UNIVERSIDAD REY JUAN CARLOS DE MADRID

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA Y TECNOLOGÍA ELECTRÓNICA

TESIS DOCTORAL

Arquitecturas de Seguridad para Sistemas Distribuidos Fraccionables, Dinámicos y Heterogéneos con Aplicación a la Computación Ubicua

Autor:

Enrique Soriano Salvador

Ingeniero en Informática

Director:

Francisco J. Ballesteros Cámara

Doctor en Informática

Madrid, 10 de julio de 2006

Enrique Soriano Salvador
Departamento de Ingeniería Telemática y Tecnología Electrónica
Universidad Rey Juan Carlos
Móstoles 28933 Madrid
E-Mail: esoriano@lsub.org
WWW: http://lsub.org/who/esoriano

Yo, Francisco J. Ballesteros Cámara, Director de la Tesis Doctoral titulada *Arquitecturas de seguridad para sistemas distribuidos fraccionables, dinámicos y heterogéneos con aplicación a la computación ubicua* realizada por Enrique Soriano Salvador, autorizo su defensa.

Francisco J. Ballesteros Cámara
Director de la Tesis Doctoral

Madrid, de de 2006

TESIS DOCTORAL:	Arquitecturas de seguridad para sistemas distribuidos fraccionables, dinámicos y heterogéneos con aplicación a la computación ubicua
AUTOR:	Enrique Soriano Salvador
DIRECTOR:	Francisco J. Ballesteros Cámara

El tribunal nombrado para juzgar la Tesis arriba indicada, compuesto por los siguientes doctores:

PRESIDENTE:	Dr.
VOCALES:	Dr.
	Dr.
	Dr.
SECRETARIO:	Dr.

acuerda otorgarle la calificación de

Madrid, de de 2006

El Secretario del Tribunal

"There is no such thing as security through lack of information"

Emmanuel Goldstein.

Prefacio

El trabajo aquí expuesto se ha desarrollado en el Laboratorio de Sistemas del Grupo de Sistemas y Comunicaciones de la Universidad Rey Juan Carlos.

El presente documento se ha creado en un entorno de computación Plan B, editado con Omero, y tipografiado con LaTeX en un sistema Debian GNU/Linux.

Agradecimientos

En primer lugar, me gustaría dar gracias a María y a mi familia por ser tan pacientes y animarme a seguir adelante. Sin vosotros, este trabajo no habría sido posible.

También me gustaría agradecer a mis compañeros del Grupo de Sistemas y Comunicaciones el apoyo y el magnífico ambiente de trabajo que han creado.

A Gorka por sus comentarios, ideas y sugerencias, que me han resultado de gran utilidad.

Por último, quiero expresar mi admiración y agradecimiento a Nemo, por haber dirigido con tanta atención este trabajo y haber dedicado una gran cantidad de tiempo a revisarlo.

Enrique
Madrid, julio de 2006

Resumen

La computación ubicua está convirtiéndose en una realidad, e implica un modelo muy cercano a un sistema distribuido fraccionable, dinámico y heterogéneo.

Nuestra visión de un entorno ubicuo es más cercana a un modelo *Peer-to-Peer* que al modelo clásico Cliente/Servidor. Los nodos son dispositivos móviles (ordenadores de mano, teléfonos, ordenadores portátiles, etc.) o no móviles (pantallas, pizarras electrónicas, estaciones de trabajo etc.) conectados mediante distintos tipos de tecnologías de red (Ethernet, Bluetooth, Wi-Fi etc.). Estos dispositivos normalmente pertenecen a un usuario, y eventualmente otros usuarios desean o necesitan utilizarlos.

El sistema es fraccionable por que pueden surgir situaciones en las que dos dispositivos se puedan conectar entre sí, pero no tengan conexión con el resto del sistema. Por ejemplo, dos usuarios se encuentran en el aparcamiento donde no hay ningún tipo de conexión con el exterior, pero ellos pueden comunicarse mediante la tecnología Bluetooth de sus dispositivos.

El sistema es dinámico, ya que los dispositivos aparecen y desaparecen constantemente (principalmente los dispositivos móviles). Los usuarios, igual que los dispositivos, aparecen y desaparecen constantemente.

El sistema es heterogéneo, debido a que los dispositivos utilizados por los usuarios son muy variados, desde ordenadores personales a electrodomésticos.

En un escenario como este se necesita una arquitectura de seguridad que ofrezca autenticación, confidencialidad, integridad y control de acceso. Creemos que ninguno de los esquemas propuestos para sistemas distribuidos y para en-

tornos ubicuos satisface las necesidades implícitas en este modelo:

- La independencia de servicios centralizados o servidores de autenticación.
- La facilidad de uso de los mecanismos de seguridad.
- La necesidad de soportar desconexiones y delegación.de recursos.
- Simplicidad del modelo.
- Extensibilidad de la arquitectura.
- Consideración de los límites de consumo de energía y de procesamiento en los dispositivos móviles.

Este trabajo presenta las ideas generales para construir una arquitectura de seguridad centrada en el humano para espacios inteligentes. También presenta el diseño, los protocolos y los detalles de implementación para el sistema operativo Plan B de tres arquitecturas incrementales que proporcionan entrada única al sistema (*Single Sign-On*), compartición de terminales y compartición de dispositivos a los usuarios de un espacio inteligente real.

Abstract

Ubiquitous computing is becoming real. It implies a model close to a dynamic, partitionable and heterogeneous distributed system. Our vision of pervasive computing is closer to Peer-to-Peer network systems than to classic client/server open network systems. In that model, the nodes can be mobile devices (mobile phones, handhelds, etc.) or non mobile devices (workstations, servers, panels, etc.), and they can be interconnected by different kinds of network technologies (ethernet, wi-fi, bluetooth). These devices belong to users, and users eventually need or want to use devices that they do not own.

The system is partitionable, because it has to be available at locations where several principals can communicate but there is no connection with the rest of the system.

The system is dynamic too, because devices, as well as principals, come and go.

The system is heterogeneous due to the wide range of devices that exist in an ubiquitous environment.

In a scenario like this, a new security architecture is needed. We think that neither classic security schemes that are used in common open network systems nor security schemes proposed for ubiquitous environments comply with some of the requirements that are implicit to this model:

- Independence of centralized services or authentication servers.
- Ease of use and the non obtrusiveness of the system.

- Support for disconnections and delegation.
- Simplicity of the scheme.
- Extensibility of the system.
- Minimizing power consumption and processing to operate with limited mobile devices.

This PhD. thesis offers the basic principles to build a human centered security architecture for today's smart spaces. It also describes the design and the implementation for the Plan B operating system of three security architectures that offer real Single Sign-On, terminal sharing and device sharing between users in our smart space.

Índice general

Capítulo 1

Introducción

La computación ubicua [193] se está convirtiendo en una realidad. Este nuevo paradigma implica un modelo de computación que es equiparable a un sistema distribuido fraccionable, altamente dinámico y heterogéneo. En este tipo de sistemas, el problema de la seguridad no es trivialmente resoluble [178; 179; 36].

La seguridad es un aspecto crítico para un entorno de estas características, ya que es necesario proporcionar autenticación a los usuarios y fijar una política de control de acceso a los recursos del entorno. Además, hay que ofrecer confidencialidad e integridad en las comunicaciones. El presente trabajo va en esta linea, y tiene como objetivo plantear arquitecturas de seguridad genéricas que tengan aplicación en entornos de computación ubicua.

El escenario considerado corresponde a un sistema distribuido **fraccionable**, debido a que múltiples usuarios pueden estar conectados entre sí, pero al mismo tiempo pueden carecer de conexión con el resto del sistema. A la vez, el sistema es altamente **dinámico**, ya que los usuarios aparecen y desaparecen constantemente: un entorno ubicuo es un entorno físico en el que los usuarios entran y salen, y con ellos sus recursos (datos y dispositivos). Por último, el sistema es altamente **heterogéneo** debido a que los dispositivos que se incluyen en el entorno ubicuo pueden pertenecer a distintas familias de *hardware*, como ordenadores personales, estaciones de trabajo, dispositivos móviles, sensores o incluso electrodomésticos. A su vez, estos dispositivos pueden estar ejecutando distintos tipos de *software* sobre distintas clases de sistemas operativos.

Un entorno de estas características no es concebible sin mecanismos que aporten seguridad. En este tipo de entorno los usuarios comparten sus dispositivos, que podrían usarse sin permiso o dañarse. Los usuarios también comparten datos que podrían corromperse, robarse o eliminarse. Además, se podría violar el derecho a la privacidad de las personas, debido a que en un sistema ubicuo se pone a disposición de los usuarios y de los programas información sobre contexto físico del entorno y la localización de individuos y objetos.

Hoy en día disponemos de los medios necesarios para crear entornos ubicuos en los que los usuarios (habitualmente humanos) puedan compartir sus recursos (normalmente decenas de dispositivos) con otros usuarios en los que confían. Esta cesión de recursos debe realizarse de la forma más transparente posible, haciendo desaparecer a las computadoras de una forma similar a la que describe Weiser en el artículo seminal "The Computer for the 21st Century" [193].

Los datos transmitidos deben mantener su integridad y la comunicación entre dispositivos debe ser confidencial, especialmente cuando se utilizan tecnologías de comunicación inalámbrica. Por otra parte, las entidades del sistema (dispositivos y usuarios) deben poder autenticarse entre sí. Por último, se debe controlar el acceso a los recursos *software* y *hardware*.

Como se detallará más tarde en el capítulo 2, correspondiente al estado del arte, la mayoría de los esquemas tradicionales de seguridad propuestos para sistemas distribuidos dependen de entidades centralizadas, como servidores de autenticación y entidades certificadoras. Varias arquitecturas de seguridad aplicables a sistemas ubicuos planteadas en la literatura [8; 6; 9] derivan a su vez en este tipo de arquitecturas clásicas [182; 86]. Otros sistemas propuestos para sistemas ubicuos también contactan con servicios centralizados para obtener información sobre el contexto del entorno.

Las entidades centralizadas introducen el siguiente problema: los usuarios que necesitan usar un dispositivo de forma segura necesitan a su vez estar en linea con el servicio centralizado. Esta restricción hace que los usuarios que se encuentran en una partición sin conexión con el servicio centralizado queden incomunicados debido a que no pueden establecer conexiones seguras. Además, los sistemas basados en entidades centralizadas dependen de un conjunto de usuarios especializados que se encargan de su configuración y administración. Por último, la inclusión de nuevos elementos (ya sean humanos o máquinas) en este tipo de sistemas provoca cierto coste administrativo.

Usualmente, los entornos ubicuos se conciben como sistemas cliente/servidor. En ese tipo de sistemas, los humanos acceden a los servicios que ofrece el

entorno mediante aplicaciones que actúan siempre como programa cliente. Estos programas cliente solicitan servicio a un programa que actúa exclusivamente como servidor, y que por lo general es único.

Nuestra visión de un entorno ubicuo es más cercana a un sistema *Peer-to-Peer* [167] que a un sistema distribuido clásico de tipo cliente/servidor:

- En un entorno ubicuo existen múltiples tipos de dispositivos, móviles o fijos, que almacenan y procesan datos pertenecientes a su dueño, normalmente un humano. Estos dispositivos suelen estar conectados entre sí mediante distintas tecnologías de red tradicionales o inalámbricas. El número de dispositivos que posee un humano es tal vez de decenas, pero no de centenas de ellos. Muchas veces no hay un cliente o un servidor claro, sino que depende del tipo de interacción.

- Eventualmente, un usuario del entorno ubicuo puede querer o necesitar usar los recursos de otros usuarios, ya sean datos o dispositivos *hardware*. Ciertas tareas realizadas por un usuario dependen de la cooperación entre diversos recursos, y algunos de estos recursos pueden no ser de su pertenencia [180]. Por esta razón, el humano debe ser capaz compartir estos recursos de una forma sencilla, de tal forma que no proporcione obstrucción al usuario.

Por lo general, las arquitecturas propuestas en la literatura para entornos ubicuos no se ajustan a un entorno *Peer-to-Peer*.

Además, muchos de estos sistemas se apoyan en el uso de un determinado *middleware*, *framework* o biblioteca de programación [72; 173; 64; 54; 156; 92]. El uso de esas herramientas introduce problemas de compatibilidad con el *software* existente. En esos casos, las aplicaciones antiguas necesitan fuertes modificaciones o dependen de *software* adicional (*wrappers*) para poder interaccionar con el sistema.

Por otra parte, los sistemas planteados para ofrecer entrada única al sistema (*Single Sign-On*) no funcionan en entornos en los que el usuario

trabaja con múltiples dispositivos simultáneamente [24; 144; 128; 170; 41; 92]. En estas situaciones, el usuario necesita autenticarse al menos una vez por cada dispositivo que está utilizando, de tal forma que no existe una entrada única al sistema ubicuo.

Por estas razones, es necesario diseñar e implementar nuevas arquitecturas de seguridad para entornos de computación ubicuos, ya que las arquitecturas existentes aportan centralización, complejidad, obstrucción e incompatibilidades.

1.1. Objetivos de la investigación

El propósito de este trabajo es diseñar una nueva arquitectura de seguridad *Peer-to-Peer* centrada en el humano que se ajuste a los sistemas descritos anteriormente. A partir de ahora, nos referiremos a la arquitectura como **SHAD** (*Secure Human-centered Architecture for Distributed-systems*).

La idea principal de SHAD es **centrar la seguridad en los humanos**. Los entornos ubicuos son sistemas físicos en los que los humanos interactúan entre ellos, por tanto, en nuestro sistema el humano es el *Peer*. Los mecanismos de seguridad deben ser cómodos para los usuarios, que no pueden estar introduciendo contraseñas o utilizando dispositivos de autenticación constantemente. A continuación se especificarán los objetivos principales que persigue el diseño de SHAD, que incluyen la entrada única al sistema (*Single sign-On*) y la cesión de recursos.

1.1.1. Independencia de entidades centralizadas

SHAD debe ser ágil en un entorno *Peer-to-Peer*, y por lo tanto no puede basarse en entidades centralizadas como servidores de autenticación, servidores de claves o entidades certificadoras. En un sistema ubicuo, los humanos se mueven constantemente acarreando sus dispositivos móviles con ellos. No es descabellado pensar que en algún momento estos humanos necesitarán usar

sus dispositivos móviles conjuntamente con otros dispositivos en localizaciones aisladas del resto del sistema, usando únicamente su PAN (*Personal Area Network*) [83].

Por ejemplo, un usuario del sistema podría querer utilizar un recurso de otro usuario en el aparcamiento subterráneo del edificio en el que ambos trabajan. En esa localización lo más probable es que no exista ninguna forma de conectarse a la red del lugar de trabajo. Una arquitectura en la que los elementos a autenticar dependen de servicios centralizados no puede afrontar una situación como esa. Un objetivo de SHAD es proporcionar seguridad en escenarios como el descrito, de tal forma que los usuarios sean independientes de los servicios centralizados del entorno.

1.1.2. Facilidad de uso y no obstrucción al usuario

La facilidad de uso y la no obstrucción al usuario son requisitos indispensables para un entorno ubicuo. Un sistema sencillo de utilizar por parte de los usuarios es más seguro que un sistema complejo [165]. En sistemas complicados, los usuarios tienden a utilizar incorrectamente las herramientas, ya sea por error o por desidia. Cuando las herramientas de seguridad se utilizan de una forma incorrecta, se pone en peligro la seguridad del sistema. Por esa razón, la arquitectura de seguridad debe ser sencilla de utilizar y lo más transparente posible para el usuario, aunque debe ser trazable y predecible. Un objetivo de SHAD es hacer que el usuario pueda compartir sus recursos de forma segura realizando tareas de administración simples en sus dispositivos.

En un entorno ubicuo, el usuario no debería interrumpir sus tareas para proporcionar al sistema ningún tipo de información y los procesos deben estar automatizados en el mayor grado posible con el fin de no distraerle [36; 59].

La autenticación a través de contraseñas aporta un gran grado de obstrucción al usuario, ya que normalmente debe introducirlas por un dispositivo de entrada (comúnmente un teclado). Por otra parte, los usuarios tienden a escoger

contraseñas poco seguras y, por lo general, es difícil hacer que estos mantengan una cierta disciplina a la hora de gestionarlas [10, Capítulo 3] [42].

En el caso de los dispositivos de autenticación no inalámbricos (p.e. Smartcards [148] o Ibuttons [76]), el grado de obstrucción disminuye aunque sigue existiendo. En estos casos, el usuario está obligado a introducir el dispositivo de autenticación en un lector, lo que implica una autenticación explícita por su parte.

1.1.3. Soporte de desconexiones

SHAD debe soportar desconexiones y delegación. Los usuarios del sistema tienen movilidad y por tanto entrarán y saldrán de este constantemente. Como el sistema debe ser lo más fiable posible y la salida del entorno por parte de los usuarios es imprevisible, se deben soportar las desconexiones.

Por ejemplo, supongamos que un usuario está utilizando un dispositivo de otro usuario y en un momento dado, éste abandona el sistema. El usuario que estaba utilizando el dispositivo tiene que ser capaz de seguir haciéndolo hasta que acaben las tareas en curso. De otra forma, el sistema ubicuo perdería disponibilidad.

1.1.4. Interoperabilidad

Como ya hemos mencionado en la introducción, algunas de las estrategias seguidas en varios de los sistemas planteados en la literatura provocan incompatibilidades con el *software* antiguo. En estos casos, los programas deben modificarse sustancialmente o implementarse desde cero para poder hacer uso de los mecanismos del nuevo entorno. En algunos sistemas, estas modificaciones pueden resultar extremadamente costosas [188].

SHAD deberá ser lo más compatible posible con las aplicaciones antiguas. Estas aplicaciones sólo requerirán pequeñas modificaciones para poder interaccionar con la arquitectura de seguridad. Dichas modificaciones se podrán

llevar a cabo independientemente del lenguaje de programación en el que esté escrito el programa y de las tecnologías que utilice.

1.1.5. Simplicidad

Tanto el diseño como la implementación de SHAD deberán perseguir la simplicidad. La simplicidad facilita la trazabilidad del sistema. También aumenta su robustez.

La literatura nos muestra que un diseño basado en la simplicidad y en el principio *extremo a extremo* hace que el sistema sea altamente adaptable [159].

1.1.6. Extensibilidad

SHAD debe ser extensible, para poder crecer y adaptarse a las necesidades de un espacio inteligente.

La arquitectura tendrá que permitir la inclusión de nuevos tipos de dispositivos en el sistema. También deberá ser extensible respecto a los protocolos y al tipo de aplicaciones a las que ofrece servicio.

1.1.7. Consideración de las limitaciones de procesamiento y energía de los dispositivos móviles

La proliferación de dispositivos móviles en los últimos años ha sido considerable. Actualmente existen equipos como teléfonos móviles, ordenadores de bolsillo, agendas electrónicas, ordenadores portátiles y otros tipos de dispositivos con capacidad de proceso y capacidades de comunicación inalámbricas. Algunos de estos dispositivos tienen una capacidad de procesamiento muy limitada [7].

Otro problema es el consumo de energía, que determina la disponibilidad de los sistemas móviles [94]. Las baterías de estos dispositivos no proporcionan tanta autonomía como sería deseable. El uso del procesador y la transmisión de datos son dos de los principales factores que determinan el tiempo de autonomía de los

dispositivos móviles. Por otra parte, la limitación de energía de los dispositivos móviles implica nuevos tipos de ataques de denegación de servicio [179].

Por estas razones, SHAD deberá seguir un diseño que intente minimizar la carga de procesamiento y la cantidad de transmisiones de datos en los dispositivos móviles.

1.2. Organización de este documento

El resto del documento se organiza como se describe a continuación. El capítulo 2 describe el estado actual de la cuestión, haciendo una breve revisión de las arquitecturas de seguridad propuestas tanto para sistemas distribuidos de tipo cliente/servidor como para entornos ubicuos, entornos *Grid*, redes *Peer-To-Peer* y sistemas de entrada única al sistema (*Single Sign-On*).

