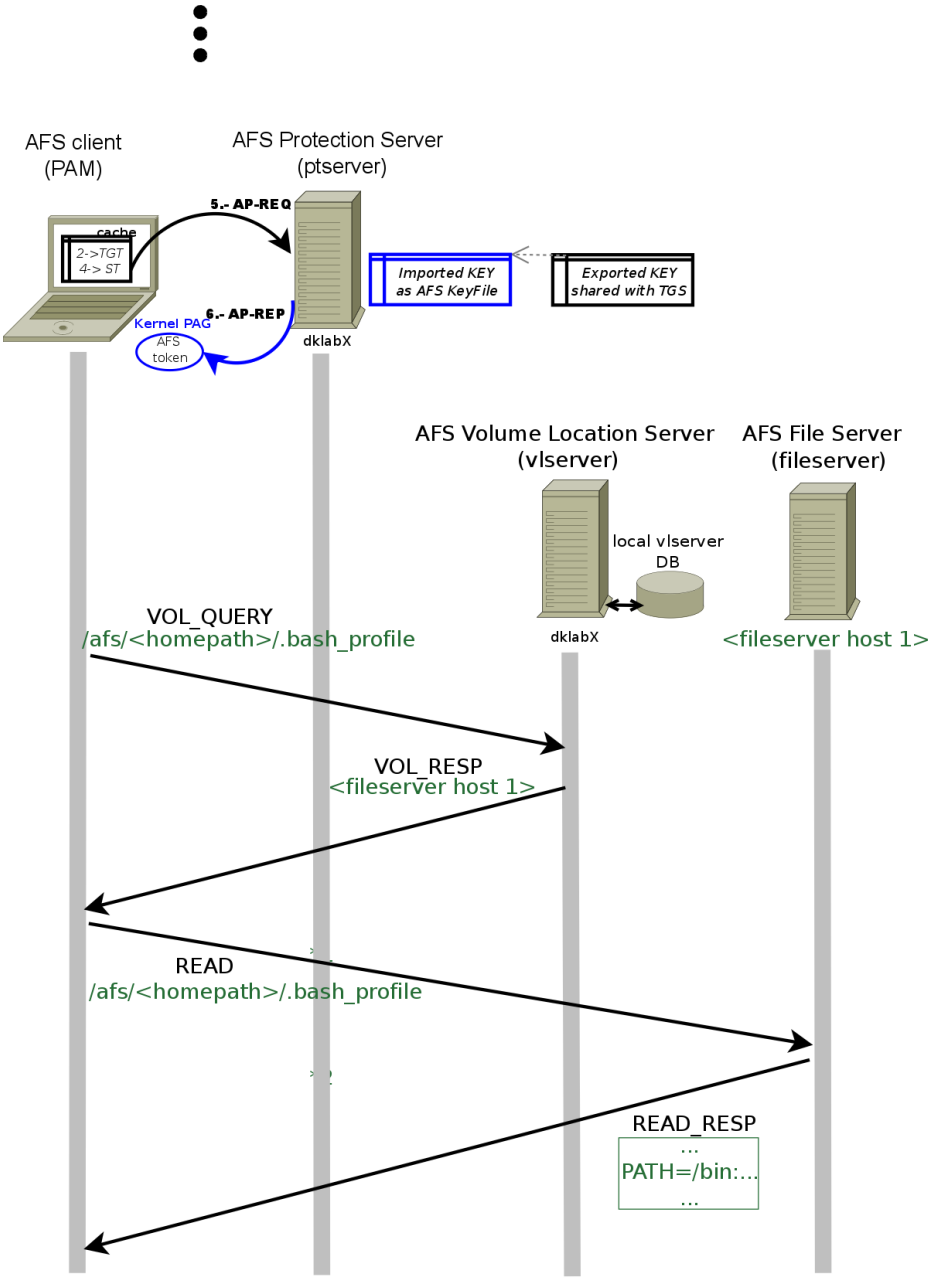
*2
```
uid             umea
uidNumber       31000
gidNumber       31000
homeDirectory   /afs/casafx.dyndns.org/user/u/um/umea
loginShell      /bin/bash
```

Finalmente, ya es posible contactar con vlserver y fileserver para usar AFS y situar a umea en su home. El servidor vlserver ha de ser descubierto a través de su registro SRV en DNS:



Entonces, el cliente AFS (representado en este momento por PAM) contacta con vlserver para conocer qué componente AFS fileserver de nuestra celda sirve los contenidos correspondientes al home del usuario. Para ello el cliente AFS realiza una consulta con argumento la ruta del home de umea; entonces vlserver la recibe, consulta su base de datos y devuelve la localización del fileserver que gestiona el volumen correspondiente a esa ruta. Es claro que, según ese proceso, fileserver no se descubre a través de registros SRV, ésto sólo ocurría con ptserver y vlserver pues, precisamente éste último, tiene como cometido descubrir al fileserver en función de la ruta AFS. Así pues, retomando AFS por donde lo dejamos (obtención del token AFS):

- Anotación: en el siguiente diagrama se utiliza de ejemplo la lectura del fichero de inicialización de la shell de umea, presente en su directorio home.



1.3.13. XMPP/Jingle

Anexo:	8

XMPP/Jingle representan nuestra implementación a los servicios persona-persona síncronos. Comenzamos describiendo qué ofrece a los usuarios finales.

Funcionalidad para el usuario y referencias a su despliegue

1. Posibilidad de conversaciones de texto, voz o videoconferencia con otras personas tanto de la organización como de fuera (incluyendo Gtalk, Facebook, otros).
2. Activación automática de la cuenta tras la primera autenticación exitosa.
3. Subscripción automáticas a los miembros de la organización.
4. Agrupamiento automático de los contactos por la sede/departamento de la organización a la que pertenezcan.
5. Posibilidad de consultar la identidad de los contactos a través de tarjetas de visita electrónicas (Vcard) automáticas.
6. Posibilidad de uso a través de programas tanto en la interfaz de línea de comandos como en la gráfica de usuario.
7. Confidencialidad.

La implementación escogida del lado del servidor fue Jabberd2, en la sección 1.1 se detalla su arquitectura de componentes que cooperan entre sí para luego, en la sección 1.2.3, sintetizar en forma de tabla sinóptica qué componente es responsable de cada funcionalidad de las enumeradas y los detalles para ponerlas en práctica. Nos fue extraordinariamente útil compilar con soporte de depuración y lanzar manualmente los componentes con el flag "-D" para descubrir erratas en la configuración y otros problemas. Adicionalmente diremos que la arquitectura de Jabberd2 tiene un punto único de fallo, pues su componente router no es clusterizable de acuerdo a su autor. Efectivamente tuvimos que monitorizar con init, cual watchdog, a ese proceso. También lo hicimos para jabberd2-s2s puesto que sólo un componente jabberd2-s2s representa la ruta por defecto a otros servidores XMPP de otras organizaciones.

Las funcionalidades 3 (subscripciones automáticas) y 5 (tarjetas electrónicas Vcard automáticas) necesitaron que confeccionásemos un parche contra las fuentes de Jabberd2 (escrito éste en C). El carácter automático de ambas funcionalidades se debe, como detalla la sección 1.2.3 en su último apartado, a que tanto las subscripciones como la información de la tarjeta se crean "al vuelo" consultando los objetos y atributos presentes en LDAP. El problema que resuelve el parche es básicamente cómo mapear el nombre de la cuenta a su nodo en el DIT LDAP. El parche se ofrece convenientemente codificado en base64 primero y como texto después, durante la sección 1.2.1. Dicha sección discute además el proceso de compilación e instalación.