En el capítulo 3 presenta las principales ideas de SHAD y distintos escenarios que ilustran sus cualidades. En este capítulo describimos los fundamentos que permitirán el diseño y la implementación de tres arquitecturas distintas: entrada única al sistema, compartición de terminales y compartición de dispositivos.

Estas arquitecturas se describen en los tres siguientes capítulos. El capítulo 4 describe la arquitectura que ofrece una entrada única real al sistema para un espacio inteligente. El capítulo 5 describe una arquitectura que permite a los usuarios compartir sus terminales de una forma segura. El capítulo 6 presenta la arquitectura para compartir dispositivos en el espacio inteligente. Los tres capítulos presentan los aspectos relevantes de su implementación para el sistema operativo Plan B.

Por último, el capítulo 7 expone las principales conclusiones y aportaciones del trabajo efectuado, y propone posibles trabajos futuros.

Capítulo 2

Estado del arte

Este capítulo muestra brevemente el contexto actual en el que se encuentran las arquitecturas de seguridad para entornos distribuidos. Para ello, realiza una rápida presentación de algunos de los esquemas de seguridad y las arquitecturas propuestas para sistemas distribuidos clásicos, sistemas ubicuos y sistemas *Peer-to-Peer*. También describe brevemente distintos tipos de dispositivos móviles propuestos para autenticar usuarios y su esquema de uso, así como los distintos sistemas de *Simple Sign-On* planteados en la literatura y en distintas soluciones comerciales.

2.1. Seguridad en sistemas distribuidos cliente/servidor

2.1.1. Kerberos

Kerberos [182] es un esquema de seguridad propuesto para sistemas distribuidos abiertos de tipo cliente/servidor que proporciona a los usuarios autenticación y confidencialidad en redes inherentemente inseguras.

Kerberos se basa en criptografía de clave simétrica. Toda entidad del sistema tiene asignada una clave privada. Tanto los clientes como los servidores comparten su clave privada con un servicio especial ofrecido por un nodo denominado centro de distribución de claves (KDC). Por tanto, en el proceso de autenticación entran en juego al menos tres entidades: el cliente, el servidor y el centro de distribución de claves.

La autenticación se basa en el algoritmo de Needham-Schroeder [123] (ver figura 2.1). Kerberos utiliza resguardos o credenciales (tickets) para autenticar a los usuarios del sistema. Los tickets son datos cifrados con la clave secreta de la entidad para la que se crean. Cuando un cliente tiene que autenticarse, solicita al KDC un ticket generado específicamente para el nodo servidor. El KDC responde a ese mensaje con un mensaje cifrado con la clave secreta del cliente (Kc). El mensaje contiene:

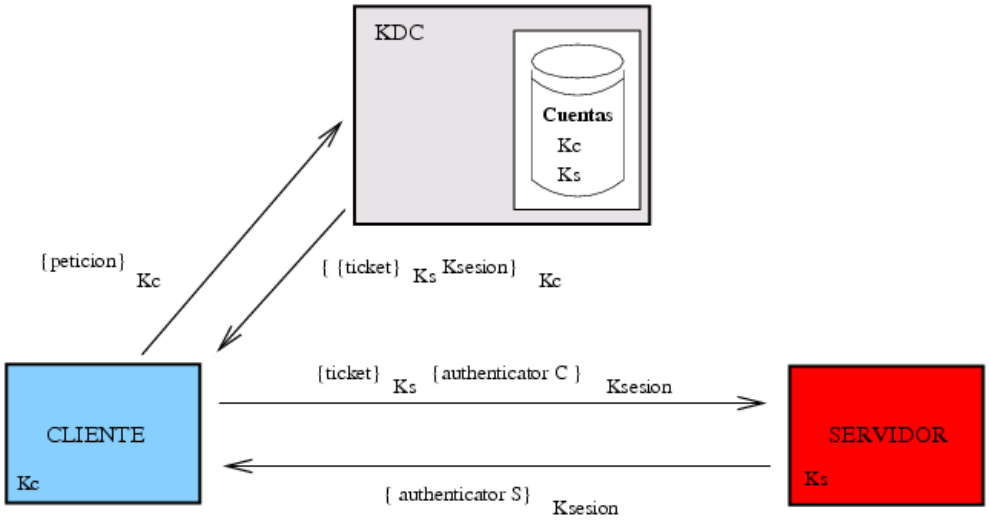

Figura 2.1: *Esquema básico de Kerberos.*

- Una clave de sesión generada por el KDC ($Ksesion$).
- Un ticket generado para el servidor, que contiene a su vez la clave de sesión junto con otros datos (identificador del cliente, dirección del cliente y el tiempo de vida del resguardo). Este resguardo está cifrado con la clave secreta del servidor (Ks).

Después, el cliente genera una estructura de datos denominada *authenticator* ($authenticatorC$). Esa estructura contiene una marca de tiempo generada por el cliente y su identificador. El *authenticator* se cifra con la clave de sesión generada por el KDC. Una vez generado el *authenticator*, el cliente lo envía al servidor junto con el ticket que le entregó el KDC.

El servidor puede comprobar la validez del resguardo descifrándolo con su clave secreta y verificando el tiempo de vida y las marcas de tiempo. También puede validar la procedencia del resguardo descifrando el *authenticator* con la clave de sesión y verificando el identificador y su marca de tiempo.

Por último, el servidor envía al cliente otro *authenticator* cifrado con la clave de sesión que incluye su identificador y la marca de tiempo anterior ($authenticatorS$). En ese momento, el cliente está en condiciones de comprobar

que el servidor es quien dice ser.

Kerberos es un sistema basado en varios servicios centralizados:

- El servicio de distribución de claves (KDC), que mantiene una base de datos con las claves de los usuarios (humanos y programas).
- El servicio suministrador de resguardos (TGS), que genera resguardos para acceder a los servicios.

Normalmente, los clientes de Kerberos obtienen un resguardo especial llamado TGT (*Ticket-granting ticket*). Este resguardo sirve para solicitar nuevos resguardos al TGS. De esa manera, el cliente no usa constantemente su clave secreta. El TGT lo obtienen siguiendo el protocolo descrito anteriormente con el TGS como servidor.

Kerberos ofrece tres niveles de seguridad basados en las claves de sesión: una autenticación por sesión, sucesivas autenticaciones en la misma sesión y autenticación más confidencialidad.

Consideramos que Kerberos no es apropiado para aportar seguridad en sistemas ubicuos, debido a su gran dependencia de servicios centralizados. Todas las entidades que necesitan autenticarse deben estar en linea con los servicios centralizados, por tanto no pueden comunicarse en ubicaciones donde no hay disponible ningún tipo de conexión con estos servicios. Por otra parte, Kerberos es complejo de administrar debido a que un único administrador tiene que encargarse de dar de alta y de baja a todos los usuarios y servicios. Además, las aplicaciones tienen que estar modificadas específicamente para usar el esquema de Kerberos.

Otro problema relacionado con este tipo de sistemas es que los servidores de autenticación son puntos únicos de fallo para todo el entorno ubicuo. Por lo tanto, hay que recurrir a técnicas de tolerancia a fallos (replicación de la base de datos y de los servidores expendedores de claves) para mantener el sistema ubicuo operacional cuando esos nodos no pueden dar servicio. Por ejemplo, esto podría ocurrir si fallan los servidores o se producen ataques de denegación de

servicio. Estas técnicas requieren además de una construcción que no rompa el sistema (por ejemplo, el sistema de *backup* también tiene que ser seguro).

2.1.2. Infraestructuras de clave pública (PKI)

En ciertas aplicaciones distribuidas, como por ejemplo en el World Wide Web, el uso de certificados basados en infraestructura de clave pública o PKI (*Public Key Infrastructure*) está ampliamente extendido. Los **certificados X.509** [74] son los más populares dentro de las PKI. Estos certificados son básicamente claves públicas pertenecientes a las entidades que están firmadas digitalmente con la clave privada de una entidad certificadora de confianza.

Cuando el usuario recibe un certificado, podrá comprobar que está firmado digitalmente con la clave privada de una entidad certificadora de la que se fía. Esta comprobación se puede realizar usando la clave pública de la entidad certificadora, conocida por todos los usuarios (que suponen que es correcta y de confianza). De esa forma, el usuario se fiará de la clave pública del certificado recibido.

Mediante el uso de un esquema de nombrado especial, como el estándar X.500 [23], se pueden ordenar las entidades y usuarios en una jerarquía de dominios, de tal forma que las entidades de dominios superiores firman digitalmente los certificados de los dominios inferiores.

Los certificados X.509 contienen, además de la firma de la entidad certificadora, otros campos de información:

- La versión de X.509 del certificado.
- Un número de serie que distingue al certificado de otros emitidos para el mismo propietario.
- El tipo de algoritmo de cifrado que se utiliza para firmar el certificado.
- El identificador del emisor del certificado, que normalmente es una entidad certificadora. Este identificador sigue el estándar X.500.

- El periodo de validez del certificado.
- El identificador del propietario del certificado, también en el estándar X.500.
- La clave pública del propietario del certificado.
- El uso que se le puede dar al la clave, por ejemplo sólo firmar y no cifrar.
- Otros nombres que se le pueden asociar a esta clave (nombres de DNS, de correo electrónico, etc.).

Los certificados X.509 son difíciles de entender debido a su complicada sintaxis. El cuadro 2.1 muestra un certificado de ejemplo. Varios autores [41; 155] consideran que las estructuras de datos en general, y en particular las relacionadas con la seguridad, deben ser fácilmente legibles por los humanos. De esta forma se facilita su depuración y la administración de los sistemas que las utilizan. Por ejemplo, **SDSI** [155] propone una alternativa al los certificados X.509 en la que usan estructuras de datos más simples.

Por otra parte, los algoritmos de clave pública utilizados comúnmente para este tipo de certificados no son la mejor opción para dispositivos con limitaciones de procesamiento y de consumo de energía.

Dentro de los sistemas basados en PKI se encuentra **SSL** [40], el protocolo de seguridad que más se usa en la actualidad. Este protocolo se apoya en parte en una infraestructura de clave pública, como veremos a continuación.

SSL se sitúa justo encima del protocolo de transporte de una pila de comunicaciones (p.e. TCP/IP), de tal forma que proporciona autenticación y confidencialidad mediante el cifrado y descifrado de los datos retransmitidos por el nivel de aplicación. Las aplicaciones no se deben modificar para añadir mecanismos de autenticación y cifrado de datos, ya que SSL actúa como una capa más de la pila de protocolos. De esta forma, se añade seguridad siguiendo la filosofía de extremo a extremo de TCP. El protocolo SSL emplea varias fases para proporcionar sus servicios:

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 7829 (0x1e95)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
                OU=Certification Services Division,
                CN=Thawte Server CA/Email=server-certs@thawte.com
        Validity
            Not Before: Jul  9 16:04:02 1998 GMT
            Not After : Jul  9 16:04:02 1999 GMT
        Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
                 OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                    33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                    66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                    70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                    16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                    c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                    8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                    d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                    e8:35:1c:9e:27:52:7e:41:8f
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
        92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
        ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
        d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
        0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
        5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
        8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
        68:9f
```

Cuadro 2.1: *Ejemplo de un certificado X.509 para una clave RSA.*

1. **Negociación de los algoritmos** a usar entre los extremos. En esta fase los extremos se tienen que poner de acuerdo en los algoritmos que van a usar para:

   - **Cifrado asimétrico**: RSA [154], Diffie-Hellman [49], DSA [119] ó Fortezza [56].
   - **Cifrado simétrico**: RC2 [153], RC4 [164], IDEA [95], DES [118], Triple-DES [118] ó AES [121].
   - **Resúmenes**: MD5 [152] ó SHA-1 [120].

2. **Intercambio de claves** públicas y autenticación basada en certificados PKI.

3. **Cifrado de la comunicación** basada en algoritmos simétricos.

TLS [48] es el protocolo sucesor de SSL, y es un estándar de la *Internet Engineering Task Force* [77]. En concreto, TLS se basa en la versión 3.0 de SSL. Estos protocolos, aunque son muy parecidos, no son compatibles entre sí.

Tanto TLS como SSL ofrecen seguridad a los protocolos de aplicación que se usan comúnmente en entornos de red. Por ejemplo, la versión segura del protocolo HTTP se denomina HTTPS [149] y puede usar los dos protocolos (TLS y SSL). Otro ejemplo es SSH [199], el acceso remoto a consola que reemplaza al protocolo Telnet [142; 141].

El principal problema que tiene aplicar un esquema como el que se utiliza en el WWW en un entorno como el descrito en la introducción es que se depende en todo momento de las entidades certificadoras y de las claves públicas de estas. Estas claves se utilizan para comprobar la autenticidad de los certificados recibidos por clientes y servidores. Por lo tanto, un usuario del entorno debería ponerse en contacto con las entidades certificadoras en cada autenticación.

Alternativamente, el usuario podría mantener replicados los certificados de las entidades certificadoras en todos sus dispositivos. Esta alternativa no sería viable sin un esquema adicional que gestionase los certificados de una forma sencilla para el usuario del entorno ubicuo.

Además, para añadir un elemento al sistema es necesario realizar el proceso de creación de sus certificados. Ese proceso implica un mayor esfuerzo administrativo, y sólo los administradores del entorno son capaces de crear los certificados. En un espacio inteligente es muy común la aparición de nuevos elementos y por tanto los costes de administración se ven incrementados.

Sin embargo, si se utilizasen este tipo de certificados siguiendo un esquema como en el que proponemos en SHAD, de tal forma que se suministrasen de humano a humano, no habría diferencia práctica entre usar una infraestructura

de clave pública y algoritmos de clave privada con claves compartidas. Hemos basado SHAD en un algoritmo de clave privada debido a que este tipo de algoritmos consumen menos tiempo de procesamiento (y por consiguiente, menos energía) que un algoritmo asimétrico.

2.1.3. Entornos descentralizados sobre la plataforma Java 2: Jini

Jini [78] ofrece una serie de componentes que permiten federar los servicios básicos para sistemas distribuidos (transacciones, eventos distribuidos etc.) en grupos de nodos sobre la plataforma Java. El centro de la arquitectura es su servicio de nombres, donde se registran los servicios y se hacen las consultas por parte de los clientes que desean usar algún servicio. Además de ofrecer esta infraestructura, Jini ofrece un modelo de programación para crear aplicaciones distribuidas. Toda comunicación entre clientes y servidores se realiza a través de objetos denominados *objetos representantes* (*Proxy*) que ejecutan en el cliente. Los objetos representantes se implementan en Java y ofrecen unas interfaces bien conocidas.

Los clientes acceden a los objetos representantes mediante invocación a método dentro su propia máquina virtual de Java (JVM). Normalmente, los objetos representantes suministran el servicio por medio de invocación remota a método (RMI), aunque lo pueden hacer de cualquier otra forma. Por esa razón, los servicios finales no necesitan ejecutar sobre la plataforma Java. Esto hace que la arquitectura sea independiente del protocolo que implementen los servicios y hace posible la integración de programas antiguos o que implementen protocolos que no son públicos.

La arquitectura de seguridad para Jini propuesta por **Eronen et al.** [54] ofrece mecanismos de seguridad para autenticar usuarios y certificar sus privilegios. El objetivo de la arquitectura es proporcionar protección contra aplicaciones y los objetos representantes que no son de confianza para el usuario.

La seguridad se fundamenta en un esquema descentralizado sobre una

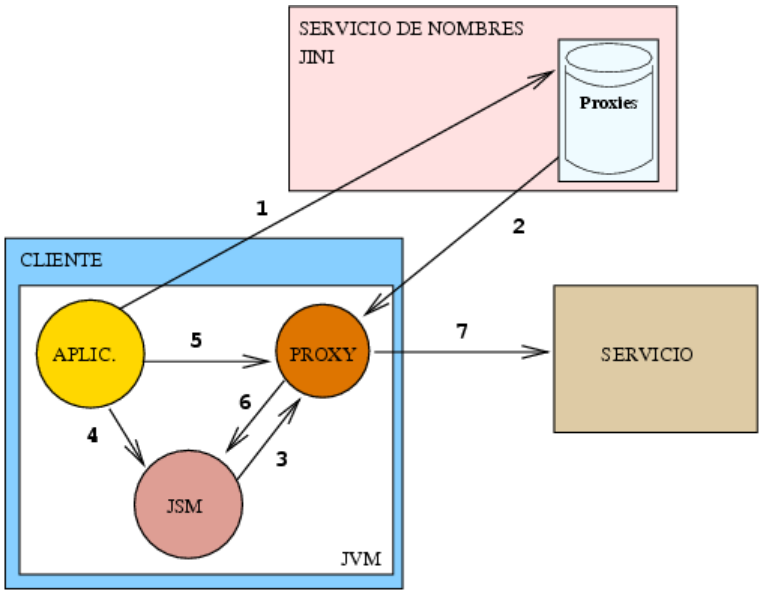

Figura 2.2: *Pasos del protocolo* Jini Decentralized Trust Managent.

infraestructura de clave pública (PKI) en la que el control de acceso sigue un esquema de gestión de confianza (*Trust Management*). Los autores proponen la utilización de certificados **SPKI** [52]. Estos certificados se utilizan para implementar la autorización y no la autenticación: los certificados indican que el propietario tiene derecho a realizar ciertas operaciones, pero no indican la identidad del mismo.

En el sistema, cada servicio posee un par de claves (pública y privada). Para poder acceder a un recurso, el usuario deberá presentar un certificado (almacenado localmente) que permite realizar dicha operación. El certificado SPKI se firma con la clave privada del administrador del sistema. Por tanto, el administrador del sistema ha tenido que proporcionar previamente dicho certificado al cliente. El usuario podrá delegar el certificado a las aplicaciones locales dependiendo del nivel de confianza que tenga en estas.

La figura 2.2 ilustra el protocolo:

1. Cuando una aplicación desea acceder un determinado servicio, se pone en contacto con el servicio de nombres de Jini. No se presenta ningún tipo de

certificado ni se aplica ningún mecanismo de seguridad en esta fase.

2. Una vez que el servicio de nombres devuelve el listado de los servicios, la aplicación escoge alguno de ellos y se descarga el objeto representante. En esta fase tampoco se aplica ningún mecanismo de seguridad.

3. Los objetos representantes contienen una firma de su código y de sus datos obtenida con la clave privada del servicio. Después de descargar el representante, el módulo de seguridad de Jini (JSM) (ejecutando en la misma máquina virtual que el cliente) puede comprobar que el representante pertenece al servicio deseado. Esa comprobación la realiza verificando la firma digital del código y de los datos del objeto.

4. Opcionalmente, puede preguntar al servicio de seguridad de Jini la clave del servicio para comprobar su integridad.

5. El cliente invoca algún método del objeto representante para acceder a un servicio.

6. El representante comprueba que el servicio al que se va a acceder es el correcto (que es el que él mismo representa) y que tiene autorización para realizar la operación en nombre del cliente. En el servidor se genera una clave temporal a la que se le asocian los derechos. Después, el servidor envía un manejador de dicha clave al objeto representante. De esta forma, el servidor puede comprobar que las peticiones están relacionadas con la clave generada, que a su vez está ligada con ciertos privilegios de acceso en el servicio.

7. Con el manejador de la clave temporal, el representante es capaz de enviar las peticiones del cliente al servidor a través de una conexión segura usando invocación remota a método sobre TLS [48]. Cuando el servidor comprueba que el representante posee el manejador de la clave de sesión, el representante envía las peticiones junto a los certificados SPKI que poseía el cliente para ese servicio.

Creemos que este esquema de seguridad no es apropiado para entornos como los expuestos en la introducción. La arquitectura descrita es difícil de administrar, ya que se debe gestionar la distribución de los certificados entre clientes y servicios.

Los usuarios no pueden compartir sus recursos libremente, ya que los certificados que tienen que presentar para que sus operaciones resulten autorizadas deben estar firmados por el administrador del sistema u otra entidad superior en la que han de confiar tanto el cliente como el servidor. De esta forma, los usuarios dependen de un administrador global para poder acceder a nuevos servicios.

Los clientes dependen de los servicios de nombres de Jini. Por tanto, dos usuarios que se encontrasen en una partición aislada del resto de la red no podrían autenticarse mutuamente ni acceder a sus servicios.

Además, este esquema puede depender de servidores de claves o entidades certificadoras. Por ejemplo, el cliente debe confiar en la clave pública del servicio para que el módulo de seguridad de Jini pueda comprobar la firma del código del objeto representante y de sus datos. A priori, el cliente debería desconfiar del servicio, ya que un adversario podría haberle suplantado. Por lo tanto, el cliente no se puede fiar de que la clave pública que le proporciona el servicio es correcta.

Para solucionar este problema, el módulo de seguridad puede seguir una de las siguientes soluciones:

- Obtener las claves públicas de un servidor central. En este caso, el módulo de seguridad de Jini tiene que contactar con una entidad centralizada para obtener la clave pública del servicio o para comprobar que es correcta. Por tanto, los clientes dependen de esta conexión para poder acceder a los servicios y no pueden acceder de forma segura a los servicios cuando esta no está disponible.

- Obtener otro certificado junto al certificado del objeto representante que demuestre que la clave es correcta. Este certificado debería estar firmado

con la clave de una entidad superior (p.e. el administrador). En este caso el cliente debería tener almacenada localmente la clave pública de esta entidad. Un problema es que la entidad superior debería certificar las claves públicas de todos los servicios, lo que hace la administración del sistema todavía más dura. Además, de usar este sistema, los usuarios dependerían de un administrador global para compartir sus recursos.

- Almacenar en el lado del cliente tantas claves públicas como servicios vaya a utilizar. En este caso, el problema del intercambio de claves entre servicios y clientes puede complicarse. Además, la gestión de esas claves por parte del usuario puede ser complicada, ya que un usuario tendría que mantener copias de todas las claves públicas de los servicios a los que puede acceder en cada una de las máquinas que utiliza. Hay que tener en cuenta que en un entorno ubicuo el número de servicios que puede exportar un usuario puede ser muy amplio. La distribución de los certificados necesarios para acceder a cada servicio puede ser demasiado compleja si no se utiliza un esquema adicional como el que proponemos en SHAD.

Como se puede observar, el principal problema para su aplicación en el tipo de sistema que tratamos es la distribución y administración de los certificados.

En este sistema, los clientes dependen de la plataforma Java. Por tanto, las aplicaciones antiguas que no están implementadas en Java necesitan usar *software* adicional (*wrappers*) para poder usar los servicios del sistema.

2.1.4. Sesame

Sesame [86] es un esquema de seguridad ideado para redes de ordenadores basados en un modelo cliente/servidor clásico. El sistema se fundamenta en las mismas ideas que Kerberos y es compatible con este. Esta arquitectura ofrece dos mecanismos de autenticación [11], uno basado en Kerberos y otro basado en una infraestructura de clave pública (PKI), en concreto en certificados X.509 [74]. Además de los mecanismos para la autenticación, Sesame ofrece

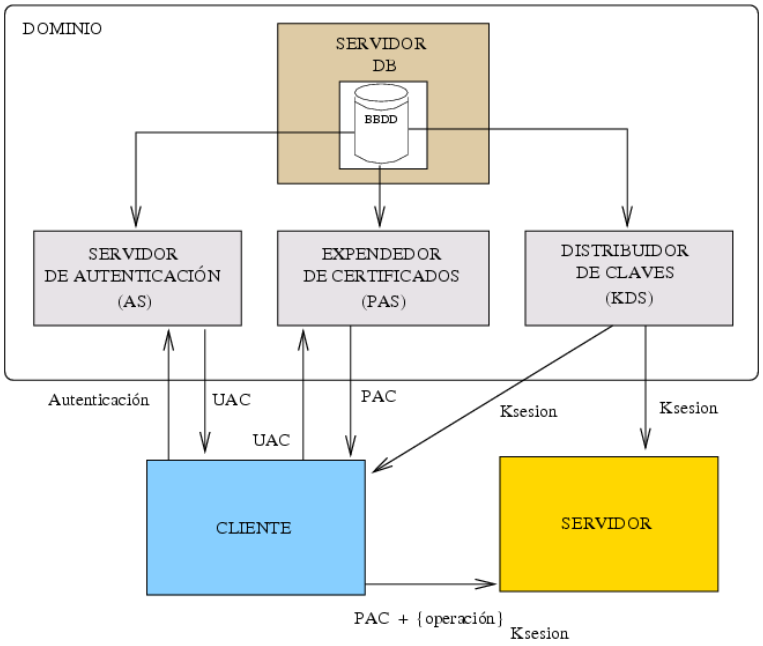

Figura 2.3: *Esquema simplificado de SESAME.*

mecanismos de delegación de privilegios, control de acceso y confidencialidad en las comunicaciones.

Sesame usa un esquema de control de acceso que se basa en roles o RBAC (*Role Based Control Access*). Este enfoque consiste en asignar roles a los usuarios del sistema. El control de acceso se realiza teniendo en cuenta el rol que asume el usuario. El enfoque clásico para el control de acceso son las listas de control de acceso ACL (*Access Control Lists*), en el que los permisos de los recursos se asignan a los propios sujetos. En Sesame, los usuarios poseen certificados de atributos (PAC) que indican sus privilegios. Los certificados incluyen la entidad del poseedor y los roles que le está permitido asumir. Los servidores mantienen listas de control de acceso que indican los permisos de cada rol para el servicio en cuestión. Para que un usuario pueda acceder a un servicio, debe enviar su credencial (junto al resto de la información) para que se aplique la política de control de acceso.

El sistema depende de cuatro servicios centralizados:

- Un servidor de autenticación (AS).
- Un servidor expendedor de certificados (PAS).
- Un servidor distribuidor de claves (KDS)
- Un servidor de base de datos (DB).

El proceso de autenticación de un usuario comienza cuando el usuario se pone en contacto con el AS (ver figura 2.3). Anteriormente, el usuario ha tenido que introducir su contraseña en el terminal. El AS envía al cliente un certificado de autenticación (UAC). Después el cliente envía dicho certificado a un PAS, que sirve los certificados de atributos (PAC) para acceder al los servicios. El usuario usa el certificado de atributos para acceder a los servicios asumiendo un cierto rol. En el propio servicio se aplica la política de control de acceso a los recursos en base al rol que asume el cliente. Cuando no es adecuado utilizar algoritmos de clave asimétrica (p.e. por razones de capacidad de procesamiento), el cliente y el servidor pueden adquirir claves de sesión ($Ksesion$) del servidor distribuidor de claves (KDS).

La base de datos (DB) guarda toda la información relacionada con la seguridad que necesitan los otros servidores.

Sesame, al igual que Kerberos, depende de servicios centralizados que deben estar disponibles para que dos usuarios del sistema se puedan comunicar. Por otra parte, Sesame utiliza técnicas de cifrado asimétrico, que son considerablemente más costosas en tiempo de ejecución (y por tanto en consumo de energía) que las técnicas de cifrado simétrico. Hay que tener en cuenta que en un entorno ubicuo el ahorro de energía puede ser vital para el uso de dispositivos móviles. Sesame tiene otro problema en común con Kerberos: las aplicaciones ya existentes tienen que modificarse para usar la infraestructura de seguridad.

Por último, un usuario del sistema deberá introducir su contraseña en todas las máquinas que desee usar. En un entorno ubicuo, un humano podría usar una gran cantidad de terminales, estaciones de trabajo, dispositivos móviles y

otros tipos de dispositivos al mismo tiempo. Usando un sistema como Kerberos o Sesame, el usuario tendría que introducir una contraseña en todos sus dispositivos. Este nivel de obstrucción no es tolerable en un entorno como el descrito por Weiser [193], aunque tampoco lo es en un contexto más simplista (como los entornos de oficina actuales) en el que los usuarios pretenden usar un gran conjunto de máquinas simultáneamente.

2.1.5. Factotum/Secstore

El sistema operativo Plan 9 [137] usa un esquema de seguridad que separa la autenticación de la lógica de los programas de aplicación. Todos los mecanismos de autenticación están fuera de las aplicaciones (servidores y clientes) y se manejan a través de un servicio llamado **Factotum** [41] que está integrado en el propio sistema operativo. El esquema de seguridad de Plan 9 tiene como objetivos la facilidad de uso y la separación de los mecanismos de seguridad de las aplicaciones.

Factotum se encarga de mantener las claves de los usuarios para distintos servicios y de negociar los protocolos de autenticación, de tal forma que una vez introducidas las claves ya no se requiere ningún tipo de interacción con el usuario. Entre todos los servicios que se pueden usar se encuentran servicios comunes (como SSH, FTP, POP, etc.) y servicios del propio sistema operativo.

Para los servicios comunes se usa su mecanismo específico de autenticación (normalmente nombre de usuario y contraseña). Para los servicios del propio sistema operativo (servidores de ficheros, servidores de CPU, etc.) se utiliza un esquema basado en resguardos (tickets) similar al del Kerberos. En este esquema, los servidores de autenticación generan y distribuyen los resguardos. Estos servidores mantienen listas de usuarios y sus claves secretas. Los servidores de autenticación se organizan en dominios de autenticación.

Factotum puede usar un almacén seguro de claves en el que el usuario puede incluir sus claves para los distintos servicios. Este almacén se llama **Secstore** [169]. Cuando arranca, Factotum puede obtener de Secstore todas

las claves del usuario si éste introduce su contraseña (la de Secstore). A partir de ese momento no se necesita interaccionar con el usuario para autenticarle ante los servicios cuyas contraseñas están guardadas en el almacén. El uso de Factotum junto a Secstore reduce drásticamente la obstrucción al usuario, ya que el usuario sólo debe introducir una sola contraseña por máquina para acceder a todos los servicios desde esa máquina.

Normalmente se mantiene sólo una instancia de Factotum en una máquina Plan 9, aunque puede haber más de una si algún proceso necesita usar un agente propio. Factotum es un servidor de ficheros, y como tal, se puede usar desde otras máquinas. En cada máquina del sistema distribuido hay al menos un agente. La comunicación necesaria para el proceso de autenticación se mantiene entre los agentes de las máquinas involucradas (cliente y servidora) y no entre las aplicaciones (programa cliente y programa servidor). Al ser Factotum el encargado de las tareas de autenticación, las aplicaciones apenas necesitan modificaciones para poder añadir funcionalidad de este tipo. Nótese que estas modificaciones consisten en incluir llamadas al sistema operativo para leer y escribir en el sistema de ficheros, por lo que las aplicaciones no necesitan grandes cambios ni utilizar ningún tipo de nueva tecnología (p.e. *middleware* o *frameworks*). Por esta misma razón, los cambios se pueden aplicar en cualquier aplicación antigua, independientemente del lenguaje de programación en el que fue implementada.

Este esquema sigue un diseño que lo hace extremadamente sencillo de utilizar y de modificar. Pero el proceso de autenticación para los servicios propios de Plan 9 sigue teniendo problemas de dependencia de servidores centralizados, ya que utiliza mecanismos de autenticación similares a los de Kerberos.

Factotum y Secstore siguen un esquema similar a algunos de los esquemas de *Single Sign-On* que se describirán en el punto 2.3, proporcionando autenticación al usuario sin obstrucción en un único puesto de trabajo. El problema de la obstrucción resurge cuando el usuario desea usar varias máquinas a la vez. Para ello, hay que introducir una contraseña en cada máquina que se use. Dado que

el número de máquinas que puede usar un humano en un entorno ubicuo es bastante amplio, la obstrucción es considerable aunque se usen agentes locales a cada máquina.

Además, el acceso al sistema sin obstrucción depende de la conectividad con un único servidor encargado del almacenamiento de claves (en este caso el servidor de Secstore). En SHAD proponemos un esquema en el que los agentes de seguridad que pertenecen al mismo usuario pueden cooperar para distribuir entre sí sus secretos, de tal forma que se evite una obstrucción innecesaria al usuario cuando se usan varios dispositivos simultáneamente.