La sección 1.2.3, apartado "Certicados X.509" detalla el uso de la CA que creamos cuando desplegamos OpenVPN. Efectivamente se crean certificados X.509 que serán usados en las comunicaciones TLS entre los distintos componentes de la arquitectura de Jabberd2. Adicionalmente el componente jabberd2-c2c utiliza su certificado también para sus comunicaciones con los clientes. Ésto es necesario, ya que no hay otra forma de proveer en dichas comunicaciones la confidencialidad mencionada en el punto 7: Jabberd2 se enlaza contra la implementación Gsasl de SASL, la cual no implementa el establecimiento

de una capa de seguridad tras la autenticación SASL-GSSAPI (y por tanto, KERBEROS) de la que hacen uso los clientes de los usuarios.

Los clientes en modo texto y gráfico a los que hace referencia la funcionalidad 6 se describen en la sección 1.5 del anexo. Es significativo que en su subsección 1.5.1 se expone cómo Pidgin (el cliente XMPP escogido) hace uso de distintos subsistemas a través de la traza de log que muestra al activar su modo depuración. Efectivamente en dicha traza se puede observar cómo descubre el servicio a través de los registros SRV en DNS, cómo se lleva a cabo la autenticación SASL-GSSAPI o cómo induce en el servidor una conexión a OpenLDAP en busca de nuevos contactos.

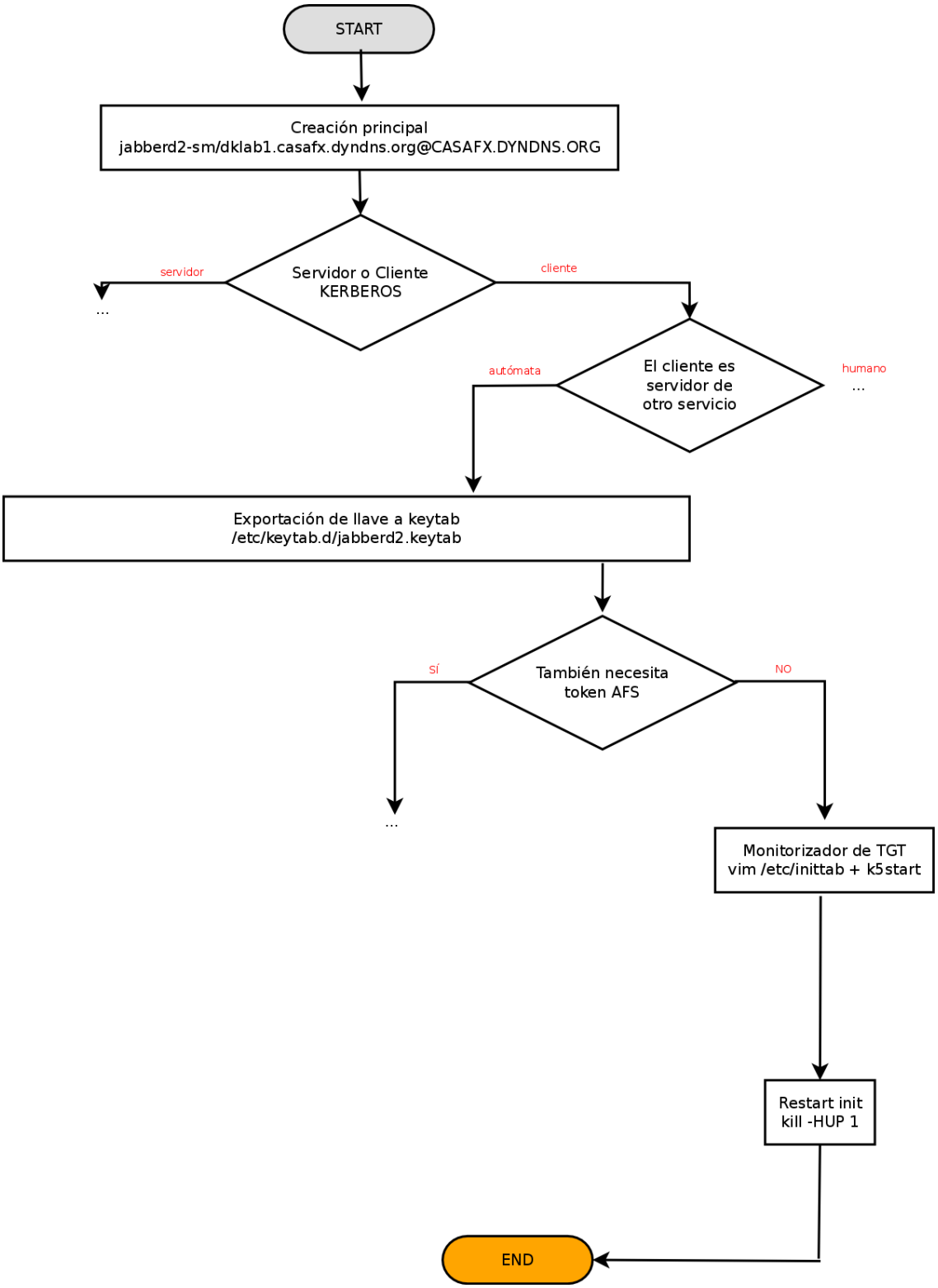
- Anotación: es conveniente resaltar que este proyecto no incluye IAX como estaba inicialmente previsto, en su lugar la información multimedia que señaliza Jingle es transportada a través de RTP. Efectivamente en un principio estaba previsto utilizar IAX, ésto era así porque el cliente XMPP/Jingle seleccionado fue Coccinella. **Cuando comprobamos que el soporte para autenticación SASL-GSSAPI (y por tanto KERBEROS) de Coccinella no funcionaba, cambiamos a Pidgin. Pero Pidgin no utiliza Jingle/IAX sino Jingle/RTP.**

- Se hacen las correspondientes pruebas. La posibilidad de telefonía parece estar disponible según nos lo indica el cliente Pidgin, sin embargo un problema de Qemu con el soporte de sonido nos impide probarlo más allá.

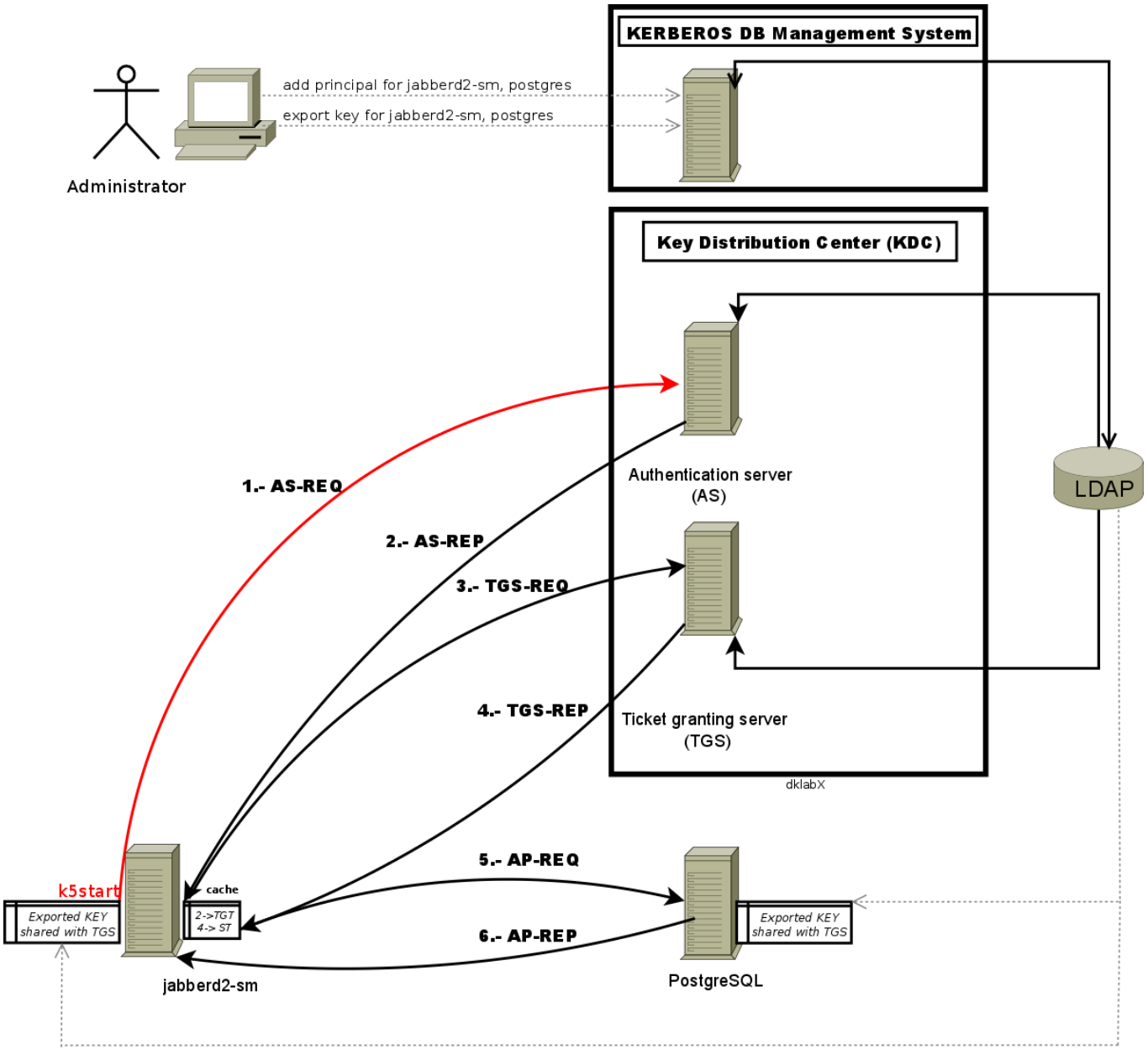
Explotación de la Infraestructura de Servicios Base