Aunque Factotum es un esquema de seguridad que proporciona secretos a las aplicaciones, no propone ningún tipo de mecanismo para permitir a los usuarios ceder temporalmente sus dispositivos. En SHAD proponemos un esquema que amplía la subsana esto, creando una arquitectura que no sólo ofrece la distribución de secretos entre los agentes que pertenecen a un mismo usuario, sino que además proporciona mecanismos para compartir dispositivos entre usuarios.

2.2. Seguridad en entornos ubicuos y redes *ad-hoc*

Algunos de los esquemas propuestos para entornos ubicuos [53; 172] intentan tomar decisiones sobre la confianza basándose en información del entorno físico de la entidad. Esta aproximación es compleja, debido a que a priori necesita una infraestructura que le permita el acceso a dicha información de una forma razonable y segura en un entorno ubicuo, y ese es un problema aún por resolver. Esta práctica introduce nuevos problemas de seguridad, puesto que hay que asegurar los sensores que proporcionan la información. El uso de información de contexto en una arquitectura de seguridad es muy interesante para enriquecer las políticas de seguridad, pero no creemos que la relación de confianza se deba basar exclusivamente en esta información.

Nuestro sistema saca partido de una infraestructura de contexto que requiere

autenticación y controla el acceso a su información. SHAD no depende de esta información para su funcionamiento y no se estima la confianza ni se toman decisiones críticas en base a ella (se utiliza principalmente para evitar confirmaciones que puedan provocar obstrucción al usuario).

Resurrecting Duckling [177] introduce el concepto de "asociación segura transitoria". Cuando dos dispositivos de un entorno ubicuo desean interaccionar de una forma segura durante un periodo de tiempo, se debe crear una relación de este tipo. La interacción segura hace que ningún otro dispositivo (del mismo tipo o de cualquier otro tipo que los participantes) pueda suplantar a otro, interferir en la comunicación o espiar los datos retransmitidos. En este trabajo se propone la siguiente política de seguridad creada especialmente para comunicaciones inalámbricas:

- Un dispositivo actúa como maestro y otro como esclavo.
- Un dispositivo esclavo no posee identidad hasta que un maestro se la da. La identidad es una clave secreta compartida entre los dos dispositivos. Esta identidad no se puede asignar nunca a través de un canal inalámbrico.
- Una vez que un dispositivo esclavo tiene identidad, obedecerá al maestro en todo momento. Sólo el maestro puede quitar la identidad al dispositivo esclavo para que pueda ligarse a otro dispositivo maestro, por tanto ninguna otra entidad puede tomar el control del esclavo sin que el maestro al que le pertenece lo libere antes.

En este trabajo se introduce la idea de usar una relación de este tipo entre el humano y un dispositivo. El humano puede ser el maestro para un dispositivo de control que a la vez es maestro para otros dispositivos. Por ejemplo, el humano es el maestro de un control remoto de un dispositivo de vídeo, al que le introduce una contraseña al iniciar la sesión. En ese momento, el control remoto posee la identidad que le ha dado el humano. Para controlar el dispositivo de vídeo, el control remoto le da una identidad, de tal forma que sólo le obedezca a él. De

esta forma se crea una cadena de mando, que transmite las órdenes del humano hasta el dispositivo final. El problema de este modelo reside en que si un eslabón de la cadena falla, la orden no le llegará al dispositivo.

El protocolo tiene una extensión para los casos en los que los dispositivos actúan como pares iguales, por ejemplo en una red *Peer-to-Peer*. En esta extensión el protocolo añade delegación: se permite que el esclavo acepte órdenes de otros dispositivos que no sean el maestro. Los dispositivos esclavos pueden obedecer a otros maestros por un periodo de tiempo sin tener que adquirir otra personalidad. En otras palabras, los dispositivos pueden cederse temporalmente a otros maestros con el consentimiento del maestro padre (el que le dio la identidad). Las políticas que se pueden aplicar a un esclavo son predicados que especifican las credenciales que un dispositivo debe presentar al esclavo para realizar ciertas acciones sobre él. Si una entidad puede demostrar que conoce la clave del esclavo (su identidad o la clave que le dio el maestro que le revivió), entonces podrá instalarle nuevas políticas. En el dispositivo esclavo, un motor de gestión de confianza (*Trust Management*) evalúa las credenciales y decide si un cliente puede realizar ciertas acciones. Por otra parte, el dispositivo esclavo puede realizar operaciones poco peligrosas que son declaradas como inocuas sin la necesidad de presentar ningún tipo de certificado.

Si un atacante se hace con un dispositivo con las credenciales necesarias para usar otros dispositivos, y es capaz de extraerlas del mismo (si el dispositivo no es resistente a forzamientos), entonces podrá acceder a ellos sin permiso. Siempre que se proporcione delegación guardando secretos en los dispositivos, nos encontraremos con ese problema.

El propio autor del sistema acepta lo que él mismo denomina el "principio del gran palo" (*The Big Stick Principle*) [179, Capítulo 4]: cualquiera que tenga acceso físico al dispositivo es capaz de tomar el control del mismo y de su contenido. En este punto, la revocación de claves es el principal reto: una vez que una clave ha sido comprometida, se debe revocar en todos los lugares en los que se usa. Si se garantiza la conectividad entre los dispositivos, entonces

el problema de revocación se puede resolver fácilmente. Sin embargo, en un entorno *Peer-to-Peer* en el que no todos los nodos tienen conectividad entre sí, la revocación de claves es complicada.

La idea propuesta en este trabajo es muy interesante si se combina con una arquitectura general que permita que los humanos compartan los dispositivos que les pertenecen. Al igual que Resurrecting Duckling, en SHAD proponemos relaciones de pertenencia centradas en el humano. La diferencia principal entre ambos enfoques es que SHAD propone relaciones no transitorias entre dispositivos y humanos, de tal forma que un dispositivo siempre pertenece al mismo usuario, aunque este pueda ceder sus pertenencias temporalmente a la gente en la que confía.

En SHAD todos los dispositivos tienen un dueño (que normalmente siempre será el mismo). Esos dispositivos pueden cederse temporalmente a otros usuarios, pero conservando siempre su identidad y su propietario. El usuario posee un terminal móvil llamado UbiTerm que controla la seguridad de todos sus dispositivos. Los UbiTerms pertenecientes a los dos usuarios involucrados en ese proceso son los encargados de gestionar de una forma segura esa cesión temporal. A diferencia de Resurrecting Duckling, los UbiTerms controlan cualquier tipo de autenticación y de cesión de los recursos pertenecientes a sus dueños. De esa forma el UbiTerm es el dispositivo maestro para todos los otros dispositivos que pertenecen al usuario y centraliza su autenticación. De hecho, el UbiTerm es el representante del humano en el sistema y centraliza su seguridad, concepto que no existe en Resurrecting Duckling.

La restricción que supone para Resurrecting Duckling asignar la identidad a un dispositivo a través de una comunicación no inalámbrica puede provocar obstrucción al usuario y, en algunos casos, puede hacer que el usuario no pueda utilizar los dispositivos libres existentes (p.e. en localizaciones donde no estén disponibles conexiones a través de cable). El hecho de que no pueda usarse un dispositivo compartido hasta que el usuario que le ha dado identidad se la retire también puede dar lugar a problemas de desaprovechamiento de recursos por

parte de usuarios malintencionados o distraídos que no liberan las máquinas que han usado previamente y en cualquier caso constituye una obstrucción al usuario. En SHAD, el dueño de los recursos autoriza su uso, de tal forma que otro usuario no puede bloquearlos (intencionadamente o por descuido). La responsabilidad de dar acceso a los recursos recae sobre su dueño, que puede restringir el uso a los usuarios que hacen un mal uso de estos.

English et al. [53] expresan la necesidad de nuevos esquemas de seguridad para los entornos ubicuos. Estos autores proponen la evaluación de la confianza entre computadoras autónomas que sólo tienen acceso a información incompleta sobre los demás componentes del entorno ubicuo. En este trabajo se evalúa el nivel de confianza en otras entidades de una forma autónoma mediante una serie de valores que los autores consideran indicadores de confianza. Estos valores representan tres tipos de información sobre una entidad:

- Opinión personal basada en observaciones sobre el comportamiento de las entidades. El sistema guarda esta información en un historial.
- Recomendaciones de terceras partes en las que se confía a priori.
- La reputación de la entidad, que se puede consultar en un servicio en el que también se confía a priori.

Mediante esta información, la confianza que tiene una entidad en otra es dinámica, ya que los tres tipos de información pueden variar con el tiempo.

La aproximación es compleja, ya que determinar qué factores hacen que un cierto comportamiento se considere malicioso o erróneo no parece una tarea trivial. En el trabajo no se presentan los factores que pueden determinar el comportamiento de una entidad. Tampoco se explica cómo se podrían asegurar esos factores. Prescindir de la interacción con el usuario puede aportar facilidad de uso y una reducción de la obstrucción al usuario, pero a la vez se pierde la información subjetiva que este pueda aportar. Además, la información obtenida sobre otras entidades (recomendaciones, reputación, etc.) puede no

estar disponible en entornos fraccionables si normalmente reside en servidores centralizados.

En SHAD proponemos un enfoque más sencillo, que se basa exclusivamente en la opinión de un humano sobre otro humano. Los dispositivos, al fin y al cabo, pertenecen a un usuario. El nivel de confianza en dispositivos ajenos es algo subjetivo y depende de la relación del usuario con el dueño de estos.

Attribute Vector Model [172] sigue la misma línea que el trabajo anterior. Su aproximación al problema se centra en la información del contexto físico de la entidad (p.e. la localización). Mediante esa información se intentan establecer relaciones de confianza en escenarios donde la información sobre la identidad de los demás puede no estar disponible (información que puede extraerse de sistemas clásicos basados en servidores centralizados). Los autores proponen un esquema que se basa en vectores que almacenan información de la identidad del usuario junto a información sobre el contexto de la entidad. Se pretende obtener un valor que represente la confianza en otra entidad mediante la aplicación de una función sobre el vector que le corresponde, de tal forma que la relación de confianza entre dos entidades depende de los vectores que les representan en cada una de las partes. Una vez que se ha aplicado la función al vector, el nivel de confianza se define en base a un umbral determinado para esa entidad.

Los autores argumentan que la relación de confianza es totalmente descentralizada, ya que sólo depende de los vectores en las dos partes. Pero esto no es totalmente cierto, debido a que los valores de ciertas posiciones de los vectores dependen de la información sobre la identidad de la entidad, y no explican cómo pueden obtenerse, aunque hacen referencia a los sistemas clásicos (se entiende que se refieren a esquemas como Kerberos o PKI). En ese caso, los valores de los vectores sí dependen de entidades centralizadas. Suponiendo que la función capture la noción de confianza de un humano de una forma aceptable, tampoco explican cómo se extrae la información relacionada con el contexto si esta información no está centralizada en servidores.

Shankar et al. [173] proponen también el uso de la información relacionada

con la localización y el contexto físico de las entidades. Los autores pretenden agrupar los dispositivos según su información de contexto (p.e. los dispositivos que están dentro de una misma sala). Los dispositivos que pertenecen a un mismo grupo pueden mantener conexiones seguras entre sí. En este trabajo, los autores presentan un marco (*framework*) que permite usar la información de contexto desde las aplicaciones. Además, ofrecen un servicio de seguridad que saca provecho de esta información.

El marco ofrece operaciones que permiten crear vistas del contexto para aplicación. De esta forma, cada aplicación podrá incluir en su vista los valores de contexto que más le interesen (p.e. la localización, la temperatura o la carga de la batería del dispositivo). Las entidades se incluyen en grupos que comparten vistas, y un servicio centralizado se encarga de establecer conexiones seguras dentro del grupo. Por esta razón se debe distribuir la clave del grupo entre las entidades de una forma segura. Los autores proponen el uso de un canal seguro entre los miembros del grupo y el servicio de seguridad que lo gestiona. Este canal se basaría en una infraestructura de clave pública (PKI) o en el uso de secretos compartidos preestablecidos en una fase de configuración.

Este trabajo se centra más en la arquitectura de contexto que en la de seguridad, ya que lo único que propone es la creación de grupos en base al contexto para después aplicar un esquema clásico de seguridad dentro de ellos. No obstante, el esquema no funcionaría en entornos fraccionables, debido a que se depende del servicio de seguridad que gestiona al grupo de entidades. Por último, y como ya se ha comentado anteriormente, el uso obligado de un marco conlleva problemas de interoperabilidad.

Secure Smart Homes [6] propone una arquitectura de seguridad que se basa en Tiny Sesame, una versión reducida de Sesame [86] creada para la plataforma Java. Los autores mantienen que las arquitecturas de seguridad convencionales (mecanismos de Java, Sesame y Kerberos) requieren demasiada capacidad de procesamiento y por tanto no son adecuadas para un entorno ubicuo. Por esta razón los autores proponen el uso de Tiny Sesame, que es una

versión más ligera respecto a necesidades de procesamiento (y por tanto, más apropiada para dispositivos móviles y dispositivos destinados a la domótica).

La arquitectura propuesta se compone principalmente de un servidor de autenticación de Sesame y de ciertos componentes que ejecutan en el cliente y en el servidor. En el cliente, un componente llamado APA (*Authentication and Privilege API*) se encarga de autenticar al cliente ante el servidor de autenticación de Sesame (AS) y el servidor de atributos de privilegios (PAS).

La autenticación de un usuario se lleva a cabo mediante la introducción de contraseñas. Una vez que el AP ha autenticado al usuario, el servidor de atributos le proporciona sus roles y atributos de privilegio. Cuando el usuario necesita usar un servicio del entorno ubicuo, un componente llamado SACM (*Secure Association Context Manager*) enviará a otro componente similar del servidor los roles obtenidos del PAS. Una vez que el componente SACM del servidor comprueba que los roles y los privilegios del usuario son correctos, el cliente puede realizar peticiones al servidor. Las peticiones las realiza mediante invocación remota a método.

Esta arquitectura se basa en Sesame y, por lo tanto, adolece de los mismos problemas. El principal problema es que los clientes dependen de un único servidor para poder autenticarse. Eso implica que los clientes sólo pueden autenticarse cuando el servidor de seguridad esté disponible. Sin contactar con el servidor de autenticación, el usuario no puede utilizar los servicios del entorno ubicuo. El servidor de seguridad es un punto único de fallo del sistema, ya que si es objetivo de un ataque de denegación de servicio y este tiene éxito, todo el entorno ubicuo quedaría bloqueado. Por otra parte, el usuario debe autenticarse introduciendo contraseñas, lo que provoca obstrucción al usuario del entorno ubicuo, como ya se ha explicado anteriormente.

Sizzle [68] es un intento de adaptar el protocolo SSL [40] a dispositivos empotrados de un entorno ubicuo. Sizzle es en realidad un servidor WWW implementado en el dispositivo empotrado junto a los distintos niveles de red de TCP/IP, incluidos SSL y HTTP. En este trabajo han conseguido

implementar el protocolo SSL de tal forma que pueda ejecutar en dispositivos con grandes limitaciones de memoria y procesamiento (los dispositivos utilizan una CPU de 8 bits). El protocolo utiliza criptografía de curva elíptica [91; 108; 164] en lugar de los algoritmos de clave pública comúnmente usados en SSL, como RSA [154]. De esta manera, los dispositivos con más carencias de procesamiento son capaces de exportar sus servicios mediante un servidor HTTP seguro.

El número de dispositivos contra los que un usuario se tiene que autentificar crece sustancialmente cuando todos los sensores y otros dispositivos empotrados exportan sus servicios usando SSL o cualquier otro protocolo de seguridad basado en certificados o en claves. Por lo tanto, Sizzle hace aun más necesario un esquema genérico de acceso a los dispositivos del entorno ubicuo que no obstruya el acceso a los clientes, ya sean humanos o programas.

HESTIA [72] es una arquitectura basada en *middleware* (Corba [38], en concreto), propuesta para entornos ubicuos y edificios inteligentes. Esta arquitectura divide el entorno en dos dominios: el dominio de servicio y el dominio de aplicación. Estos dominios se separan en diferentes redes mediante el uso de *firewalls* y NAT (*Network Address Translation*).

El dominio de servicio ofrece a sus clientes mecanismos de tolerancia a fallos, calidad de servicio y privacidad mediante el uso de nodos especiales denominados *middleboxes*. Estos nodos ofrecen una interfaz programable orientada a objetos que proporciona los servicios de control y seguridad.

Además, en este dominio reside una base de conocimientos y un motor de inferencia (ambos centralizados) donde se almacenan información de contexto, jerarquías de roles y políticas de seguridad. El motor de inferencia es capaz de aplicar las políticas de seguridad y de detectar intrusiones. Los servicios instalan *proxies* en los *middleboxes* de este dominio, que a su vez ofrecen a los usuarios la interfaz para usar dichos servicios. La interfaz depende de la política de control de acceso y de los roles asignados.

Los *proxies* son objetos no persistentes y mantienen un estado limitado. El

dominio de aplicación ofrece a los usuarios una interfaz para usar los servicios disponibles del dominio descrito anteriormente basándose en el rol que tienen asignado.

Los clientes localizan los *middleboxes*, que están separados en una red distinta, mediante un protocolo de descubrimiento. Una vez localizados, los clientes obtienen las interfaces necesarias para poder usarlos mediante invocación remota a método (RMI).

Para ofrecer protección en la red de *middleboxes* se utilizan diversas capas que se integran en el *middleware*:

- **Capa de reparto de carga:** evita la sobrecarga de los *middleboxes*. Si un nodo se sobrecarga, la capa de reparto traslada algunos de sus servicios a otros nodos que están menos ocupados.
- **Capa de tolerancia a fallos:** el *middleware* proporciona tolerancia a fallos manteniendo replicados los servicios en distintos *middleboxes*. Si fallan varios nodos que replican el acceso a un servicio, el *middleware* se encarga de arrancar otros nodos dedicados a ello.
- **Capa de anonimato:** esta capa hace que la identidad y localización de los clientes y de los *middleboxes* no estén disponibles.
- **Capa de calidad de servicio:** permite a los usuarios especificar la calidad de servicio (QoS) que desean.
- **Capa de seguridad:** esta capa permite a los *middleboxes* proporcionar confidencialidad, integridad y control de acceso a los servicios. Los servicios pueden añadir políticas de seguridad a esta capa.

HESTIA es una arquitectura que está específicamente orientada a proporcionar seguridad en entornos en los que se centralizan los servicios y no se contempla la interacción entre usuarios que comparten sus recursos. Esta arquitectura deriva de otras arquitecturas de seguridad propuestas para los espacios inteligentes del proyecto GAIA [156], que se describen a continuación.

Como en el caso de todas las arquitecturas propuestas para GAIA, la arquitectura ofrece mecanismos de seguridad para acceder a servicios centralizados siguiendo un esquema cliente/servidor clásico. Por ello HESTIA no es apropiada para entornos *Peer-to-Peer*. Nuestro objetivo en SHAD es ofrecer una arquitectura genérica para entornos ubicuos y sistemas distribuidos menos restrictivos que sigan un esquema *Peer-to-Peer*.

2.2.1. Arquitecturas propuestas para GAIA *Smart Spaces*

GAIA [156; 58] es un intento de crear espacios inteligentes (*Smart Spaces*) para conseguir un entorno ubicuo. GAIA es una arquitectura basada en *middleware*. Concretamente, GAIA utiliza Corba [38]. A continuación se detallan los esquemas propuestos para los espacios inteligentes de GAIA.

GAIA Security Architecture [190; 160] utiliza certificados PKI como credenciales para las entidades del sistema. Estas credenciales se reparten a clientes y a servidores, y pueden ser de tres tipos:

- **Las credenciales genéricas** se utilizan para que una aplicación pueda realizar operaciones en nombre de un usuario. En esta arquitectura se sigue un esquema de control de acceso por roles (RBAC) donde un usuario puede tener varios roles, y mediante este tipo de credencial puede ceder un rol a una aplicación cliente.

- **Las credenciales restringidas** sirven para ceder privilegios a aplicaciones que no son de confianza. Estas credenciales son iguales que las anteriores, pero además añaden una lista con los privilegios que especifican la aplicación a la que se ha cedido la credencial y los recursos para los que la puede utilizar.

- **Las credenciales no derivables** son aquellas que puede usar el cliente para autenticarse pero que no conceden ningún privilegio a la entidad que las recibe. Estas credenciales llevan el identificador del servicio para el que

se han creado. Por lo tanto no podrán utilizarse para autenticar al usuario ante otro servicio distinto.

El servidor de autenticación (AS) del espacio inteligente es el encargado de crear las credenciales que el usuario debe obtener para que los clientes que ejecutan en su nombre puedan solicitar servicio a los servidores del espacio inteligente. Por lo tanto, el usuario deberá autenticarse ante el AS. Un usuario puede autenticarse mediante contraseñas o usando distintos dispositivos móviles.

Dentro de los dispositivos de autenticación propuestos para GAIA, se encuentran relojes de pulsera [7], en concreto relojes Matsucom's OnHand^{tm} PC [129]. Los relojes disponen de un puerto de comunicación por infrarrojos (IRdA), capacidad de proceso y capacidad de almacenamiento.

El mecanismo de autenticación de estos dispositivos se basa en certificados de clave pública más ligeros que los certificados X.509 [74]. Los certificados sólo contienen el identificador del usuario, su clave pública, la autoridad certificadora que lo respalda, algunos datos adicionales y la firma de la entidad certificadora. El certificado del propio usuario se almacena en la memoria del reloj. Debido a la carga que supone el uso de criptografía de clave pública, los relojes negocian una clave para un algoritmo de criptografía de clave privada siguiendo el protocolo Diffie-Hellman [49]. El protocolo de autenticación es el siguiente:

1. Cuando el usuario entra en el espacio, un dispositivo de autenticación transmite al reloj la clave pública del espacio.

2. El reloj envía su certificado al dispositivo de autenticación. El dispositivo de autenticación verifica la firma del certificado consultando a la autoridad certificadora.

3. Se negocia una clave de sesión usando el algoritmo de Diffie-Hellman.

4. El dispositivo de autenticación genera un mensaje aleatorio que servirá como reto, y se lo envía al reloj.

5. El reloj cifra el mensaje recibido con la clave de sesión y se lo devuelve al dispositivo del espacio.

6. Si el mensaje recibido por el dispositivo del espacio es descifrado y se corresponde con el mensaje aleatorio, el usuario está autenticado. En otro caso, no lo está.

El protocolo se puede ampliar para que tenga autenticación mutua si se repiten los tres últimos pasos en sentido de comunicación inverso, para que el reloj pueda autenticar al dispositivo del espacio activo.

En este esquema surge un problema: el reloj del usuario tiene que confiar en que el dispositivo del espacio activo es quien dice ser, ya que no puede consultar a la autoridad certificadora para comprobar que la clave pública del espacio es correcta.

El control de acceso a los recursos del espacio inteligente se gestiona mediante tres tipos de políticas. La política basada en roles se aplica a los ficheros del sistema y a los servicios del entorno. Para los ficheros de los usuarios, se utiliza un control de acceso discrecional, de tal forma que los propietarios eligen qué usuarios pueden acceder a sus ficheros. Para los recursos más delicados, como por ejemplo documentos de alto secreto, se aplica un control de acceso obligatorio (*Mandatory Access Control*), en el que los dueños de los recursos no gestionan su acceso.

Estos aspectos del control de acceso se controlan mediante el servicio de control de acceso (ACS). Cuando un cliente necesita usar un servicio, un componente situado en la misma máquina intercepta la petición. Ese componente se denomina interceptor, y es el encargado de agregar las credenciales del usuario en las peticiones. En el lado del servidor, otro componente interceptor captura la petición y contacta con el servicio de control de acceso para presentar las credenciales incluidas en esta. El servicio de control de acceso comprueba el tipo de control de acceso del recurso y garantiza el acceso dependiendo de la credencial entregada y la política asociada al recurso. Para

llevar a cabo el control de acceso, el servidor de autenticación y el servidor de control de acceso deben compartir claves.

Esta arquitectura es flexible y permite un preciso control de acceso a los recursos, pero a la vez es demasiado compleja. El uso de una arquitectura basada en *middleware* hace que los desarrolladores dependan de la plataforma empleada y que aplicaciones antiguas deban ser reescritas o envueltas por otros programas para poder interactuar con en el entorno.

GAIA sigue un esquema cliente/servidor en el que los clientes acceden a servicios atravesando una capa intermedia que es la que añade la autenticación y el control de acceso. Esa capa intermedia, implementada en el bus del *middleware*, utiliza varios servicios centralizados. Por lo tanto, una aplicación cliente depende de varios servicios para poder autenticarse y presentar sus credenciales. A su vez, esos servicios centralizados dependen entre sí: los servicios de control de acceso y de autenticación dependen de un distribuidor de claves común. El fallo o desconexión de cualquiera de esos servicios implica la denegación de servicio en todo el espacio.

Esta arquitectura no hace uso de información de contexto obtenida del entorno ubicuo. Eso supone una gran desventaja, ya que esta información puede ser muy valiosa para determinar si ciertas acciones son peligrosas.

Otro problema es que este tipo de arquitecturas hace la administración del sistema demasiado trabajosa. Los administradores del sistema deben incluir los nuevos recursos en el servicio de control de acceso y gestionar las políticas. Un usuario no es capaz de introducir un dispositivo nuevo en el sistema sin contactar con los administradores. Además, el servicio de autenticación también necesita administrarse por separado, ya que los usuarios necesitan tener cuentas en dicho servicio.

GAIA propone un tipo de entorno ubicuo aislado en el que los servidores centralizados controlan a los usuarios, dispositivos y servicios. En la práctica, pueden existir otros tipos de entornos ubicuos menos restrictivos y más abiertos en los que los usuarios interaccionan entre sí compartiendo sus recursos con la

gente en la que confían. En un entorno más permisivo, los usuarios necesitarían tener más control sobre sus recursos y un grado de libertad mayor a la hora de poder acceder a los servicios.

SHAD propone una arquitectura de seguridad genérica para entornos ubicuos menos restrictivos que el propuesto en GAIA, en el que el acceso a los recursos no se realiza siguiendo un esquema tan centralizado. En SHAD, los humanos no se autentican ante el sistema, sino que se autentican ante otros humanos. Los servicios son recursos que pertenecen a un humano o a un usuario virtual.

Cerberus [9] es una arquitectura de seguridad propuesta para los espacios inteligentes de GAIA que se basa en otros trabajos descritos anteriormente [6; 190; 160]. Cerberus es un servicio de GAIA que incluye funciones para autenticar usuarios y que además proporciona información sobre el contexto. La arquitectura se compone de los siguientes servicios: servicio de seguridad, infraestructura de contexto, base de conocimiento y, por último, un motor de inferencia.

El componente de seguridad se encarga de autenticar usuarios. Los métodos para autenticar usuarios varían dependiendo del nivel de confianza del método y la situación en la que se encuentra el usuario. Los métodos con un nivel de confianza menor sólo se podrán usar en situaciones en las que el contexto indica que el nivel de seguridad es bajo. El contexto indica la fortaleza que debe tener el método de autenticación en un momento dado. Normalmente, cuanto más incómodo es el método de autenticación, más alto es el nivel de confianza de este.

Cuando un cliente se debe autenticar, utiliza módulos PAM (*Pluggable Authentication Modules*). De esa forma se separa el proceso de autenticación de la aplicación. Cada método de autenticación debe tener su propio módulo PAM. Los módulos de autenticación se dividen en dos submódulos: uno que implementa los mecanismos de autenticación de Sesame [86], Kerberos [182], autenticación por retos y mediante contraseñas, y otro que es específico para cada dispositivo de autenticación.

La infraestructura de contexto de GAIA se encarga de proporcionar información de contexto al motor de inferencia. A su vez, la base de conocimiento contiene las políticas de seguridad expresadas en lógica de primer orden. Esas políticas pueden ser de dos tipos: unas se utilizan para determinar el nivel de confianza de un método de autenticación y otras se encargan de definir cuándo una entidad puede acceder a un determinado recurso.

El motor de inferencia es el encargado de evaluar el contexto y marcar el nivel de seguridad. Dependiendo del nivel de seguridad, el usuario tiene que utilizar un método de autenticación con un grado de confianza mayor o menor. También se encarga de indicar a los servicios si una entidad puede acceder a un recurso, teniendo en cuenta las políticas de control de acceso al recurso, las credenciales de la entidad y la información de contexto. Por esta razón, el motor de inferencia debe consultar a los proveedores de información de contexto de la infraestructura de GAIA.

A diferencia de las otras arquitecturas propuestas para GAIA, Cerberus saca partido de la información de contexto del entorno. Pero los problemas de centralización y dependencia de servicios afectan a esta arquitectura más que a sus predecesoras, ya que incluye dos servicios centralizados más: el motor de inferencia y la base de conocimiento. Esta arquitectura necesita un gran esfuerzo administrativo, ya que las reglas de la base de conocimiento añaden complejidad al esquema de control de acceso. Por último, Cerberus sufre los mismos problemas de interoperabilidad que las arquitecturas descritas anteriormente.

2.3. Sistemas de entrada única al sistema (*Single Sign-On*)

La entrada única al sistema (*Single Sign-On* o SSO) en el ámbito de un entorno de trabajo de oficina (o en el de un entorno ubicuo) consiste en facilitar al usuario el acceso al sistema, de tal forma que no tenga que autenticarse

constantemente para acceder a diferentes servicios. En un sistema de *Single Sign-On*, el usuario sólo se debe autenticar una vez ante el sistema, que gestionará las sucesivas autenticaciones sin interrumpirle.

Actualmente existen una serie de sistemas que proporcionan *Single Sign-On* en entornos clásicos donde el usuario trabaja con una única máquina. Sin embargo, hasta donde sabemos, estos sistemas no contemplan la posibilidad de que el usuario utilice múltiples máquinas simultáneamente.

A continuación se comentarán varios de esos sistemas y los inconvenientes que surgen al aplicar sus esquemas de uso un entorno como el descrito en la introducción.

Password Safe [166] es un organizador de contraseñas diseñado para el sistema operativo MS Windows [115]. Este agente se encarga de guardar un repositorio de contraseñas del usuario, para que este pueda gestionarlas de una forma cómoda. Cuando un usuario necesita proporcionar una contraseña a una aplicación, la debe seleccionar con el ratón de una lista y pegarla en la aplicación en lugar de introducirla por teclado. De esta forma se evita que el usuario tenga que recordar las contraseñas de todos los servicios que usa. El repositorio de claves se guarda en el dispositivo de almacenamiento del terminal en el que se usa el agente, cifrado mediante un algoritmo fuerte.

Este esquema sigue provocando obstrucción al usuario incluso cuando usa una sola máquina, ya que se tiene que seguir autenticando explícitamente, aunque de una forma más cómoda que introduciendo claves por teclado. En todo caso, este sistema no proporciona SSO al usuario. Existen diferentes versiones de este programa para otras otras plataformas (Mac OS, UNIX, Windows Mobile etc) con nombres como MyPasswordSafe [116] o Pwsafe [145].

SSH-agent [24], el agente de seguridad de SSH [199], proporciona una entrada única al sistema a los programas clientes de SSH. El usuario puede generar pares de claves de cifrado asimétrico para definir una identidad del usuario. Una vez generadas, el usuario debe distribuir las claves públicas entre las máquinas a las que quiere acceder sin obstrucción. Cuando el usuario inicia

la sesión, el agente de SSH adquiere las claves privadas del disco y las almacena en su memoria (opcionalmente puede requerir una contraseña para cada una de estas claves privadas). Para conectarse a una de las máquinas en las que se ha almacenado la clave pública de alguna de las identidades que tienen guardadas en la máquina local, no es preciso introducir ninguna contraseña. En ese caso, el cliente de SSH se pone en contacto con el agente a través de un *Unix-Domain Socket* [184, Capítulo 16] del sistema operativo para que firme bloques de datos con la clave privada de la entidad. De esa manera, el agente proporciona autenticación pero nunca entrega las claves que tiene almacenadas.

El agente de SSH sólo proporciona SSO para el servicio de acceso remoto a consola. Además, si un usuario desea utilizar localmente múltiples máquinas a la vez, entonces deberá introducir al menos una contraseña por cada máquina para iniciar la sesión (la contraseña de usuario del sistema, la que puede solicitar el agente SSH para añadirle una identidad o ambas). Si no se solicita autenticación ni para el inicio de sesión ni para añadir las identidades en el agente de SSH, entonces cualquier atacante podrá arrancar la máquina y acceder a todas las cuentas sin que el dueño ni siquiera fuera advertido de ello.

Por el contrario, en nuestro caso los agentes de SHAD cooperan para suministrar los secretos necesarios para que las aplicaciones puedan acceder a los diferentes servicios. El usuario puede definir las situaciones y eventos que se le tienen que notificar en el UbiTerm (p.e. la petición de un cierto secreto o el arranque de una máquina en su nombre). Además, puede especificar conjuntos de acciones que además de notificarse, requieren una confirmación explícita (p.e. la petición de un secreto sensible desde uno de sus agentes).

El agente de SSH también tiene problemas con la distribución de las claves. El usuario debe gestionar manualmente la distribución de las claves mediante su copia en sus cuentas de usuario. Si un usuario desea acceder sin obstrucción desde varias máquinas, entonces no sólo debe distribuir las claves públicas, sino que también debe distribuir las claves privadas correspondientes a sus identidades. SHAD, en cambio, permite al usuario almacenar las claves y contraseñas de

todos los servicios del entorno en un único repositorio, de tal forma que las únicas claves que debe distribuir manualmente el usuario son las del propio protocolo de SHAD.

Mozilla [112] o, **Netscape** [124] y otros navegadores ofrecen mecanismos de *Single Sign-On* para almacenar las contraseñas para los protocolos HTTP y FTP. Igual que en el caso anterior, estos programas almacenan las contraseñas del usuario para acceder a un servicio en concreto (en este caso a los servicios WWW y FTP). Además, el SSO es local, ya que los secretos no son accesibles desde otros navegadores que puede estar utilizando el usuario en otras máquinas.

Otros programas actúan como agentes que proporcionan autenticación a las aplicaciones que utiliza el usuario. A continuación se presentarán varios sistemas de este tipo.

SecureLogin Single Sign-On [144] y **Focal Point** [128] son sistemas de SSO comerciales disponibles para el sistema operativo MS Windows. Su enfoque es similar.

SecureLogin almacena las credenciales de un usuario en un servicio de directorio LDAP [198]. Cuando un usuario arranca la estación de trabajo, se autentifica por primera vez y el sistema recupera sus secretos del servidor de LDAP. La autenticación puede seguir dos métodos distintos:

- **Autenticación en segundo plano:** en ambos lados de la aplicación distribuida (cliente y servidor) hay componentes de SecureLogin que negocian la autenticación. Cuando el usuario quiere acceder al servicio, el cliente contacta con el servidor. En ese momento, el componente situado en el servidor envía un reto al componente situado en el cliente. Al reto tiene que responderse mediante un resguardo, denominado *passticket*. Si el proceso de autenticación entre los componentes prospera, se notifica a la aplicación que el usuario se ha autenticado.

- **Guardando y pasando la contraseña:** el servicio de directorio tiene almacenadas las contraseñas de un usuario. Cuando una aplicación necesita que el usuario se autentifique, SecureLogin recupera la contraseña y se

la pasa sin que el usuario tenga que intervenir. El agente intercepta los diálogos que piden contraseñas sin que el usuario se percate, e introduce la contraseña en el campo de la entrada de texto correspondiente. Este método es más común que el anterior.

SecureLogin puede guardar la contraseña localmente para no volver a preguntar al servidor de secretos si el usuario necesita autenticarse de nuevo ante esa aplicación. Este proceso es transparente para el usuario, al que en ningún momento se le notifica que el agente se ha autenticado ante una aplicación.

En realidad, SecureLogin se compone de tres tipos de agentes:

- Un agente encargado de proveer contraseñas al navegador WWW MS Internet Explorer.
- Un agente encargado de capturar los diálogos de autenticación de aplicaciones de MS Windows.
- Un agente para terminales de MS Windows. Este es el encargado de aportar las claves para las sesiones de terminal remoto a otros sistemas diferentes (UNIX, etc.).

Factotum, el agente de seguridad del sistema operativo Plan 9, descrito anteriormente en la sección 2.1, proporciona autenticación a las aplicaciones siguiendo dos métodos distintos. En algunos casos, Factotum usa un esquema basado en RPCs (*Remote Procedure Call*) [186, Capítulo 2] para suministrar los secretos a los programas que gestionan la autenticación. En otros casos no se limita a servir un secreto a la aplicación, sino que se ocupa de autenticar al cliente ante el servidor siguiendo el protocolo adecuado. De esta forma, Factotum hace que la lógica relacionada con la seguridad se pueda separar de la lógica de la propia aplicación. Por ejemplo, esto es así para la autenticación del protocolo 9P [3] utilizado por los servidores de ficheros de Plan 9.

Un esquema de SSO centralizado y local a una máquina no resuelve el problema de obstrucción al usuario en un sistema ubicuo fraccionable. Cuando un humano está utilizando varias máquinas simultáneamente, incluso si usa un sistema de *Single Sign-On* local a todas ellas, se debe autenticar al menos una vez por máquina. En esos casos el problema de obstrucción aparece de nuevo. Si el usuario está trabajando en un sistema diseñado para hacer el uso simultáneo de múltiples máquinas transparente, entonces esa propiedad desaparecerá cuando el usuario deba autenticarse de una forma explícita en cada uno de los ordenadores que desea utilizar.

Por último, los agentes de este tipo de sistemas deben tener conectividad con el servidor central de claves para proporcionar SSO al usuario. Por esta razón, este tipo de esquemas no funciona en particiones sin conexión con el servidor de claves.

SHAD sigue un esquema centrado en el humano y en su servidor personal de autenticación que proporciona acceso simple al sistema en un entorno poco restrictivo en el que los usuarios utilizan el *hardware* que les rodea de una forma cómoda y transparente.

Microsoft .Net Passport [113; 114] ofrece mecanismos de *Single Sign-On* entre distintos dominios administrativos, con el objetivo de operar a gran escala y poder autenticar a los usuarios desde cualquier servidor web de Internet. Por tanto, su objetivo es distinto a los sistemas descritos anteriormente, cuyo principal ánimo era proporcionar SSO de carácter general (esto es, a distintos tipos de aplicaciones y servicios).

Los servidores web que desean utilizar los mecanismos de SSO de Passport deben registrarse en el sistema. Durante este proceso, se negocia un secreto compartido entre el proveedor del servicio y Passport, y se le asigna un identificador único. Dicho secreto es una clave secreta que se utilizará para cifrar datos mediante un algoritmo de criptografía simétrica (en concreto, Triple DES).

El esquema de Passport se basa en el uso de un servicio centralizado de autenticación que expende credenciales a los clientes en forma de *cookies* (para

que, de esa forma, se pueda guardar el estado en el navegador web). Cuando el navegador del usuario accede al servicio web, es redirigido al servidor de Passport para que el usuario se autentique por primera y única vez. Si el servidor de Passport identifica correctamente al proveedor de servicios que ha redirigido al cliente, entonces insta al usuario a que introduzca un identificador y una contraseña. Este proceso se lleva a cabo a través de una conexión TLS. Una vez autenticado el usuario, el servidor de Passport almacena en el navegador web tres *cookies*:

- Un resguardo con el identificador del usuario y una marca de tiempo.
- Información sobre el perfil del usuario.
- Una lista de los proveedores de servicio que ha visitado el usuario.

Acto seguido, el servidor de Passport cifra las dos primeras *cookies* con la clave que comparte con el proveedor del servicio, e incluye dicha información en la URL que devuelve al navegador como redirección. De esta forma, el proveedor del servicio puede comprobar que el cliente ya está autenticado ante Passport.

Una vez que el usuario ha introducido su identificador y contraseña en Passport, no necesitará volver a hacerlo mientras que conserve las *cookies* en el navegador y estas continúen siendo válidas. Si el usuario accede a otro proveedor de servicio, este redirigirá al navegador hacia el servidor de Passport, que podrá autenticar al usuario mediante el resguardo guardado como *cookie*. Si el resguardo es válido, el servidor actúa como en el caso expuesto anteriormente, redirigiendo al navegador a una URL del proveedor de servicio que incluye la información cifrada de autenticación.

El esquema de autenticación es semejante al usado en otros sistemas, por ejemplo en Kerberos, por tanto adolece de los mismos problemas para su aplicación en sistemas fraccionables. Además, Passport contempla un único dominio: mantiene una base de datos que almacena los secretos de todos los proveedores de servicios y usuarios, mientras que en otros sistemas es posible

separar la autenticación en distintos dominios. Teniendo en cuenta el ámbito del sistema (toda Internet), la centralización es masiva y peligrosa, como describen Lopez et al. [100] y Oppliger [130]. Además, mediante el esquema propuesto en Passport, la aplicación cliente (el navegador) delega totalmente la autenticación en el servidor de Passport, y no demuestra al proveedor del servicio su conocimiento de la autenticación mediante el uso de *authenticators*. Este hecho supone un riesgo de seguridad, debido a que da a lugar a ataques de reutilización de credenciales antiguas. Este y otros problemas relacionados con Passport fueron tratados por Kormann et al. [93].

CorSSO [85] es un sistema de *Single Sign-On* distribuido propuesto para entornos *Peer-to-Peer* que a primera vista podría parecer adecuado para un entorno como el que tratamos en este trabajo. En CorSSO los clientes no dependen de un solo servicio centralizado como los sistemas descritos anteriormente.

CorSSO propone el uso de criptografía de umbral [45] (*Threshold Cryptography*) para repartir claves entre distintos servidores de autenticación, de tal forma que un programa que actúa como cliente no depende de un único servidor de autenticación para acceder a un servicio en concreto.

En este sistema se propone eliminar la autenticación de las aplicaciones y reemplazarla en los servidores de autenticación. Cuando un cliente necesita acceder a un servicio, accede a los servidores de autenticación que haya disponibles en ese momento. Si el cliente es capaz de recuperar de los servidores disponibles un número suficiente de claves, entonces el cliente podrá autenticarse ante el servicio mediante un método de retos. El servidor debe distribuir previamente las claves necesarias entre los distintos servidores de autenticación. Para distribuir y recuperar esas claves, CorSSO adopta un esquema de nombrado propio. Los servidores de autenticación pueden traducir de ese esquema de nombrado propio de CorSSO a los esquemas de nombrado específicos del servicio y al esquema de nombrado específico del servidor de autenticación. El cliente se deberá autenticar ante los servidores de autenticación para obtener las claves

de cifrado de umbral. Esa autenticación se realiza usando esquemas comunes, como certificados PKI o contraseñas.

CorSSO soluciona el problema de dependencia de un sólo servidor de claves por parte de los clientes, ya que estos dependen de un subconjunto de los servidores de autenticación que poseen claves para el servicio al que necesitan acceder. Sin embargo, no soluciona el problema de autenticación en localizaciones aisladas, ya que el cliente debe tener conectividad con un subconjunto de los servidores que poseen las claves necesarias. Por tanto, no es *Peer-to-Peer*. Por lo contrario, nuestra propuesta sí funciona en esas situaciones.

Alternativamente, si los servicios usasen criptografía de clave pública común en lugar de criptografía de umbral, el programa cliente no dependería de un subconjunto de servidores de autenticación, sino que dependería sólo un servidor de dicho conjunto. De esa forma el cliente podría autenticarse en situaciones adversas en las que no posee conectividad con una gran parte del sistema. Además, un adversario podría autenticarse haciéndose únicamente con las claves de un único servidor de autenticación, lo que hace al sistema más inseguro. Por tanto CorSSO está sacrificando la disponibilidad del servicio por seguridad ante ataques en los servidores de autenticación.

CorSSO tampoco ofrece *Single Sign-On* real a los usuarios que usan varios dispositivos simultáneamente, ya que estos se deberán autenticar al menos una vez por cada máquina que utilizan (dependiendo del esquema usado para que el usuario se autentique ante los servidores de claves).

Por estas razones, CorSSO no parece apropiado para entornos ubicuos en los que los usuarios interaccionan y comparten dispositivos y recursos a la vez que utilizan servicios y sistemas convencionales. Finalmente, el esquema que propone CorSSO necesita la modificación de los programas servidores, clientes y de los servidores de autenticación. Este hecho implica incompatibilidad con los sistemas actuales y dificultad de implantación.

2.4. Dispositivos de autenticación móviles

El uso de dispositivos móviles con el fin de autenticar a los usuarios de un entorno ubicuo lo han propuesto distintos trabajos de investigación y productos comerciales. A continuación se describirán algunos de los dispositivos de este tipo y después se comentarán algunos de los esquemas de uso planteados en la literatura.

2.4.1. Tipos de dispositivos

Las tarjetas inteligentes o **Smartcard** [148], los **iButtons** [76] y los **USB Tokens** [157] son dispositivos que permiten almacenar certificados digitales, claves, datos sobre información biométrica, nombres de usuario y contraseñas. Estos dispositivos son capaces de ejecutar algoritmos de cifrado de clave simétrica y asimétrica. También se ha propuesto el uso de tarjetas PC-Card (tarjetas PCMCIA o Cardbus) para el cifrado de datos personales.

Este es el caso de **Fortezza** [56], que consiste en una tarjeta con una interfaz PC-Card que se compone de un procesador y una memoria que dependen de la versión de la tarjeta.

Algunos modelos son capaces de cifrar usando algoritmos de Tipo 1 según la clasificación de la Agencia Nacional de Seguridad de los EE.UU. [126]. Estos dispositivos se usan en entornos militares con el fin de cifrar comunicaciones habladas realizadas mediante terminales especiales de telefonía. Este tipo de tarjetas son costosas y dependen del interfaz PCMCIA, por lo que muchos dispositivos móviles tales como ordenadores de mano y teléfonos móviles no pueden usarlas.

Otro problema de estos dispositivos es que la mayoría no poseen capacidad de comunicación inalámbrica, de tal forma que el usuario debe insertarlos en lectores o en los puertos de las máquinas para poder autenticarse. Por tanto, cuando un usuario usa este tipo de dispositivos para autenticarse ante un ordenador remoto, no sólo debe confiar en el dispositivo de autenticación sino

que también debe confiar en la máquina en la que lo ha insertado, que actúa como intermediaria entre el dispositivo y el ordenador remoto.

El problema es que estos dispositivos provocan obstrucción, ya que la autenticación por parte del usuario es explícita: el usuario necesita insertar físicamente el dispositivo en un lector para autenticarse. Si el usuario necesita autenticarse en múltiples ocasiones, al menos deberá insertar físicamente el dispositivo tantas veces como máquinas use.

Las **etiquetas RFID** son dispositivos de pequeño tamaño y forma (etiquetas, pegatinas etc.) que incorporan una antena, posiblemente con una pequeña memoria, y en ocasiones un chip que puede procesar datos. La información almacenada se lee desde otro dispositivo (lector de RFIDs). Existen dos tipos de dispositivos RFID: pasivos y activos. Los dispositivos pasivos son aquellos que no tienen fuente de alimentación propia y adquieren la energía de la corriente inducida en la antena por la recepción de la radio frecuencia, suficiente para responder al mensaje recibido. Los dispositivos activos son aquellos que sí poseen fuente de energía propia, y por tanto pueden tener algo de capacidad de procesamiento y almacenar más información que los pasivos. Los dispositivos pasivos son mucho más baratos que los activos y se proponen como una alternativa a los actuales códigos de barras. Estos dispositivos resultan muy útiles para identificar a usuarios, pero su gran limitación de procesamiento hace que no sean apropiados para proporcionar autenticación.

Personal Server [75] es un dispositivo móvil de muy reducido tamaño que permite ejecutar múltiples aplicaciones específicas. Este dispositivo también se ha propuesto como plataforma para entornos ubicuos. Su principal problema es que no es un dispositivo de propósito general. Los dispositivos destinados sólo a la seguridad (o a otras funciones específicas) no captan la atención del humano, que tiende a olvidarlos o extraviarlos [7]. En nuestra opinión, los dispositivos móviles de autenticación deben ofrecer además otros servicios generales para que el usuario les preste más atención.

Al-Muthadi et al. [7] proponen el uso de un reloj de pulsera Matsucom's

OnHand^{tm} PC [129] dotado de capacidad de procesamiento y de un puerto de infrarrojos (IRdA) para autenticar usuarios. En este caso, el uso de comunicación por infrarrojos puede provocar obstrucción al usuario, ya que hay que hay que dirigir el haz de infrarrojos al dispositivo receptor. El uso de tecnologías basadas en radiofrecuencias parece más adecuado para proporcionar servicios de autenticación sin obstrucción. Aunque las comunicaciones de este tipo se puedan interceptar fácilmente, un correcto uso de las herramientas criptográficas disponibles hoy en día puede proporcionar un gran nivel de confidencialidad e integridad.

Actualmente, existe una gran cantidad de modelos de teléfonos móviles y ordenadores de mano equipados con este tipo de tecnologías de comunicación inalámbrica. SHAD se basa en el uso de un ordenador de mano o un teléfono móvil *Smart Phone* de propósito general dotado de varias tecnologías de comunicación por radiofrecuencias como Wi-Fi [195], Bluetooth [69] y GPRS [187]. Este dispositivo, a la vez de ofrecer al usuario la funcionalidad habitual, actúa como un servidor personal de seguridad y ofrece una entrada única al sistema (SSO). Por esa razón, es de esperar que el usuario lo tenga en consideración, ya que no sólo es esencial para su autenticación, sino que lo es para su trabajo diario (p.e. como teléfono, agenda o lector de correo electrónico). Además, los dispositivos de este tipo tienen capacidades de procesamiento menos limitadas que la mayoría de los dispositivos descritos anteriormente.

Los ordenadores de mano y *Smart Phones* vienen provistos de sistemas operativos muy completos, como Symbian [107], PalmOS [151] y Windows Mobile [96]. Estos sistemas operativos ofrecen interfaces de programación para el uso de las tecnologías de comunicación y las herramientas criptográficas, por lo tanto es más sencillo implementar los protocolos de seguridad en ellos que en otros dispositivos citados con anterioridad.

2.4.2. Esquemas de uso propuestos

Beafour et al. [25] presentan un sistema para la apertura automática

de cerraduras que sustituye a las llaves tradicionales. El sistema se basa en dispositivos móviles (en concreto teléfonos móviles) y en cerraduras electrónicas. Ambos tipos de dispositivos se pueden comunicar mediante Bluetooth. Este trabajo propone el uso del teléfono móvil (que denominan "Servidor Personal") para autenticar al usuario ante actuadores (las propias cerraduras). En el sistema, los dispositivos móviles utilizan certificados basados en una infraestructura de clave pública X.509 [74] para autenticarse ante las cerraduras. Las cerraduras no se conectan a la entidad certificadora ni a ningún servicio expendedor de claves, por lo que tienen que actualizarse manualmente cada cierto periodo de tiempo.

Este trabajo propone una solución a medida para un problema concreto, pero no propone una solución para sistemas ubicuos completos o entornos de trabajo actuales. Además, este sistema introduce tareas de administración que lo hacen poco cómodo de usar, por ejemplo la renovación manual de las claves cada poco tiempo. Por el contrario, SHAD presenta una arquitectura que permite solucionar problemas generales de autenticación, y a su vez proporciona servicios de autenticación para problemas específicos como el tratado en este trabajo.

The Master Key [200] propone igualmente un dispositivo que autentica al usuario en el entorno ubicuo para abrir cerraduras. El usuario debe apretar un botón cuando está cerca de una cerradura. En ese momento, el dispositivo radia mensajes que autentican al usuario. Los mensajes del protocolo se cifran mediante una clave compartida entre la cerradura y el dispositivo. Al igual que el trabajo anterior, The Master Key ofrece una solución para un problema específico, no una solución general para controlar la seguridad del usuario en el entorno ubicuo.

Corner et al. [39] proponen el uso de un dispositivo móvil con capacidad de procesamiento y de comunicación inalámbrica para autenticar a los usuarios ante las aplicaciones tras periodos de inactividad. El objetivo del trabajo es asegurar las aplicaciones que utiliza un usuario cuando este deja de estar presente de una forma transitoria. Un usuario deja de estar presente si se pierde el enlace

inalámbrico entre la máquina que ejecuta la aplicación y el dispositivo móvil. Cuando esto sucede, la aplicación deja de ejecutar y se cifra su memoria, que puede contener datos sensibles como contraseñas o datos personales. A la vez, el uso del dispositivo móvil proporciona comodidad al usuario, que no debe introducir contraseñas (tan sólo las que requiere el dispositivo móvil).

Los autores presentan dos formas de proteger la memoria de las aplicaciones:

- La protección transparente a la aplicación consiste en el cifrado de la memoria correspondiente a la aplicación que se desea proteger. Esto no obliga a modificar las aplicaciones ya existentes. Las claves con las que se cifra la memoria de la aplicación residen en el dispositivo móvil. Cuando una aplicación deja de tener enlace con su dueño, pasa a estado de hibernación (por tanto, no puede seguir ejecutando). Antes de volcar la memoria a disco, se cifra con la clave proporcionada anteriormente por el dispositivo móvil. Cuando el usuario vuelve, la memoria del proceso se descifra y se abandona el estado de hibernación. En ese momento, el proceso vuelve a estar listo para ejecutar.

  Esta técnica precisa de una gran cantidad de memoria, debido a que la memoria cifrada debe convivir con la memoria descifrada en los procesos de hibernación y de reanimación. Con esta aproximación también existen problemas con aplicaciones que se comunican a través de memoria compartida y a las aplicaciones que dependen de conexiones de red.

- La otra técnica de protección propuesta obliga a modificar las aplicaciones antiguas y a programar las nuevas teniendo en cuenta la seguridad. Al contrario que la anterior técnica, esta no se basa en suspender las aplicaciones del usuario. Los autores ofrecen una biblioteca de programación que facilita mecanismos para cifrar las partes sensibles de la memoria de la aplicación, por ejemplo, las estructuras de datos que almacenan claves. Esta técnica incrementa la latencia en la ejecución de los programas.

Este trabajo es otra solución a medida en la que se trata el problema específico de proteger la memoria de las aplicaciones, pero no es un esquema de seguridad completo.

Como sucede en el caso anterior, el esquema propuesto en SHAD puede afrontar problemas específicos como el tratado en este trabajo, pero a la vez puede proporcionar mecanismos generales de autenticación, confidencialidad, integridad y una entrada única al sistema (Single Sing-On) en un entorno ubicuo.

El trabajo presentado por **Bussard et al.** [33] presenta un protocolo para la autenticación de dispositivos y de usuarios en entornos ubicuos basado en la localización de los dispositivos (en concreto la distancia física entre los dispositivos). Si entre el dispositivo de autenticación y el dispositivo que se desea usar hay una distancia física muy corta (del orden de centímetros), entonces se puede autenticar al usuario evitando ataques de "hombre en medio" (*Man-in-the-middle Attack*). Los autores proponen el uso de dispositivos tales como anillos electrónicos (Java Ring) [81] o iButtons [76].

Cuando el anillo necesita autenticarse ante otro dispositivo, determina la distancia física que los separa midiendo el tiempo de respuesta a unos mensajes que retransmite por un enlace inalámbrico. Esos mensajes son retos de un bit a los que se le responde con mensajes del mismo tamaño. El valor del bit de respuesta a cada reto depende de un secreto compartido entre el anillo y el dispositivo. Si las respuestas son correctas y lo suficientemente rápidas, el poseedor del anillo se autenticará correctamente. El secreto compartido entre el anillo y el dispositivo se intercambia en una fase previa poniendo en contacto físico el anillo con un lector (integrado en el dispositivo). El contacto físico entre los dos dispositivos avala el intercambio, ya que ningún intruso podría estar en contacto con el lector y pasar inadvertido.

En un entorno ubicuo puede que no haya dispositivos a la vista, o puede que no estén físicamente cercanos (p.e. una impresora que está en otra habitación). De todas formas, la autenticación basada en la cercanía puede resultar muy útil para autenticar a los usuarios ante los terminales.

En el trabajo no se explica cómo se controla el acceso a los recursos, y se trata sólo la autenticación. En este caso nos encontramos ante otro sistema hecho a medida para solucionar un problema concreto (la autenticación ante terminales). Nuestro enfoque es más general. SHAD proporciona el mismo grado de libertad al usuario que el sistema propuesto en este trabajo: el usuario no debe introducir contraseñas o autenticarse de forma explícita cuando desea usar un terminal en el entorno ubicuo. A lo sumo, debe confirmar la operación apretando un botón de su UbiTerm.

Vadja et al. [189] proponen una serie de protocolos para autenticación usando etiquetas RFID pasivas. Estos dispositivos son muy baratos (lo suficiente como para reemplazar a los actuales códigos de barras) y tienen la capacidad de emitir señales de radiofrecuencia, procesar información (aunque de una forma muy limitada) y almacenar información del orden de centenas de bits. El problema es que estos dispositivos no pueden usar protocolos de seguridad normales, debido a sus grandes limitaciones. Las etiquetas RFID no tienen capacidad para ejecutar algoritmos comunes de cifrado o de resumen. Los autores proponen varios protocolos de autenticación que utilizan retos, suponiendo que las etiquetas RFID y el lector comparten un secreto de antemano.

En SHAD consideramos el uso de dispositivos móviles con una capacidad de procesamiento muy superior al de las etiquetas RFID y con memoria del orden de decenas de Megabytes. Por esta razón, SHAD usa protocolos de autenticación que se construyen sobre un algoritmo de clave simétrico estándar.

Shared PC [80] propone un sistema que permite mover las sesiones de los usuarios entre distintas máquinas para que así puedan acceder a un único escritorio desde cualquier ordenador público conectado a la red. En este trabajo proponen el uso de tarjetas inteligentes (*Smartcards*) para autenticar al usuario y descifrar sus datos personales (p.e. contraseñas, ficheros de datos o *cookies* del navegador WWW).

Para guardar las sesiones se utiliza un ordenador que denominan Shared PC. El Shared PC debe estar conectado constantemente a la red y actúa como

un servidor personal para el usuario. En él se almacenan los datos del usuario cifrados con una clave que se guarda en la tarjeta inteligente.

Cuando un usuario llega a un PC público, debe insertar la tarjeta que almacena la clave con la que se descifrarán la información de la sesión y sus ficheros. Para autenticarse ante la tarjeta, el usuario debe introducir una contraseña. Durante la sesión, los ficheros modificados se actualizan en el servidor. Los datos creados en el ordenador público se borrarán cuando el usuario acabe la sesión.