Como sabemos, Jabberd2 utiliza en exclusiva a PostgreSQL para algunos de sus metadatos. Para ello, hemos de crear un rol de PostgreSQL llamado "jabberd2" y una base de datos también llamada "jabberd2". Jabberd2 diseña un esquema SQL para ese almacenamiento, pero antes de cargarlo lo hemos de modificarlo ligeramente para que Bucardo lo pueda utilizar; la sección 1.2.2 tiene los detalles.

El componente jabberd2-sm es el que se conecta a PostgreSQL, de forma que necesita soporte para autenticarse a través de KERBEROS tal como hicimos para Bucardo. Si ya hemos editado el pg_ident.conf con los mapeos correspondientes del principal a "jabberd2", podemos pasar a crear el principal y demás modificaciones:

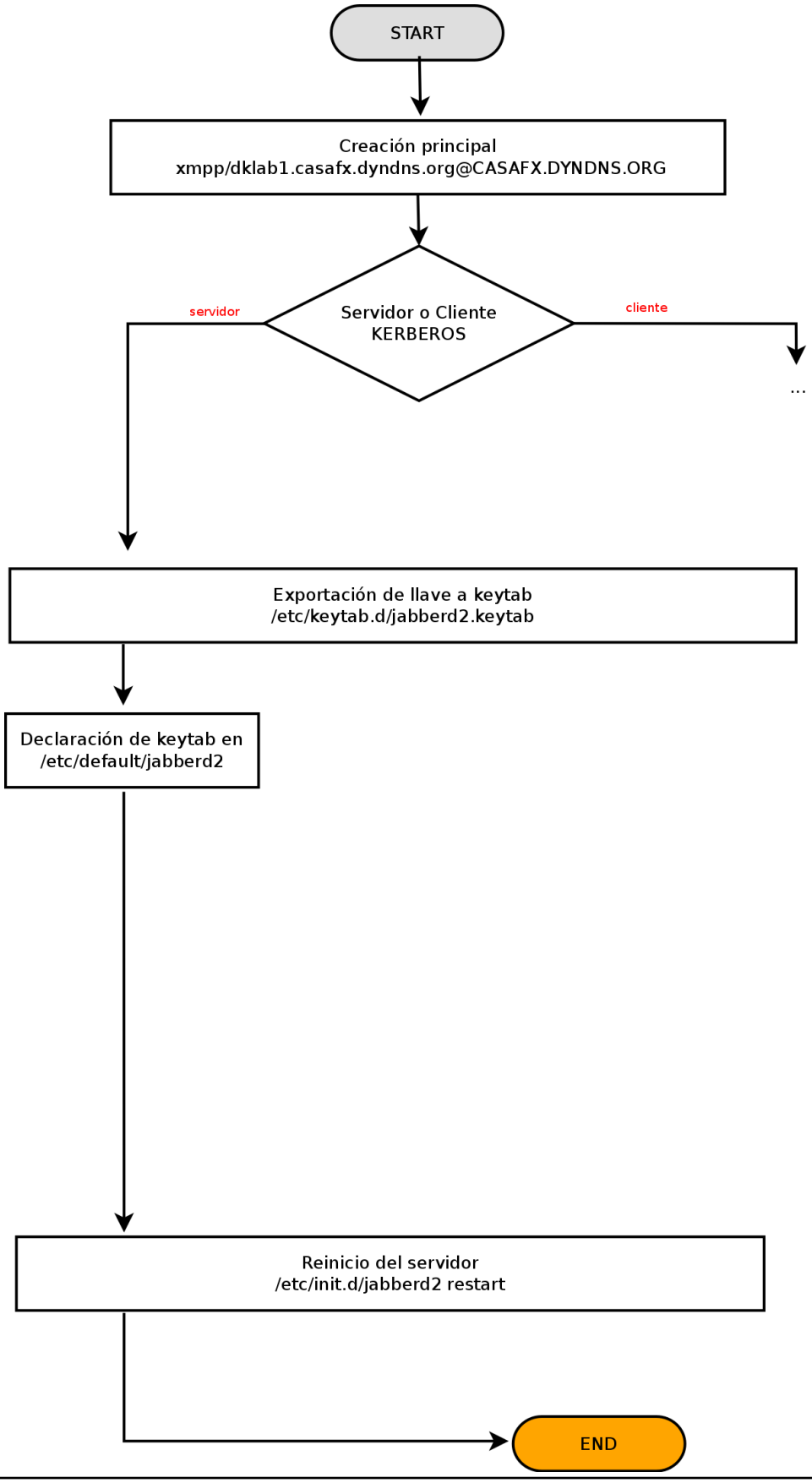


La autenticación KERBEROS entre jabberd2-sm y PostgreSQL se da, de forma parecida a como vimos para Bucardo:



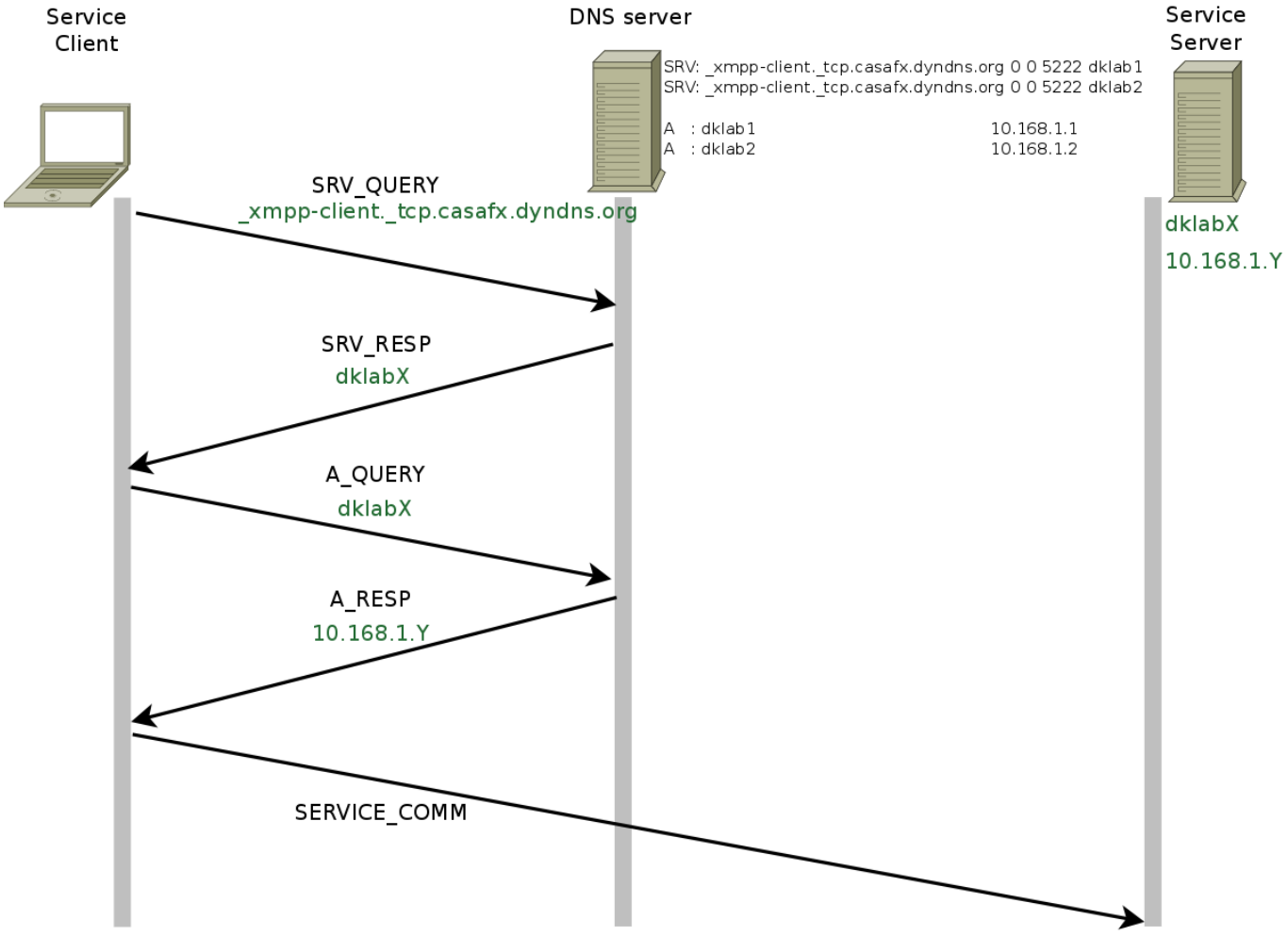
Evidentemente, puesto que ya se creó el principal para postgres con anterioridad, aquí se muestra sólo para clarificar la relación entre ambos principal.

El componente jabberd2-c2s debe, por su lado, kerberizarse para actuar con el rol de servidor KERBEROS durante la autenticación de los clientes finales:

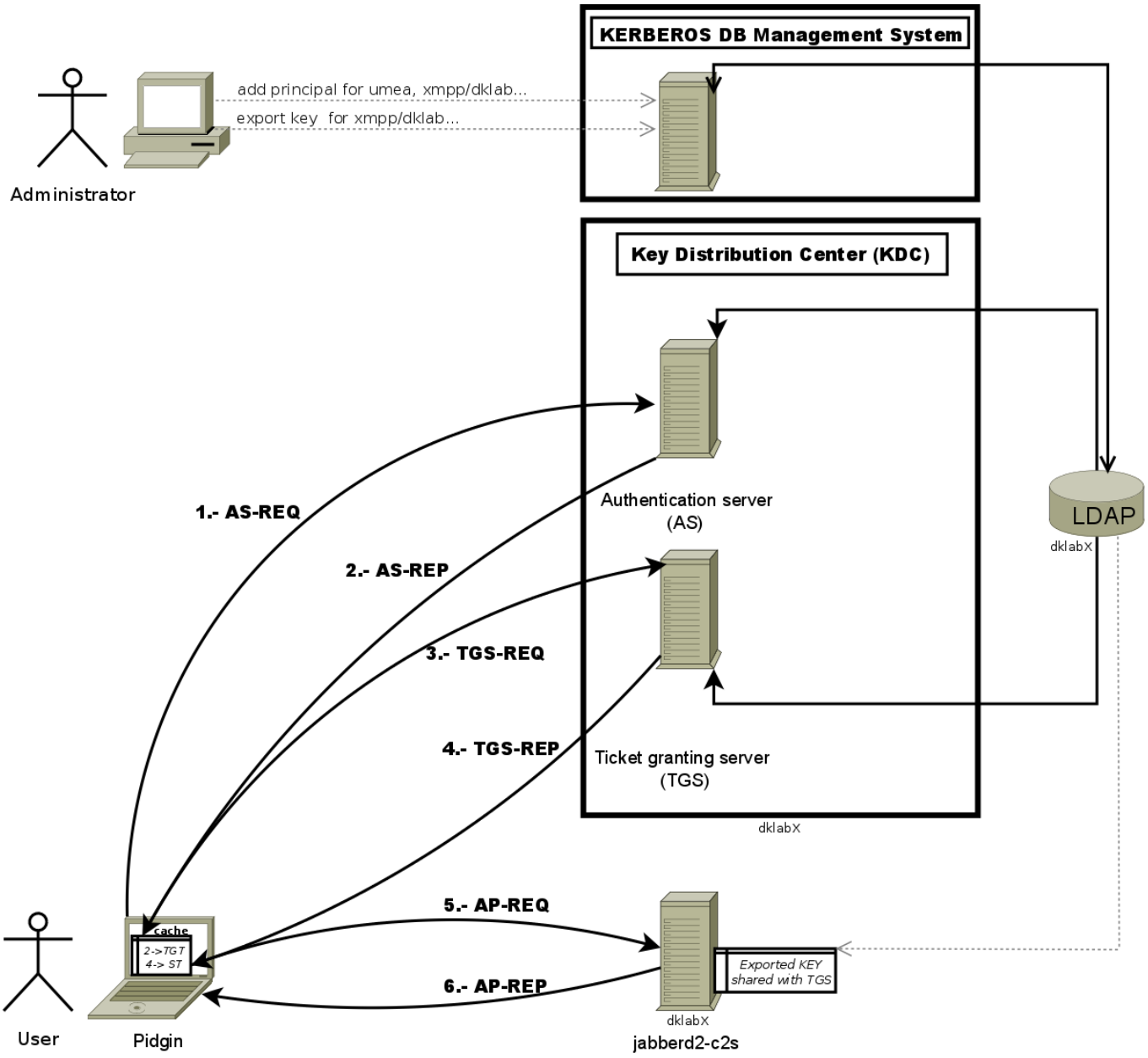

START
Creación principal
xmpp/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
Servidor o Cliente
KERBEROS
servidor
cliente
...
Exportación de llave a keytab
/etc/keytab.d/jabberd2.keytab
Declaración de keytab en
/etc/default/jabberd2
Reinicio del servidor
/etc/init.d/jabberd2 restart
END

En dklab2 se realiza un despliegue parecido al que hemos hecho para dklab1.