Mediante este esquema de uso el usuario debe introducir la contraseña cada vez que inicia la sesión en un ordenador público. Además, este sistema restringe el uso de la información del usuario a ordenadores que tengan la suficiente capacidad de almacenamiento, comunicación y procesamiento como para recibir los datos del usuario, descifrarlos y almacenarlos temporalmente. Por esa razón, el uso de dispositivos tales como ordenadores de mano y teléfonos móviles puede estar excluido.

Kopp et al. [92] abogan por el uso de *middleware* para afrontar el problema en entornos ubicuos. Para solucionar el problema de la entrada única al sistema (*Single Sign-On*), los autores proponen el uso de un dispositivo móvil con interfaz USB llamado CryptoToken. Este dispositivo dispone de un procesador integrado con el que se cifran y descifran los datos. Además posee una memoria donde se almacenan una serie de certificados que se utilizarán para la autenticación.

Este esquema sigue provocando obstrucción al usuario, ya que este debe introducir este dispositivo en las máquinas que desea utilizar. Cuando un usuario necesita utilizar más de una máquina al mismo tiempo, resurge el problema de entrada única al sistema. Para evitarlo, el usuario debería poseer más de un CryptoToken. Este hecho potenciaría el riesgo de pérdida del dispositivo de autenticación. Por último, el sistema adolece de los problemas de interoperabilidad generados por el uso de *middleware* ya comentados en puntos anteriores.

SECURE Sign-On [170] propone un esquema de SSO en el que los

secretos del usuario se almacenan en tarjetas inteligentes *Smartcard*. En ese caso, el cliente deberá introducir la tarjeta en un lector en lugar de teclear contraseñas. Como ya dijimos, la introducción de tarjetas causa obstrucción al usuario, aunque en menor grado que las contraseñas. Además, hace imposible la autenticación en concurrente en distintas máquinas (a no ser que se repliquen las tarjetas, lo que supone un gran riesgo).

2.5. Esquemas de confianza basados en el humano

SHAD propone un esquema de seguridad en el que los usuarios plasman su noción de confianza en otros humanos de una forma explícita, sencilla y muy aproximada a las interacciones sociales de la vida real. En nuestro sistema, los usuarios se relacionan en el entorno ubicuo y emparejan sus servidores personales de seguridad. Después regulan el grado de confianza en los otros humanos con los que han emparejado sus dispositivos asignando roles y atributos en su servidor personal.

La necesidad de plasmar de forma explícita el nivel de confianza ya se ha tratado en otros trabajos relacionados. **Langheinrich** propone en [97] cambiar el concepto de confianza en los entornos ubicuos, y dejar en manos del humano la elección de las entidades en las que confía. El autor defiende en este trabajo el uso directo de la noción de confianza del humano, que es altamente subjetiva, en lugar de tratar de estimarla sin intervención humana. Esta es la linea que intentamos adoptar en SHAD mediante servidores personales de autenticación.

Creese et al. [42] sostienen que hay que tener en cuidadosa consideración al humano a la hora de medir la confianza. En este trabajo se enumeran varios factores por los que el humano debe tener el papel de especificar el nivel de confianza:

- Los humanos se preocupan por sus recursos de valor.

- Los humanos quieren mantener el control sobre sus pertenencias valiosas.
- Los humanos suelen usar mecanismos de seguridad electrónica (contraseñas etc.).

Los autores también enumeran argumentos en contra de delegar la evaluación de confianza y la seguridad en el humano:

- Es difícil especificar la noción de confianza a la máquina y la interfaz entre el humano y la máquina no ser muy efectiva.
- La mayoría de las vulnerabilidades se deben a la irresponsabilidad de los humanos.
- Las contraseñas elegidas por los humanos tienden a ser frágiles.
- El deseable que las máquinas realicen las tareas frecuentes y gravosas.

Teniendo en cuenta estos factores, es posible elaborar un sistema que plasme la noción de confianza del humano en el sistema. El principal reto es la elaboración de una interfaz que permita expresar esta noción de una forma sencilla y lo más exacta posible.

SHAD propone una arquitectura centrada en el humano que utiliza un esquema sencillo y natural para expresar la confianza.

2.6. Seguridad en redes *Peer-to-Peer* y *Grid Computing*

En el ámbito de las redes *Peer-to-Peer* y los entornos de computación *Grid* también se han propuesto esquemas de confianza y arquitecturas de seguridad. A continuación se describirán algunas de las arquitecturas propuestas para este tipo de sistemas. Aunque or lo general, estos esquemas son difícilmente aplicables a un entorno ubicuo.

Igual que algunos de los trabajos propuestos para entornos ubicuos, **Secure PC** [88] propone un esquema para evaluar la confianza en entidades externas basado en estimaciones para crear un conjunto de entidades fiables. En este esquema las entidades pueden propagar la noción de confianza. Pero esto entra en contradicción con la opinión de otros autores respecto a la falta de transitividad en la relación de confianza [84].

El modelo se basa en que las entidades con las que ya se ha intercambiado información normalmente se consideran más fiables que las demás. Creemos que esa hipótesis no tiene por qué ser cierta en todos los casos, por ejemplo, cuando una entidad fiable pasa a ser de pronto una entidad maliciosa. Como ya hemos anotado anteriormente, creemos que la noción de confianza debe estar centrada en el humano, y este debería fijarla de una forma explícita.

En **GridBox** [50] se presenta un sistema de control de acceso para entornos de *Grid Computing*. El sistema es un complemento a los mecanismos de seguridad existentes tanto para comunicaciones, como por ejemplo protocolos de transporte seguros como SSL [40] o TLS [48], como para la seguridad de los propios servicios, por ejemplo Java Sandbox [63], UNIX Jails [87] o UNIX Chroot [27].

En GridBox proponen un esquema de control de acceso basado en listas de control (ACL) que definen el acceso de los procesos a los recursos del sistema operativo. El control de acceso se realiza a tres niveles:

1. **Perfil del nodo:** ACL que define los permisos generales en un nodo del Grid.

2. **Perfil de la aplicación:** define el control de acceso de una determinada aplicación para el espacio de nombres, la red, la creación de procesos y la ejecución de programas.

3. **Perfil de la máquina:** define aspectos relacionados con los recursos de la máquina, por ejemplo, poniendo límite al número de procesos que puede ejecutar, etc.

Gridbox se basa en una capa de *middleware* que intercepta las llamadas al sistema de los procesos para aplicar el control de acceso. Por ejemplo, para controlar las llamadas al sistema de apertura de ficheros, se puede proporcionar al *middleware* una lista de control de acceso (ACL) que especifique qué ficheros puede abrir cada proceso del Grid. Este sistema es difícil de administrar y no es aplicable a un entornos como el que tratamos en este trabajo. Además, interceptar llamadas no garantiza que las originales no se puedan usar.

Detsh et al. [46] proponen un marco llamado P2PSLF para Java JXTA [64] que provee seguridad a una red de *Grid Computing*. Siguiendo una organización jerárquica, cada isla del *Grid* mantiene un representante de tal forma que la red *Peer-to-Peer* es en realidad la red formada por ellos. Para cada par de nodos y sentido de la comunicación se pueden especificar las necesidades de seguridad (sólo autenticación, autenticación más confidencialidad, etc.).

Para proporcionar los servicios de seguridad se introduce una capa de *middleware* entre la aplicación y JXTA. Los mecanismos de seguridad son módulos del *framework*, y cada nodo puede tener un subconjunto de ellos. Cuando un nodo necesita servicio, emite mensajes de radiado que contienen la petición. El cliente recibe la respuesta de los nodos que están capacitados para proporcionar el servicio requerido.

Cuando un nodo ha elegido a otro para la operación, se selecciona algún mecanismo de seguridad que tengan en común. Si un nodo no implementa mecanismos de seguridad, puede usar los servicios de forma insegura (se mantiene la compatibilidad con nodos antiguos).

En el trabajo no se especifica cómo intercambian las claves los distintos nodos ni dónde residen los secretos, por lo que no se resuelven los principales problemas que aparecen en un entorno como el descrito en los escenarios que consideramos nosotros.

GSI (*Grid Security Infrastructure*) [194] es el esquema de seguridad que usa el conjunto de herramientas **Globus Toolkit** [62] de la Globus Alliance [60]. GSI se basa en criptografía asimétrica, y utiliza firmas electrónicas y certificados

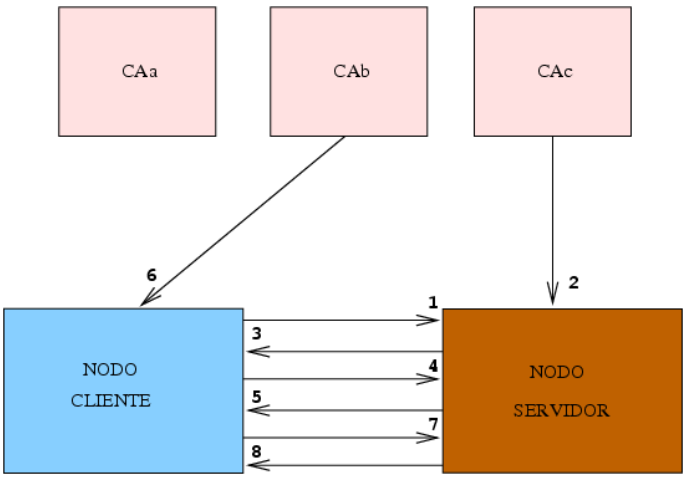

Figura 2.4: *Esquema del protocolo GSI.*

X.509 [74] para proporcionar autenticación entre los nodos que forman el *Grid*.

Por defecto, GSI no proporciona confidencialidad en la comunicación entre los nodos para aumentar el rendimiento del Grid, pero sí proporciona la integridad de los datos transmitidos. Al usar certificados para la autenticación, los nodos deben confiar en autoridades certificadoras (CA) comunes.

Por tanto, cuando se configura un nodo, las CAs elegidas por el usuario deben firmar su certificado. Los otros nodos deben obtener los certificados en los que se especifica la clave pública de las autoridades certificadoras elegidas. Por lo tanto, esos nodos deben confiar en que esos certificados son correctos y no se han comprometido en ningún momento. Las claves privadas de todos los nodos están protegidas mediante contraseñas o tarjetas inteligentes.

La autenticación entre nodos sigue el siguiente protocolo (ver figura 2.4):

1. El nodo cliente se conecta al nodo servidor y le entrega su certificado firmado por una entidad certificadora, por ejemplo CAa. Ese certificado tiene como objetivo demostrar que el nodo cliente es quien dice ser y no otra entidad, y contiene el identificador del nodo, su clave pública y el identificador de la autoridad certificadora que se desea utilizar. Todos estos datos están firmados por la autoridad certificadora.

2. El servidor intenta comprobar que el certificado es correcto. Si el servidor confía en esa autoridad y posee su certificado, puede comprobar que el certificado que el cliente le ha enviado es correcto verificando la firma de la entidad certificadora.

3. Cuando el servidor confía en que el certificado es de quien dice ser, envía un mensaje de reto con datos aleatorios al cliente.

4. El cliente debe cifrar los datos del reto con su clave privada y enviar el resultado al servidor. Después, el servidor descifra los datos recibidos y comprueba que están cifrados con la clave privada del cliente. Así, el servidor puede comprobar que el cliente posee la clave privada y está dispuesto a continuar con la conexión.

5, 6 y 7. Una vez que el servidor confía en que el nodo cliente inició la sesión y que es quien dice ser, se repiten los tres primeros pasos en sentido inverso (del servidor al cliente) para que el cliente pueda autenticar al servidor.

GSI proporciona mecanismos para delegar privilegios y así evitar que el usuario tenga que introducir contraseñas en todos los nodos (*Single Sign On*). Para ello, el sistema utiliza unos certificados especiales, que se denominan "certificados representantes". Los nodos crean y firman sus propios certificados representantes con su clave privada. Los certificados contienen la identidad del usuario, una nueva clave pública y su tiempo de vida. La nueva clave privada correspondiente a ese certificado representante se almacena en el nodo creador. En este caso, el protocolo de autenticación cambia, ya que se establece una cadena de confianza entre los distintos certificados. El nodo servidor recibe el certificado del dueño junto al certificado representante. Entonces, el servidor comprueba con la clave pública que contiene el certificado del dueño que el certificado representante está firmado con la clave privada de este. Después, se utiliza la clave pública de la entidad certificadora para validar el certificado del dueño.

La política de control de acceso se aplica mediante un protocolo llamado CAS (*Community Authorization Service*). El control de acceso se centraliza en un servidor CAS que provee servicio para un grupo de usuarios y expende certificados GSI para representar a los miembros del grupo ante los dueños de los recursos. Los dueños de los recursos especifican los grupos en los que confían. El acceso a los recursos se especifica mediante mecanismos comunes tales como cuotas o atributos en ficheros.

Además, el servidor CAS tiene sus propias políticas de control de acceso y de cesión de privilegios. Cuando un nodo desea usar un recurso, se pone en contacto con su servidor CAS y le comunica la operación que quiere realizar. Si la política de control de acceso del servidor CAS determina que el usuario tiene privilegios para acceder al recurso, enviará al cliente un certificado representante GSI especial que contiene el permiso para realizar las operaciones requeridas. Después, el usuario contacta con el dueño del recurso y le presenta el certificado representante. El dueño del recurso aplicará entonces su propia política de control de acceso para ese grupo de usuarios. Después aplicará las restricciones que se especifican en el certificado representante. La operación se llevará a cabo si ambos grupos de restricciones lo permiten.

Globus depende de múltiples servicios centralizados (p.e. las autoridades certificadoras y los servidores CAS). Además, en el caso en el que se utilizan mecanismos para reducir la obstrucción en su uso, hay que confiar en que los dispositivos de almacenamiento de los nodos sean seguros (ya que el compromiso de un certificado representante pone en peligro todo el proceso de autenticación). Por último, el uso de cifrado asimétrico no lo hace adecuado para dispositivos con limitaciones de consumo de energía y de procesamiento.

2.7. Tabla comparativa

A continuación se presenta una tabla comparativa (cuadro 2.2) en la que aparecen los sistemas más relevantes que se han presentado en este capítulo y

las cualidades más importantes para que se ajusten a las necesidades que creemos tiene un entorno ubicuo que sea dinámico, fraccionable y heterogéneo.

La primera columna indica si el sistema es independiente de entidades centralizadas. La segunda indica si el sistema interopera correctamente y puede usarse con aplicaciones antiguas con facilidad. La tercera muestra si el sistema se puede usar en dispositivos móviles con limitaciones de consumo de energía y procesamiento. Esta cualidad se evalúa en base a los algoritmos de cifrado que puede utilizar y a la cantidad de mensajes que utiliza. La quinta columna indica si el esquema es aplicable para ceder dispositivos entre humanos sin aportar obstrucción. En el caso de que la respuesta a la quinta columna sea positiva, la sexta columna indica la simplicidad de la administración de dicho sistema. La séptima columna muestra si el sistema aporta SSO real en un espacio inteligente. La octava y última columna muestra si el sistema saca partido de alguna infraestructura de contexto.

Las celdas vacías indican que carecemos de la información sobre su implementación o que el requisito no es aplicable a ese tipo de sistema. Por ejemplo, para SecureLogin no podemos rellenar su casilla de cesión de dispositivos sin obstrucción porque se trata de un sistema que únicamente proporciona SSO.

Como puede observarse en el cuadro, ninguna de las propuestas descritas cumplen los requisitos más relevantes de los que se han detallado en el capítulo de introducción. Sin embargo, nuestra propuesta sí los cumple, ya que ha sido diseñada especialmente para un entorno ubicuo que se ajusta a un sistema distribuido fraccionable, dinámico y heterogéneo.

	Independencia de entidades centralizadas	Interopera.	Dispositivos móviles	Cesión sin obstrucción	Administración sencilla	SSO real para todos los servicios	Uso de información de contexto
Kerberos	NO	NO	SI	NO	NO	NO	NO
Sesame	NO	NO	SI	NO	NO	NO	NO
Jini DTM	NO	NO	SI	SI	NO	NO	NO
Resurrecting Duckling	SI		SI	NO	SI	NO	NO
Secure Smart Homes	NO	NO	SI	NO	NO	NO	NO
HESTIA	NO	NO	SI	NO	NO	NO	SI
Cerberus	NO	NO	SI	SI	NO	NO	SI
SecureLogin SSO	NO	NO				NO	NO
Factotum	NO	SI	SI			NO	NO
CorSSO	NO					NO	NO
SECURE sign-on	SI	NO				NO	NO
GSI	NO		NO			NO	NO
SHAD	**SI**	**SI**	**SI**	**SI**	**SI**	**SI**	**SI**

Cuadro 2.2: *Comparativa entre los sistemas estudiados y SHAD.*

Capítulo 3

Enfoque, principios de diseño y metodología

Este capítulo describe los principios de diseño que proponemos para SHAD. En el capítulo anterior se describieron las diferentes soluciones propuestas para distintos entornos distribuidos, quedando patente que dichas arquitecturas no proveen una solución general para entornos ubicuos permisivos que se ajusten a un sistema distribuido dinámico, heterogéneo y fraccionable.

Los principios descritos en el presente capítulo son comunes para la arquitectura y los protocolos que serán descritos en capítulos posteriores. Estos principios forman parte de la contribución de este trabajo.

3.1. Arquitectura permisiva a través de un dispositivo personal de autenticación

SHAD debe ser ágil en un entorno *Peer-to-Peer*, en el que el humano es el *Peer*. Cada usuario posee un UbiTerm que le representa en el sistema. La misión principal del UbiTerm es controlar las actividades del usuario y permitirle el uso de los elementos del sistema ubicuo. Por ello, el UbiTerm es el candidato ideal para actuar como un **servidor de autenticación *Peer-to-Peer***, o humano a humano.

El uso del UbiTerm nos permite afrontar las necesidades de no obstrucción, delegación y de soporte de desconexiones. Ya que los usuarios llevan consigo un dispositivo móvil que les representa en el sistema, este se puede encargar de proporcionarles seguridad de una forma transparente y automatizada en la mayoría de las ocasiones. El usuario no está obligado a introducir contraseñas o a utilizar dispositivos de autenticación constantemente, ya que los UbiTerms son capaces de negociar claves sin supervisión humana. El UbiTerm almacena las claves y las autorizaciones necesarias para que los dispositivos del usuario puedan acceder a los recursos del entorno. Las autorizaciones pueden ser credenciales de atributos, resguardos (*tickets*) o cualquier otro tipo de mecanismo de este tipo.

Como el UbiTerm actúa como un servidor de autenticación, puede proporcionar mecanismos de delegación de privilegios y de soporte de

desconexiones a sus clientes. De esa forma, un usuario puede seguir utilizando de forma controlada los recursos ajenos aunque su dueño ya no esté presente.

El UbiTerm proporciona al usuario una entrada única al sistema ubicuo: **el usuario debe proporcionar una sola contraseña** al UbiTerm para poder acceder a todos los servicios (incluidos los servicios cliente/servidor comunes), aunque utilice múltiples dispositivos simultáneamente.

El uso del UbiTerm para aportar seguridad al sistema ubicuo cumple con la necesidad de independencia de entidades centralizadas. Si dos usuarios quieren compartir sus recursos, no necesitan tener conectividad con el resto del sistema: los UbiTerms pertenecientes a los dos usuarios involucrados son los encargados de proporcionar los mecanismos de seguridad a los dispositivos de sus propietarios.

Como se detalla en el capítulo 2 dedicado al estado del arte, el uso de dispositivos móviles ya se ha propuesto en varios trabajos relacionados. El principal problema con estos dispositivos de autenticación es que, como objetos físicos, se pueden pueden extraviar o robar. Si un dispositivo de este tipo se extravía, el poseedor del dispositivo puede autenticarse con una identidad que no le pertenece.

Este es un riesgo importante, pero en realidad aparece con muchos de los objetos que un humano considera valiosos en su vida cotidiana. Comúnmente, todos nosotros dependemos de objetos físicos cuyo robo o extravío puede provocarnos un gran trastorno. Estos objetos pueden ser tarjetas de crédito, llaves para puertas de casa o documentos de identidad. Por lo tanto, estamos aceptando a diario el **compromiso entre comodidad de uso y nivel de seguridad**.

Cuando una persona pierde un objeto que considera valioso, normalmente actúa de inmediato avisando a las autoridades competentes para anular su uso. En el caso de un sistema informático, se puede actuar de la misma manera: revocando las claves que permiten la autenticación ante los demás usuarios.

3.2. Diseño centrado en el humano

Uno de los retos actuales en la computación ubicua es el control de los recursos del sistema para que estos puedan cooperar entre sí. Este problema está directamente relacionado con la noción de pertenencia de los recursos [180]. Mediante un esquema centrado en el humano y en su representante en el sistema (el UbiTerm) es posible gestionar de una forma sencilla la pertenencia de los recursos del entorno ubicuo. El UbiTerm es el encargado de gestionar todos los recursos pertenecientes a un humano y de controlar su cesión a otros usuarios del entorno ubicuo.