Por tanto si, por ejemplo umea quiere utilizar el servicio XMPP, lanza su cliente (Pidgin) y éste realiza un descubrimiento del servicio basado en los registros SRV y el nombre del dominio. Este último lo conoce porque lo induce del domain-part de su JID (la parte tras la @ en el nombre de su cuenta):



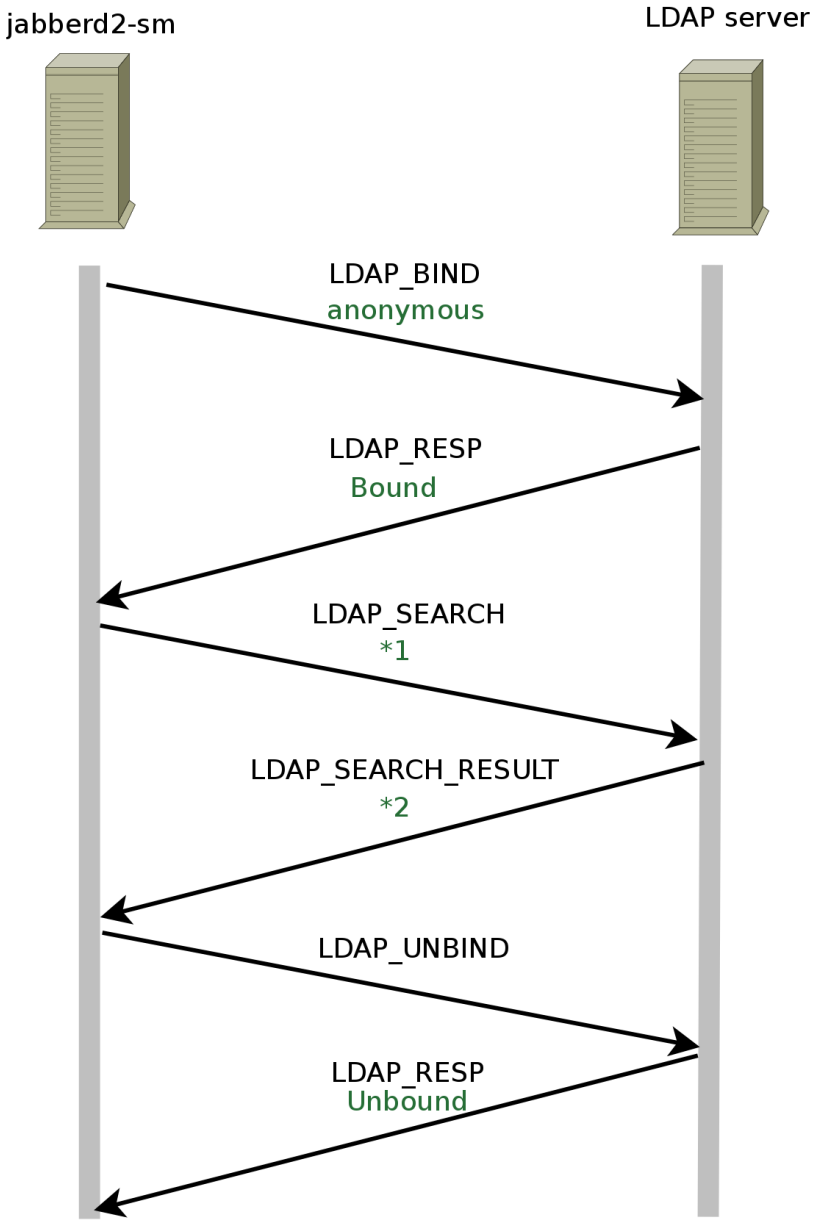
Entonces tiene lugar la autenticación SASL-GSSAPI (KERBEROS):



Debe quedar claro el mapeo en relación al servicio XMPP. Acorde con este ejemplo observamos que se cumple la relación que dimos en el apartado introductorio a los servicios finales:

Principal KERBEROS	Identidad XMPP (JID)
umea@CASAFX.DYNDNS.ORG	umea@casafx.dyndns.org

Si todo va bien, entrará en juego jabberd2-sm para hacer consultas en SQL y LDAP de forma que pueda cambiar el estado de presencia (SQL) o construir un roster y una Vcard al vuelo (LDAP):



*1
```
base="ou=accounts,dc=casafx,dc=dyndns,dc=org" scope=2
filter="(&(objectclass=posixaccount)(uid=umea))"
attr=uid departmentNumber cn mail
```

*2
```
aemu: uid                 aemu
      departmentNumber    SedeB
      cn                  nydala
      mail                aemu@casafx.dyndns.org

_
nuevo: uid                nuevo
       departmentNumber   ...
       ...
```

Con esa información, jabberd2-sm construirá un roster que le hará llegar al cliente XMPP. Hacemos notar que la comunicación LDAP no se ha precedido del descubrimiento de los servidores LDAP a través de consultas a registros DNS de tipo SRV porque jabberd2-sm no soporta ese descubrimiento de servicio.

1.3.14. SMTP, IMAP, MANAGESIEVE

Anexo:	9

El servicio de correo es nuestra implementación de servicio persona-persona asíncrona. Hemos procurado hacerlo robusto y flexible implementando varias de las tecnologías y funcionalidades que se han ido uniendo al servicio en distintos puntos de su dilatada línea histórica.

Funcionalidad para el usuario y referencias a su despliegue

1. Posibilidad de enviar, recibir y gestionar correo electrónico (incluyendo la gestión automática a través de filtros Sieve).
2. Subscripción automática a cuentas de correo públicas (boletines informativos de la organización, etc).
3. Barreras ante correo no deseado.
4. Barreras ante distribución de software malicioso a través del correo.
5. Posibilidad de autocompletado de la dirección y nombre de un destinatario de la organización al redactar un correo.
6. Correo informativo rebotado ante imposibilidad de entrega.
7. Posibilidad de usar un smarthost (basado en Gmail o Live) condicionalmente al dominio del destinatario.
8. Posibilidad de correo de aviso ante ausencias ("vacation mail").
9. Posibilidad de reenvío automático de correos.
10. Posibilidad de aliases.
11. Posibilidad de entrega de correo adicionalmente en el directorio home.
12. Posibilidad de denegar un envío de correo si se supera cierto tamaño máximo.
13. Posibilidad de cuotas de almacenamiento (ésto normalmente será considerado por los usuarios como una limitación).
14. Posibilidad de marcado del correo recibido.

15. Posibilidad de administración remota de los filtros de correo.

16. Posibilidad de uso a través de utilidades tanto en la interfaz de línea de comandos como en la gráfica de usuario.

17. Confidencialidad.

Son necesarios varios componentes y varios protocolos adicionales para desarrollar una infraestructura completa de correo. La sección 1.1 ofrece un repaso somero a la arquitectura del correo y detalla las implementaciones utilizadas del lado del servidor: Exim4 (SMTP/LMTP M.T.A.), Dovecot (M.D.A., IMAP server, MANAGESIEVE server), Procmail (M.D.A. alternativo).

La funcionalidad 3 (barreras ante correo no deseado) se implementa a través de filtros de contenido en la entrega (Spamassassing), así como verificación de origen basada en IP (SPF) y en firma digital (DKIM). Los detalles relativos a Spamassassing y pruebas de éste se encuentran en la sección 1.2.6. Por su lado los detalles y tests para SPF y DKIM se encuentran en sus homónimos apartados de la sección 1.2.8 del anexo.

La funcionalidad 4 (barreras ante distribución de software malicioso a través del correo) se implementa a través de otro escaner de contenido (Clamav), esta vez en tiempo SMTP. La sección 1.2.5 describe el proceso y prueba de este subsistema.

Por otro lado, las posibilidades 6 (correo rebotado ante problemas a la entrega), 7 (smarthost condicional) 8 ("vacation mail"), 9 (reenvíos automáticos), 10 (aliases), 11 (entrega adicional en el home) o 12 (límite de tamaño) corresponden a la configuración de enrutamiento del correo llevada a cabo en Exim4 y configurable a través de LDAP. La sección 1.2.4 en su apartado "Sistema de conguración de exim4" describe las generalidades de configuración de Exim4 (ciertamente complejas y extensas en aras de la flexibilidad). Entonces, la sección 1.2.8 en su apartado sobre el fichero de configuración local-router-250-ldap detalla cómo se configura cada uno de los routers utilizando el esquema LDAP del proyecto Qmail-ldap[103] y modificando el (providencial, dada la complejidad que comentábamos) ejemplo de uso confeccionado por Edison Wong[104] para Exim4. El caso del smarthost es diferente, el esquema LDAP de Qmail-ldap no adelanta soporte alguno para ésto, de forma que tuvimos que extenderlo en un espacio de OID privado[105], posteriormente diseñamos un router y transport específicos para Exim4 y realizamos pruebas utilizando cuentas de Gmail y Live de nuestra propiedad. Todo lo relacionado sobre smarthost condicionales se detalla en la sección 1.5.

De forma parecida, la funcionalidad 11 (entrega adicional en el home) se consigue configurando los transportadores de Exim4, véase la sección 1.2.8 en su apartado sobre el fichero local-transports-40-ldapusers. Además, la sección 1.2.9 en su último apartado desarrolla cómo usar Procmail para esa entrega alternativa en el home AFS. La entrega habitual también se realiza en AFS pero en una rama dedicada opaca a los usuarios

[103] http://www.qmail-ldap.org/wiki/index.php/Qmail.schema
[104] http://edin.no-ip.com/blog/hswong3i/exim4-ldap-0-0-5-initial-released
[105] Concretamente un subespacio de 1.3.6.1.4.1.7914
http://www.oid-info.com/cgi-bin/display?nb=7914&father_oid=1.3.6.1.4.1&action=display

pero no al servidor IMAP. Un componente de Dovecot ("deliver") realiza esta entrega sistemática.

La posibilidad 13 (cuotas) o 14 (marcado del correo recibido) se debe a comportamientos estandarizados del protocolo IMAP y soportados por el componente "imap" del proyecto Dovecot. Así mismo ocurre con la funcionalidad 2 (descubrimiento automático de buzones públicos), pues se apoya en los Namespaces y sistema de subscripciones de IMAP.

La funcionalidad 15 (administración remota de los filtros de correo) es el propósito del protocolo MANAGESIEVE, implementado de nuevo por un componente del proyecto Dovecot, "sieve". De esta forma la entrega del correo, la gestión IMAP y la gestión MANAGESIEVE son servicios proveídos por distintos componentes de Dovecot, a los que hay que añadir un cuarto componente, "dovecot-auth". Éste provee la autenticación SASL-GSSAPI en todos los protocolos (incluído SMTP, ya que Exim4 lo utilizará en esa fase de sus comunicaciones). La sección 1.2.9 detalla el despliegue completo de todos los componentes de Dovecot.

La confidencialidad a la que se refiere el último punto, acusa los mismos problemas que tenía, por ejemplo, Jabberd2. La implementación de SASL-GSSAPI (Gsasl o Dovecot-sasl) no implementa la parte del estándar referida a la negociación de una capa de seguridad que provea esa confidencialidad. Por tanto nos vimos obligados a utilizar alternativamente TLS y la sección 1.2.2 explica cómo crear los certificados X.509 de los servidores utilizando nuestra CA.

La posibilidad 5 (autocompletado de destinatarios) y 16 (uso en modo texto o gráfico) se refieren a la parte del cliente. El cliente en modo texto seleccionado fue mutt, y la sección 1.4.1 expone ejemplos de uso de todas las funcionalidades al completo disponibles en este sistema de correo; en el caso particular de MANAGESIEVE se utiliza una utilidad adicional, sieve-connect. La sección 1.4.2 presenta a Thunderbird/Icedove como el cliente gráfico de referencia y muestra una serie de pruebas de las funcionalidades más importantes, excepto para MANAGESIEVE ya que nuestro esquema de autenticación SASL-GSSAPI no es soportado por la extensión de Thunderbird que provee el soporte para este protocolo. Tanto Mutt como Thunderbird disponen de mecanismos para hacer búsquedas en LDAP. Dado que nuestra DN en LDAP para almacenar las cuentas de usuario autoriza a conexiones anonymous realizar búsquedas, ambos clientes proveen funcionalidad de autocompletado, como dijimos y como se presenta en las secciones referidas.

Explotación de la Infraestructura de Servicios Base

No vamos a detallar de nuevo el uso que hacen los servidores del servicio de correo de la infraestructura base. Los diagramas presentados hasta ahora en los demás servicios deberían haber dado ya una idea de cómo funciona el sistema y cómo lo haría para implementar lo comentado en el apartado inmediatamente anterior.

1.3.15. Cómo incorporar una nueva máquina a nuestro parque informático.

Anexo:	7

Un apartado del anexo sobre el servicio de Shell describe cómo desplegar la parte de cliente de nuestra infraestructura en una nueva máquina que incorporásemos a nuestro parque informático.

Concretamente, el apartado 1.6.2 del anexo detalla el proceso por completo para una máquina con Debian. Adicionalmente se expone un tipo de despliegue especial para laptops que visitan varias redes, "Roaming Setup".

El apartado 1.6.3 detalla parcialmente (no hay otro modo) el proceso que seguir para Windows.

El apartado 1.6.4 expone la situación general en SO POSIX y revela algunas claves específicas para MAC.

1.3.16. Cómo crear una nueva cuenta centralizada para un usuario nuevo de nuestra organización.

Anexo:	7

Otro apartado del anexo sobre el servicio de Shell, el 1.7, sumariza cómo crear todos los recursos necesarios (principal KERBEROS, id y volúmenes AFS, DN LDAP...) para que, ante la llegada de un nuevo integrante de nuestra organización, podamos ofrecerle una cuenta centralizada que le permita utilizar el parque informático.

1.4. Resultados Obtenidos

Tanto los usuarios finales como el staff de administración mejoran su situación de partida:

- *Usuarios de la organización*: básicamente disminuye el componente intrusivo que tiene la tecnología sobre el flujo de trabajo del usuario:
  - Disponibilidad: un miembro de la organización tendrá una cuenta de shell centralizada que le permitirá logearse y acceder a su ubicuo directorio home con su configuración y programas, todo ello cualquiera que sea la máquina desde la que hace el log-in (la de su despacho, la de un compañero, las de otra sede... en definitiva donde esté, está disponible).
  - Comodidad: Tras ese log-in inicial, podrá utilizar los demás servicios de forma autenticada pero sin que se le vuelva a interrumpir para pedir las credenciales (sistema SSO, Single Sign On).

- *Administradores*: Para el administrador de la infraestructura, la configuración de los servicios y cuentas se facilita al producirse en un único lugar accesible por red (ramas del árbol LDAP en general). También OpenLDAP, MIT Kerberos u OpenAFS permiten delegar de forma autorizada partes de esa configuración para que cada posible departamento pueda ocuparse de sus cambios. Además el centralizar los metadatos tiene el pequeño coste de tener que efectuar en cada ordenador una configuración mínima (mínima por ser la primera también la última vez) y que consiste básicamente en apuntar a la información y protocolos centralizados. Como contraparte, hay que aprender tecnologías que, aunque maduras y potentes, son complejas y requieren tiempo de entrenamiento para utilizarlas eficazmente. Por tanto:

    - Administración eficiente: el entorno crece pero la carga de trabajo del administrador, en general, no.
    - Administración flexible: posibilidad de delegaciones.
    - Independencia: la organización consigue su autoaprovisionamiento y a bajo coste[106].

1.5. Conclusiones

Hemos desplegado una infraestructura que cumple los requisitos de una organización del perfil mencionado en el apartado "Alcance del Trabajo", y lo hemos hecho conforme a como se desgranó en el apartado "Justificación". Por tanto hemos conseguido, punto por punto:

- Despliegue de un Servicio de Cuentas Centralizadas de Shell, gracias a sistemas como en los SO clientes como NSS y PAM (en Debian), KERBEROS, LDAP y AFS en los servidores.
- Despliegue de servicios típicos persona-persona, gracias a las implementaciones que Jabberd2, Exim4 y Dovecot proveen de protocolos sobradamente conocidos como XMPP, SMTP o IMAP.
- Despliegue de un sistema de ficheros distribuído, robusto, seguro y eficiente, sobre todo en lecturas, con OpenAFS.
- El sistema se presenta como escalable, puesto que distribuyen varias instancias de un servidor por las máquinas de la red; concretamente todos los servidores están desplegados por duplicado (dklab1 y dklab2). Excepto para gestores de bases de datos con esquema de replicación Master-Slave (DNS), integrar un nuevo servidor es tan sencillo como repetir los pasos dados en dklab2, en otra máquina. Bucardo no permite un Master adicional a los dos que ya existen.

[106] http://www.netways.de/uploads/media/Fabrizio_Manfred_Use_Distributed_Filesystem_as_a_Storage_Tier__ENG_.pdf ref

- Fiable: el historial y los casos de éxito del software utilizado (especialmente del "Magic Trio": OpenLDAP, MIT Kerberos, OpenAFS) nos permiten considerarlos especialmente sólidos y estables. Adicionalmente, la distribución de varios servidores aumenta la resistencia a la interrupción del servicio.

- Seguro: los servicios utilizan para las autenticaciones un protocolo dedicado, KERBEROS, que utiliza mecanismos como cifrado, firmado, comprobación de integridad de datos, notaría... que hacen de él un esquema fuerte. Además, las comunicaciones que se autorizan tras una autenticación exitosa son confidenciales gracias a SASL-GSSAPI y/o TLS (en el caso de NTP, Autokey v2 IFF).

- Multiplataforma: se dan indicaciones para que SO comunes se integren en mayor (GNU/Linux) o menor (Windows, no WDC) medida. El software cliente para servicios finales es tanto multiplataforma como popular (Pidgin, Mozilla Thunderbird,...).

- Se ha utilizado Software Libre disponible desde los repositorios de Debian; las libertades que otorga nos permitireon diseñar un parche para Jabberd2 y conseguir que funcionase completamente en nuestro despliegue, así como descubrir algún bug y publicar su correspondiente incidencia. Adicionalmente el coste directo ha sido mínimo.

A su vez hemos encontrado algunos problemas reseñables:

- Bucardo fue seleccionado para las replicaciones de PostgreSQL aplicadas a las tablas de Jabberd2, pero es un sistema transitorio mientras termina el desarrollo del soporte nativo síncrono multimáster de PostgreSQL.

- No disponemos de un cliente gráfico para MANAGESIEVE.

- La transferencia de zonas DNS cuando se usan vistas yerra.

- Las implementaciones de SASL-GSSAPI no dan buen soporte para el establecimiento de una capa de seguridad subsecuente a la autenticación. GSASL o Dovecot-SASL no lo implementan en absoluto. Cyrus-SASL sí lo hace, si bien usando un tamaño de clave algo pequeño y no configurable (ssf=56[107]).

1.6. Recomendaciones que se desprenden del estudio; líneas de trabajo futuro

Nos centraremos primero en nuevas necesidades, nuevos requisitos en que se realiza el despliegue, y cómo ésto inmediatamente implica que la Infraestructura de Servicios Base debe ser modificada. Adicionalmente recordaremos que el esquema de replicación actual de PostgreSQL deberá ser sustituído en un futuro próximo. En último lugar no

[107] http://www.openldap.org/lists/openldap-devel/200905/msg00034.html

hablaremos de que la situación cambie significativamente sino de mejoras o alternativas a los servicios finales.

1.6.1. Servicios Web; Federaciones

Citando el último párrafo del ejemplo con el que comenzábamos este documento:

"[...] el acuerdo es unánime: la práctica totalidad de las actividades de la organización encontrarán ventajas en utilizar la red y los servicios telemáticos. Pero la totalidad de actividades de la organización ya no son sólo las comunicaciones internas, sino también las que se producen al colaborar con otras organizaciones o al atender a individuos interesados en las actividades de la organización. Éso conlleva en el primer caso que debamos de alguna forma federar los sistemas de esas otras organizaciones con los nuestros, y en el segundo caso que personas no técnicas (y con equipos y posibilidad de instalar software variada...) tengan que usar una cierta variedad de servicios y aplicaciones con lo que, para hacerlo fácil, hay incluso quien opina que podría escojerse el navegador web como interfaz y puerta de entrada a todos ellos..."

Las situaciones que propone el ejemplo sólo podríamos resolverlas añadiendo una nueva capa a nuestra Infraestructura de servicios base por un lado y, óptimamente, añadiendo por otro una capa que ofrezca software como servicio web.

- CAS, WebAuth, Cosign o PubCookie son servicios adicionales que a través de diferentes arquitecturas y mecanismos utilizan HTTP Cookies y Kerberos TGT para proveer SSO a servicios web. Puesto que el mecanismo de cookies es inseguro, todos dependen de la creación previa de canales seguros para la transmisión de las cookies y otras precauciones. Son muy utilizados en centros académicos anglosajones, puede verse algunos estudios comparativos en:
  - http://www.umich.edu/~umweb/downloads/WebSSOImplementationComparision.pdf
  - http://www.jisc.ac.uk/uploaded_documents/CMSS-Gilmore.pdf
- Otras soluciones para la web pierden la ventaja de no necesitar nada especial en el cliente:
  - KX.509 se basa no en cookies sino en certifcados X.509 y, de nuevo, Kerberos. Un servicio de Autoridad Certificadora Kerberizada (KCA) expide un certificado X.509 como resultado de una autenticación KERBEROS exitosa. El certificado, que caduca en cuestión de horas, es la base de la autenticación de los demás servicios. Necesita software especial en el cliente, sin embargo.
  - SPNEGO nace como un subprotocolo de HTTP para negociar un mecanismo de autenticación incluyendo a KERBEROS (es por tanto una especie de SASL

para HTTP, concepto éste del que existe algún borrador pero poco más). SPENEGO es ideal para ofrecer servicios web a usuarios que también tienen Cuentas Cetralizadas de Shell en cuyo log-in se expide un TGT Kerberos. Hay varios estudios sobre Kerberos y la Web (con o sin SSO) que pueden consultarse en:

- http://kerberos.org/software/kerbweb.pdf
- http://workshop.openafs.org/afsbpw06/talks/wes-kerberos-on-web.pdf

Por su lado, también hemos mencionado que colaborar con otras organizaciones impone a nuestra infraestructura la necesidad de federarse con las de otras organizaciones. En la terminología que venimos usando, "Federación" implica que los sistemas de una organización permiten delegar la *autenticación* de un usuario a los sistemas de la organización a la que pertenezca éste y que está federada con nosotros. Y, aunque la autenticación se delega, la *autorización* no: es una organización la que define qué pueden hacer los usuarios de otras organizaciones federadas.

Efectivamente, se consigue que las organizaciones interactúen (por ejemplo que un investigador acceda a un paper en otra organización) a la vez que evita replicar identidades y el costo de mantenimiento que tiene ésto, sin mencionar el hecho de que sólo la organización de origen de un usuario sabe si éste sigue perteneciendo a la organización. Puede verse un video introductorio creado por JISC, la institución británica para el desarrollo de tecnologías de la información en el ámbito de la educación y la investigación:
http://www.youtube.com/watch?v=65ct24b9IAI

- Debemos mencionar que cuando todas las organizaciones participantes fuesen homogéneas y usasen un despliegue como el nuestro, MIT Kerberos dispone de una funcionalidad, "Cross Realm Authentication", que permite esa federación incipientemente.

  - http://web.mit.edu/kerberos/krb5-1.5/krb5-1.5.4/doc/krb5-admin/Cross_002drealm-Authentication.html

- Pero los principal Kerberos son tickets de formato fijo y no permiten especificar toda la metainformación que rellenaría el hueco entre sistemas heterogéneos. Es por ello que se utiliza una capa adicional donde (entre otras soluciones) un token SAML es pasado con esa información. Un token SAML no es otra cosa que un lenguaje XML (sus siglas corresponden a Security Assertion Markup Language) para expresar autorizaciones, identidades etc. Existen, por cierto, propuestas para integrar SAML en Kerberos y a la inversa, pero sólo propuestas. Puede verse, por otro lado, un ejemplo de uso de token SAML aplicado a Google Apps en:

  - http://code.google.com/googleapps/domain/sso/saml_reference_implementation.html

- Una solución web SSO para ofrecer capacidades de federación y que utiliza SAML es Shibboleth. Shibboleth por tanto extiende lo que dijimos para CAS, Cosign, etc pero para federaciones (aunque también tiene soporte para intrainstitucional

como aquellas). De hecho, inicialmente (v1.3) no ofrecía mecanismo de autenticación propio, pero podía utilizar a, por ejemplo[108], el de un CAS intrainstitucional para ello. Shibboleth es muy empleado en el mundo anglosajón, en España existe PAPI (Rediris) si bien ambos proyectos colaboran y se integran entre sí.

- http://shibboleth.internet2.edu/
- http://papi.rediris.es/

1.6.2. Soporte completo para Cuentas Centralizadas de Shell no POSIX (Windows)

Por ahora no es posible usar una infraestructura preexistente con OpenLDAP y MIT Kerberos como la nuestra y que, añadiendo Samba4, éste utilice a aquellos para autenticar y montar los home para máquinas con Windows. Es por ello que nosotros dimos una solución ciertamente parcial usando una cuenta local intermediaria etc.

Sí parece posible que, al contrario, MIT Kerberos pueda expedir tickets utilizando a Samba4 y su base de datos, alguien lo ha implementado alguna vez:

- http://www.samba.org/~idra/blog/id_006.html
- Progresión de Samba4 (en desarrollo -avanzado- aún) como Active Directory:
  - http://www.samba.org/samba/news/articles/abartlet_thesis.pdf

Por el momento, el único despliegue que conocemos y que aproxima de forma aún parcial pero algo más fiel ese soporte de cuentas centralizadas para Windows (con Software Libre y no con MS Active Directory) despliega un PDC (Primary Domain Controller) con Samba3, usa un esquema LDAP adicional "samba.schema" y lo administra con los scripts de smbldap-tools; consúltese en [8].

1.6.3. Soporte nativo en PostgreSQL de replicación síncrona multimaster

Bucardo era sólo una solución parcial, limitada y transitoria mientras ese soporte es terminado y liberado por los desarrolladores de PostgreSQL/Postgres-XC[109].

Por su lado, las líneas de trabajo siguientes plantean algunas posibles mejoras de los servicios finales.

[108] http://www.ja-sig.org/wiki/display/CASUM/Shibboleth-CAS+Integration

[109] http://michael.otacoo.com/postgresql-2/postgres-9-1-setup-a-synchronous-stand-by-server-in-5-minutes/

1.6.4. Asterisk

Asterisk es básicamente una PBX (de "Private Branch Exchange", centralita telefónica) por software, soportando tanto interfaces analógicas (a través del hardware adecuado) como protocolos de internet para VoIP/Videoconferencia. Comparándolo con nuestra solución de VoIP:

- Aporta una gran cantidad de subservicios de valor añadido tales como mensajes de voz, salas de multiconferencia, prueba de eco...
- Asterisk soporta el uso de LDAP, también SQL, para almacenar sus metadatos. Nosotros lo hemos probado parcialmente y funciona.

También hay aspectos negativos de calado:

- Los protocolos típicos, SIP e IAX, no soportan KERBEROS.
- La arquitectura distribuída parece ser reciente, en evolución y poco documentada. Es decir, no parece fácil desplegar varias instancias y que clientes no conectados a la misma se puedan comunicar entre sí[110].

Otra implementación, centrada esta vez en la robustez del servicio de un sólo protocolo (SIP), es Kamailio[111].

1.6.5. Ejabberd

En los últimos años ha aumentado indirectamente el uso de XMPP, gracias a que grandes plataformas web como las de Facebook[112], Twitter[113] y otros[114] lo utilizan para sus servicios de mensajería. Además, ha trascendido que en esos despliegues se ha utilizado como base a la implementación se servidor XMPP Ejabberd, escrita en Erlang/OTP. Parece ser que el lenguaje y el entorno que lo acompaña, con su modelo de concurrencia basado en paso de mensajes, son especialmente adecuados para escalar masivamente sistemas de mensajería. Comparativamente con Jabberd2:

- Un despliegue con Ejabberd escala mejor y no tiene punto débil (Single Point of Failure), al contrario que Jabberd2 (su componente router no era clusterizable, hecho éste confirmado por su actual desarrollador[115]).

[110] http://leifmadsen.com/sites/default/files/
Why_Cluster_An_Introduction_to_Asterisk_Clustering_and_Database_Integration
_AstriCon_2008_LMadsen.pdf
ref

[111] http://www.kamailio.org/w/features/

[112] http://www.process-one.net/en/ejabberd/article/facebook_chat_is_developed_in_erlang/

[113] http://www.pixzone.com/blog/217/erlang-helps-rails-to-scale/

[114] http://www.process-one.net/en/imtrends/

[115] No obstante, en Septiembre de 2012 se está originando una discusión en la lista de correos del proyecto precisamente para cambiar ésto. Consúltese:
http://www.mail-archive.com/jabberd2@lists.xiaoka.com/msg01908.html

- Tiene soporte nativo para salas de conferencia (multichat) así como transportes HTTP. Tradicionalmente incluía una pasarela a redes IRC, aunque en la rama 3.X ésto puede cambiar.
- Incluye una interfaz web de administración.

Pero para plantearnos sustituir Jabberd2 por Erlang, éste debiera solucionar algunas de sus debilidades:

- Sólo la rama 3.x (a principios de 2012 en estado de madurez alpha aún), dispone de soporte para SASL-GSSAPI y por tanto KERBEROS.
- El soporte para LDAP es más pobre que el de Jabberd2. Especialmente, no parece soportar la creación del roster al vuelo.

1.6.6. Evaluación de AFS como medio de almacenamiento del correo

AFS nos permitía que todos los servidores de correo tuviesen el mismo comportamiento. Ésto era así porque entonces todos tenían acceso al almacén de correos al ser AFS en red. Otra posibilidad es no utilizar AFS sino especializar un grupo de servidores SMTP en el almacenamiento y otro en la comunicación de los clientes. Si tras atender al cliente el correo ha de ser almacenado, los segundos transpasan el correo a los primeros.

El segundo esquema parece más complejo de desarrollar y mantener, pero en según qué casos puede ser necesario frente al uso de AFS por motivos de desempeño. No obstante, en nuestro caso no sólo el desempeño fue criterio y consideramos decisiva la oportunidad que nos brindaba el almacenamiento de correos para mostrar cómo desplegar servidores que necesitan usar AFS, asunto no trivial.

- http://wiki.openafs.org/AFSLore/AdminFAQ/#3.52%20%20Is%20it%20a%20good%20idea%2
  3.52 Is it a good idea to store mail in AFS?
- www.howtoforge.com/setting-up-a-mail-server-using-exim4-clamav-dovecot...

Bibliografía

[1] *http://www.ipligence.com/worldmap/.*

[2] www.n economia.com. *Perspectivas Económicas y Empresariales Diciembre 2011*, **2011**.

[3] www.n economia.com. *Consecuencias de las TIC en la economía*, **2002**.

[4] www.cincodias.com. *Zentyal proyecta abrir este año sucursales en Europa y América.*

[5] Stallings, W. *Fundamentos de Seguridad en Redes. Aplicaciones y Estándares*, **2004**, p 108.

[6] Mills, D. L. *The Autokey Security Architecture, Protocol and Algorithms*, **2006**.

[7] Milicchio, F.; Gehrke, W. A. *Distributed Services with OpenAFS for Enterprise and Education*, **2007**, p 71.

[8] Milicchio, F.; Gehrke, W. A. *Distributed Services with OpenAFS for Enterprise and Education*, **2007**, pp 161–194.