En SHAD nos apoyamos en los siguientes principios:

- **Los dispositivos siempre tienen un dueño**. Por ejemplo, la impresora que está situada en el despacho de Francisco pertenece a Francisco.

  Los dispositivos que no son propiedad de un único usuario, sino que son comunes y conciernen a un grupo, pertenecen a usuarios virtuales del sistema. Por ejemplo, los dispositivos situados en una sala común de un departamento, como una pantalla táctil y un proyector de vídeo, pueden pertenecer a un usuario virtual que los comparte de la misma manera que lo haría un usuario real (humano). El UbiTerm del usuario virtual no tiene necesidad de ser un dispositivo móvil, sino que puede ser un ordenador común conectado a la red.

- **Los propietarios de los dispositivos son humanos que pueden confiar en otros humanos**. Siguiendo con el ejemplo anterior, si Francisco confía en Eva, Eva puede utilizar la impresora de Francisco. En SHAD, cuando dos humanos deciden confiar mutuamente, emparejan los dispositivos que los representan en el sistema ubicuo: sus UbiTerms. Nótese que el emparejamiento de UbiTerms es un proceso similar al usado para emparejar dispositivos Bluetooth [69].

En este esquema, la confianza opera entre dos humanos. Para que los dos humanos puedan compartir sus recursos, es necesario que ambos tengan

confianza en la otra parte. Nosotros, al igual que otros autores [84], consideramos que la noción de confianza entre los humanos no es transitiva. Por esa razón, para que dos humanos puedan compartir recursos en SHAD, tienen que haber emparejado previamente sus UbiTerms de una forma explícita.

Es necesario aplicar algún tipo de política de control de acceso a SHAD. Se ha elegido una aproximación basada en roles (RBAC [162]): el humano asocia roles a los otros humanos en su propio UbiTerm. Como ya hemos comentado, la confianza es recíproca, pero se puede graduar en base a los roles que se le asocien a cada uno de los humanos. Si una de las partes tiene un menor grado de confianza que la otra, puede asociarle un rol más restrictivo, ya que los roles en ambas partes son independientes entre sí.

Un diseño centrado en el humano permite plasmar el concepto de confianza en el sistema a la vez que lo mantiene simple. En otros sistemas, la noción de confianza se basa en estimaciones hechas por las máquinas a partir de ciertos factores como la reputación de ciertos usuarios o la información de contexto extraída del sistema ubicuo. En nuestra opinión, mediante esta aproximación no se puede expresar correctamente un concepto tan subjetivo como es el concepto de confianza entre humanos. Nosotros creemos que el usuario tiene que plasmar su noción de confianza de una forma explícita. De otro modo, los humanos desconfiarían del sistema. Además, creemos que la confianza expresada por el humano debe ser, como en la vida real, referente a otro humano o a un grupo de ellos.

3.3. Integración a nivel de sistema operativo

SHAD afronta el problema de interoperabilidad mediante su integración con el sistema operativo a nivel de sistema de ficheros. SHAD adopta las ideas del sistema operativo Plan 9 From Bell Labs [137] aplicadas al entorno de Plan B [21; 20], sistema operativo en el que lo hemos implementado e integrado. En Plan 9 todos los recursos del sistema se representan como ficheros y todos los ficheros

se pueden exportar por la red.

Mediante el uso de sistemas de ficheros en red para exportar la interfaz de los dispositivos, los programas existentes pueden sacar partido de los nuevos servicios del espacio inteligente sin apenas modificaciones. Esto se debe a que para acceder a los servicios del entorno únicamente hay que usar las llamadas al sistema operativo para manipular una abstracción bien conocida: el fichero. Por lo contrario, los sistemas que se basan en *middleware* o *frameworks* fuerzan a reconstruir o envolver las aplicaciones antiguas para que puedan interoperar con el entorno ubicuo. En estos casos, se reduce el número de aplicaciones que pueden interactuar con el sistema ubicuo, ya que su adaptación conlleva un mayor esfuerzo. También obligan a que las nuevas aplicaciones tengan que implementarse en un lenguaje y/o una plataforma específica.

Nosotros adoptamos la misma idea para SHAD. Las aplicaciones que necesitan autenticación para acceder a un recurso (un dispositivo del entorno o un servicio común) interaccionan con SHAD a través de su sistema de ficheros. Por tanto, para que una aplicación pueda usar los servicios de SHAD, únicamente debe leer y escribir en los ficheros de control de su interfaz.

Cuando se intenta operar sobre un fichero remoto para acceder a un dispositivo del espacio inteligente, los dos sistemas involucrados se autentican y controlan el acceso a través de SHAD. La integridad y la confidencialidad de las operaciones sobre el dispositivo quedan en manos del protocolo de sistema de ficheros en red que se utiliza. No obstante, SHAD también puede establecer un canal seguro para ello.

Por último, el diseño de SHAD tiene en consideración que los dispositivos móviles están limitados en lo que respecta a la autonomía de sus baterías y la capacidad de procesamiento y almacenamiento, por lo que sus protocolos de comunicación son ligeros y ahorrativos en lo que respecta a mensajes retransmitidos.

3.4. Escenarios de ejemplo

A continuación presentamos tres escenarios que podrían darse en un entorno ubicuo permisivo. Estos escenarios de ejemplo nos permiten ilustrar los principales problemas con los que trata SHAD. Los escenarios muestran cómo los principios descritos en el punto anterior permiten a SHAD afrontar esos problemas de una manera simple y eficaz.

3.4.1. Escenario 1: uso de múltiples dispositivos y recursos sin obstrucción

En la figura 3.1 podemos observar un escenario conflictivo que se produce a diario en los entornos de computación actuales.

El usuario posee tres dispositivos: una estación de trabajo, un ordenador portátil y un Pocket PC. Cada dispositivo utiliza un sistema operativo diferente, por tanto el esquema es heterogéneo respecto a *hardware* y *software*. Cuando el usuario llega al entorno de trabajo, su ordenador portátil y su estación de trabajo están apagados y debe encenderlos. Tanto la estación de trabajo como el ordenador portátil solicitan al usuario que se autentique para poder iniciar su sesión (pasos 1, 2 y 3). Entonces, el usuario tiene que escribir una contraseña en cada dispositivo.

Ahora imaginemos que el usuario desea usar en todas sus máquinas un sistema de ficheros importado a través de la red desde un servidor del entorno de trabajo. A la hora de montar el sistema de ficheros remoto en sus máquinas, el usuario debe autenticarse en todas ellas de nuevo (pasos 4, 5 y 6). La autenticación en el Pocket PC provoca más obstrucción todavía, ya que el dispositivo de entrada es el reconocedor de escritura y el usuario debe introducir algunos caracteres no alfabéticos que conforman una clave relativamente segura [158].

Como se puede observar, en este escenario la obstrucción al usuario es constante, aun cuando el número de dispositivos que usa es pequeño. En un

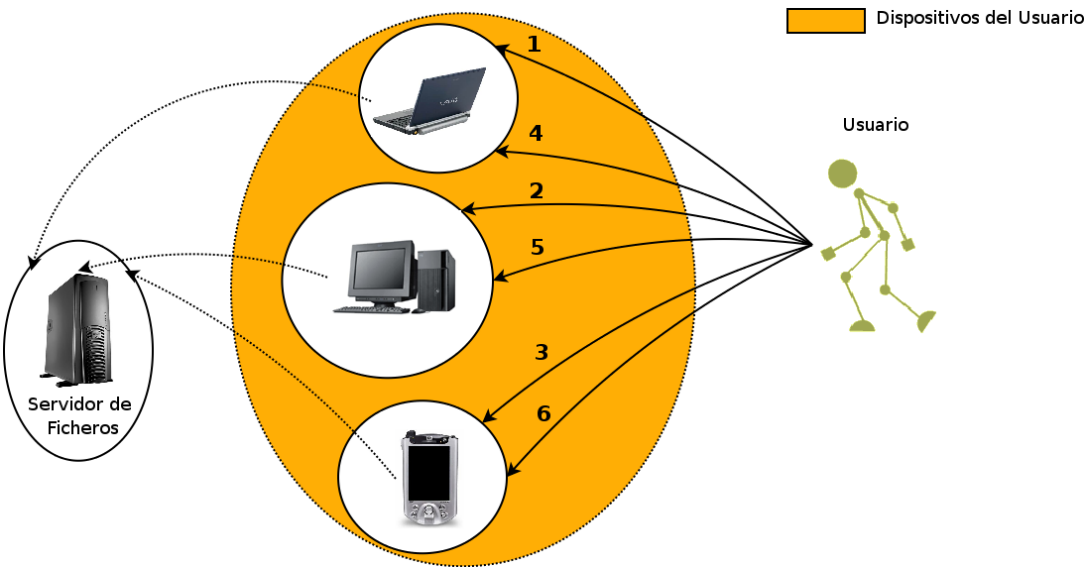

Figura 3.1: *Escenario en el que el usuario debe introducir una contraseña para cada servicio en todos sus dispositivos.*

corto periodo de tiempo, el usuario ha necesitado introducir seis contraseñas para utilizar las máquinas que le pertenecen.

Si en lugar de usar autenticación por contraseñas el usuario utilizase autenticación mediante dispositivos, el usuario tendría que haber introducido su dispositivo de autenticación en tres lectores diferentes un total de seis veces (suponiendo que hayan disponibles lectores para Pocket PCs).

Si el usuario hubiera usado un sistema de *Single Sign-On* como los que se han descrito en el capítulo 2, tendría que haber introducido al menos una contraseña por dispositivo. En ese caso, el sistema sigue aportando obstrucción, que crece con el número de máquinas que utiliza el usuario.

Ahora supongamos la misma situación en un sistema como SHAD (ver la figura 3.2). El usuario, cuando entra en el entorno ubicuo, arranca todas sus máquinas. El ordenador de mano, que en realidad es su UbiTerm, es el primero en arrancar. En ese momento, el UbiTerm pide que al usuario que se autentique (mediante una contraseña, la lectura de su huella digital o cualquier otro método que acepte el hardware del UbiTerm). Después de esta primera autenticación

(paso 1), el UbiTerm descifra un fichero de claves que tiene almacenado en una tarjeta de memoria externa, por ejemplo en una tarjeta SD Card [168]. El fichero que contiene todos los secretos del usuario está fuertemente cifrado mediante un algoritmo de clave simétrica. Una vez descifrados, los secretos del usuario se almacenan en la memoria de un agente de SHAD que ejecuta en el UbiTerm.

Cuando el usuario arranca su ordenador portátil, el sistema operativo requiere autenticación. En ese momento, un agente de SHAD que ejecuta en el ordenador portátil detecta que el ordenador de mano (el UbiTerm) está activo y su agente posee los secretos del usuario. Mediante un protocolo seguro, el agente de SHAD del ordenador portátil solicita al agente del UbiTerm que le proporcione la contraseña correspondiente. El protocolo se desarrolla de una forma correcta, y el UbiTerm envía la contraseña correspondiente al agente del ordenador portátil (paso 2). Después, el agente de SHAD del ordenador portátil realiza la autenticación por el usuario, pasando la contraseña al sistema operativo e iniciando la sesión local. El proceso de arranque de la estación de trabajo se lleva a cabo de una forma similar (paso 3). Más tarde, las máquinas del usuario necesitan montar el sistema de ficheros remoto y deben autenticarse ante el servidor. Para ello, solicitan al agente del UbiTerm la contraseña del usuario para dicho servidor. El protocolo prospera correctamente y el agente del UbiTerm proporciona la contraseña a los agentes que la solicitaron (pasos 4 y 5). Los agentes de SHAD de las tres máquinas se encargan de autenticar al usuario ante el servidor de ficheros. Al final, el sistema de ficheros se monta correctamente y el usuario es capaz de usarlo desde todos sus dispositivos.

Nótese que hasta el momento el usuario sólo ha tenido que introducir la contraseña que permite al UbiTerm descifrar el fichero que contiene todos los secretos del usuario. Por tanto, **SHAD ofrece SSO real para un espacio inteligente**.

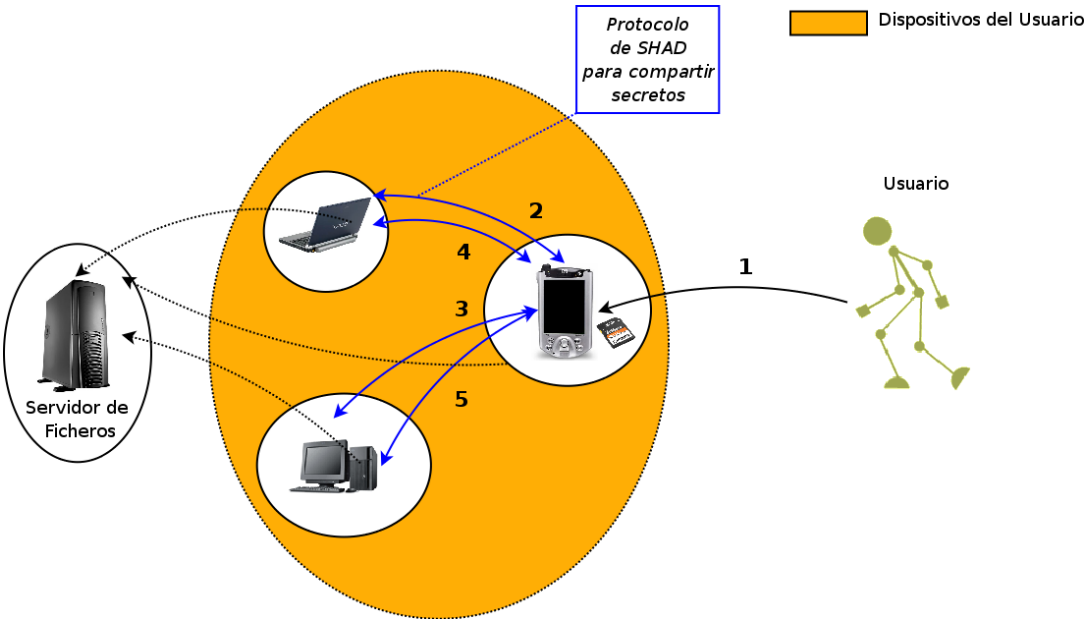

Figura 3.2: *Escenario en el que el usuario usa el UbiTerm para distribuir sus secretos entre las máquinas que está utilizando.*

3.4.2. Escenario 2: facilidad para compartir dispositivos y recursos

Supongamos el escenario descrito en la figura 3.3. Katia y Gorka son usuarios del entorno ubicuo y trabajan juntos en el mismo departamento, por lo que normalmente se encuentran varias veces al día por los pasillos y despachos. En uno de sus encuentros, ambos deciden que confían en el otro y acuerdan emparejar sus UbiTerms. En ese momento, sitúan sus UbiTerms uno en frente del otro y, mediante una comunicación vía infrarrojos (que impone estar cercano físicamente y por tanto incrementa la seguridad ante ataques de "hombre en medio"), los dispositivos negocian un secreto compartido y se emparejan.

Es importante remarcar que esta no sería la única forma de emparejar dos UbiTerms. Para ello, los usuarios podrían utilizar varias técnicas con distintos grados de seguridad, como por ejemplo el intercambio de los secretos por correo electrónico cifrado con PGP [201], charlas mediante un programa de *chat* sobre un canal seguro (SSH [199] o SSL [40]) o cualquier otro método que el usuario

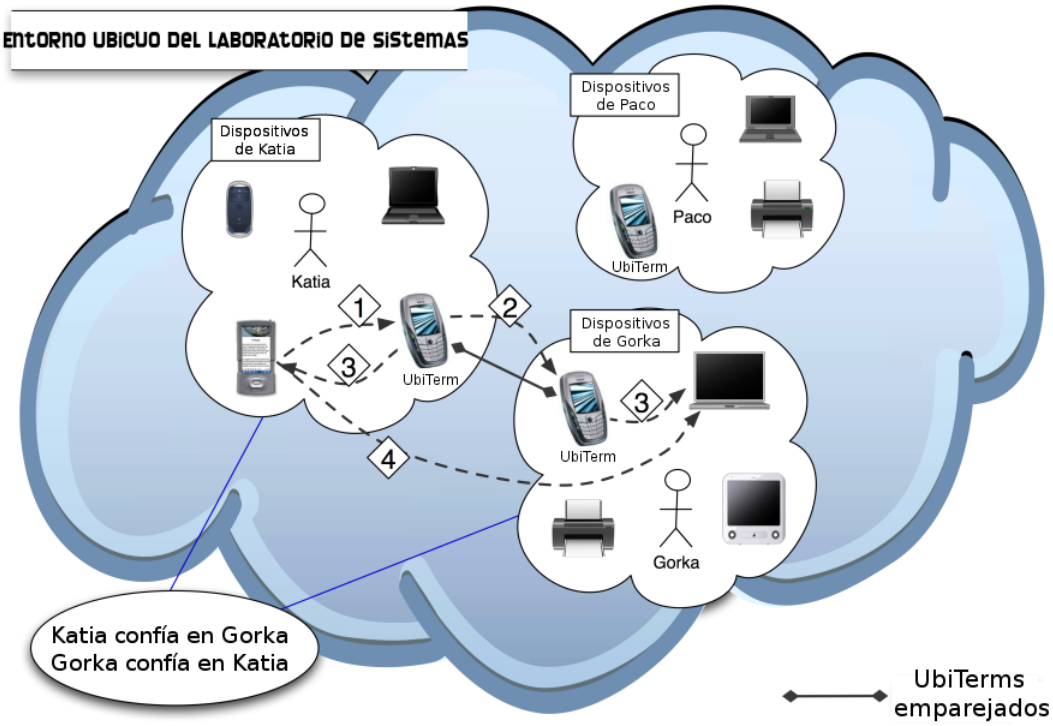

Figura 3.3: *La ilustración representa los pasos que se siguen para la comunicación entre la PDA de Katia y el ordenador portátil de Gorka.*

considere oportuno.

Después de emparejar los UbiTerms, Gorka y Katia se asignan un rol respectivamente, cada uno en su propio UbiTerm. Katia confía plenamente en Gorka, por lo que le asigna un rol de *usuario de todos dispositivos de entrada/salida*. Por otra parte, Gorka es más cauteloso administrando sus recursos y asigna a Katia un rol de *usuario de dispositivos de sólo salida*.

Unos días después, Katia encuentra a Gorka en su lugar de trabajo. Katia lleva consigo un PDA, y Gorka un ordenador portátil. Además, ambos llevan encima sus respectivos UbiTerms. En ese momento, Katia necesita que Gorka escuche un fichero de audio comprimido que tiene almacenado en su PDA. Para reproducir el fichero, el PDA de Katia utiliza los altavoces del ordenador

portátil de Gorka, ya que tienen más calidad que su altavoz. Los pasos serían los siguientes:

1. El PDA de Katia se pone en contacto con su UbiTerm. El UbiTerm de Katia va a actuar como un servidor de autenticación para que la PDA de Katia pueda contactar con el ordenador portátil de Gorka.

2. El UbiTerm de Katia se pone en contacto con el UbiTerm de Gorka. Como los dos UbiTerms están emparejados, pueden usar un secreto compartido negociado con anterioridad para establecer una conexión segura en este momento y autenticarse mutuamente. En ese momento, el UbiTerm de Katia pide que se genere una autorización para que la PDA de Katia pueda acceder al sistema de audio del portátil de Gorka. Como Katia tiene asignado un rol de *usuario de dispositivos de sólo salida* en el UbiTerm de Gorka, tiene el acceso garantizado para el sistema de salida de audio del portátil. El UbiTerm de Gorka genera la autorización y se la envía al UbiTerm de Katia.

3. El UbiTerm de Katia envía la autorización a la PDA. Concurrentemente, el UbiTerm de Gorka envía otra autorización al portátil (cifrado con un secreto compartido entre el portátil y la PDA). Esta autorización da acceso a Katia al sistema de audio del portátil.

4. La PDA de Katia establece contacto con el portátil de Gorka usando la autorización que se ha generado antes especialmente para ello. El portátil de Gorka puede validar la autorización y ambos son capaces de establecer una comunicación segura. A partir de este momento, la aplicación de reproducción de audio que se ejecuta en el PDA de Katia usa el sistema de salida de audio del portátil.

Imaginemos que un rato después, Katia sigue usando el sistema de audio del portátil de Gorka desde su PDA, pero Gorka se ausenta del sistema (con su UbiTerm). En ese caso no hay problema, ya que la PDA de Katia todavía posee

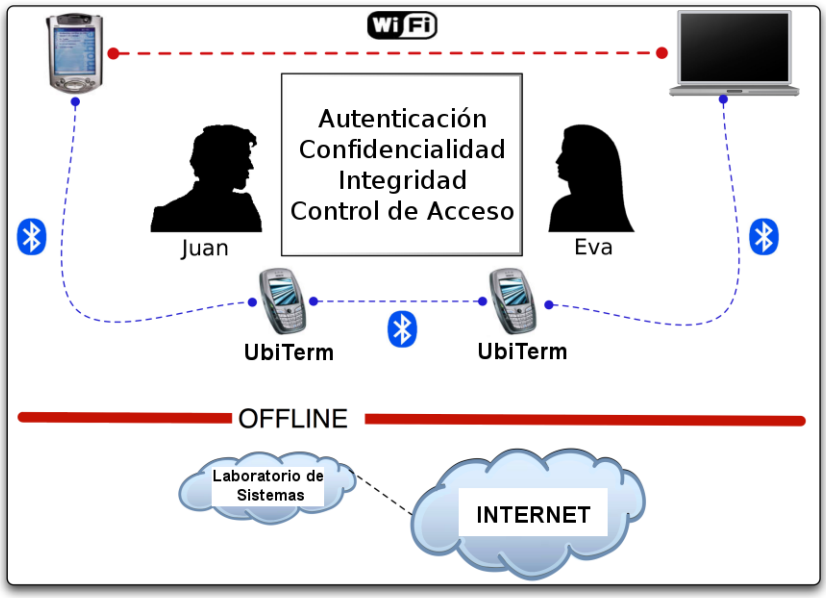

Figura 3.4: *Ejemplo de comunicación entre usuarios en lugares aislados.*

la autorización, y no es necesario que el UbiTerm de Gorka esté presente en el sistema para que Katia siga usando el dispositivo de audio.

3.4.3. Escenario 3: independencia de servicios centralizados

Supongamos ahora el escenario ilustrado en la figura 3.4. Juan y Eva son dos usuarios del sistema ubicuo que confían entre sí y tienen emparejados sus UbiTerms.

Un día se encuentran en un lugar aislado de la red del departamento, por ejemplo en el aparcamiento subterráneo del edificio. Juan lleva encima su UbiTerm y un Pocket PC en el que tiene almacenado un vídeo de alta resolución. Eva lleva su UbiTerm y un ordenador portátil con pantalla de 17 pulgadas. Tanto el Pocket PC como el portátil pueden comunicarse mediante Bluetooth [69] y Wi-Fi [195] (LAN inalámbrica IEEE 802.11 [1; 127]).

Juan quiere enseñar el vídeo a Eva en ese momento, pero su Pocket PC

no tiene suficiente resolución como para visualizarlo correctamente. Para ello, utiliza la pantalla de alta resolución del portátil de Eva. El proceso es similar al del escenario anterior: el Pocket PC contacta con el UbiTerm de Juan para establecer una comunicación segura con el portátil de Eva. Los UbiTerms se comunican entre sí usando, por ejemplo, la tecnología Bluetooth. La conexión entre el portátil y el Pocket PC podría establecerse por tecnología Wi-Fi.

En este escenario, aun estando en un lugar en el que no hay forma de contactar con ningún servicio centralizado, los usuarios son capaces de compartir sus recursos de una forma sencilla y sin obstrucción. Una arquitectura de seguridad dependiente de servicios centralizados no podría funcionar en un escenario como este. Sin embargo, SHAD puede hacer frente a esta situación por su diseño centrado en el humano.

3.5. Hipótesis, costes y compromisos

A continuación se enumerarán las hipótesis y suposiciones que se aceptan en el diseño de SHAD. También se exponen los costes y los compromisos que tiene que afrontar el usuario para utilizar el sistema.

3.5.1. Hipótesis

1. El entorno ubicuo en el que los usuarios interactúan es un entorno realista y posible en la actualidad. El usuario posee varios dispositivos, tal vez decenas, pero no centenas. Estos dispositivos poseen capacidad de procesamiento y de comunicación (inalámbrica y/o cableada).

   Esta hipótesis es importante, ya que si el número de dispositivos pertenecientes al usuario fuera un orden de magnitud superior, el esquema propuesto podría no ser el más apropiado. En ese caso se tendría que diseñar una arquitectura jerárquica que permitiese gestionar un gran número de dispositivos. Por supuesto, la eliminación de esta hipótesis da lugar a una linea de posibles trabajos futuros.

2. Los algoritmos de cifrado disponibles son algoritmos seguros y correctos. SHAD utiliza algoritmos que se consideren estándares en la industria.

3. Los sujetos a proteger son los recursos, ya sean datos o dispositivos, que pertenecen al usuario. Los usuarios habitualmente son humanos, aunque pueden existir usuarios virtuales.

4. La red de comunicaciones es inherentemente insegura: el adversario puede espiar los datos transmitidos, eliminar mensajes de la red, generar mensajes nuevos y duplicar mensajes existentes.

5. El adversario no tiene acceso físico al hardware del usuario.

6. Un usuario está dispuesto a llevar en todo momento una CPU con él, llamada UbiTerm. Esta CPU será comúnmente un ordenador de mano, un teléfono móvil *Smart Phone* o un dispositivo de similares características.

   Como ya se ha explicado anteriormente, el UbiTerm es el dispositivo que proporciona a SHAD un diseño centrado en el humano. Si un usuario usase un dispositivo fijo (p.e. un PC) como UbiTerm, entonces se perdería el diseño centrado en el humano. Así mismo, tendría varias consecuencias negativas para la arquitectura. En primer lugar, se perdería la propiedad de independencia de servicios centralizados porque el usuario dependería de la conectividad con el PC que ejecuta su UbiTerm. De esta forma, el usuario sólo se podría autenticar en localizaciones que tuvieran conectividad con su UbiTerm. Además, el usuario no podría confirmar operaciones en cualquier momento, sino que sólo lo podría hacer cuando estuviera trabajando en el PC que ejecuta su UbiTerm. De la misma forma, no podría recibir avisos o notificaciones urgentes sobre la seguridad en sus pertenencias a no ser que estuviera trabajando localmente en dicho PC.

7. Los humanos en los que el usuario confía actúan de buena fe. El uso de cualquier dispositivo cuyo propietario actúa maliciosamente es intrínsecamente inseguro.

3.5.2. Costes y compromisos

El usuario del sistema tiene que aceptar los siguientes puntos:

1. El protocolo de seguridad que implementa SHAD añade latencia a las comunicaciones e incrementa la carga de CPU de los dispositivos. Como consecuencia se reduce el tiempo de autonomía de las baterías de los dispositivos móviles, especialmente el del dispositivo que actúa como UbiTerm.

2. SHAD introduce un nuevo riesgo sobre los sujetos: el UbiTerm puede extraviarse. Un adversario que se haga con el UbiTerm puede hacerse pasar por su dueño en el sistema ubicuo y adoptar su identidad. Compromisos:

   - El usuario considera su UbiTerm como un objeto de valor equivalente a cualquiera de los objetos físicos de los que depende en la vida real.
   - Si un UbiTerm queda comprometido en algún momento, su dueño actúa en consecuencia activando los mecanismos necesarios para revocar sus privilegios ante los recursos de los demás.

3. El usuario tiene que encargarse de emparejar su propio UbiTerm con el de los usuarios en los que confía. Esta acción se realiza como un acto social más de la vida real. Puede hacerse fácilmente y de una forma automatizada. Compromisos:

   - Un usuario únicamente empareja su UbiTerm con individuos en los que confía, siendo consciente de que estos pueden acceder a sus recursos de una forma controlada.
   - El usuario asigna roles a los usuarios emparejados para controlar el acceso a sus recursos. Los roles regulan el nivel de confianza en los usuarios emparejados.

4. El usuario tiene que autenticarse ante su UbiTerm, tal y como lo realizaría en un sistema operativo corriente con control de acceso a los recursos. Esta

autenticación sirve para activar el UbiTerm y proteger todos los secretos del usuario. Compromisos:

- Normalmente esta autenticación se realiza a través de una contraseña. En ese caso, el usuario elige una contraseña segura para su UbiTerm.

5. El usuario tiene que realizar tareas de administración sencillas en su UbiTerm para ajustar el nivel de obstrucción y seguridad. Compromisos:

   - El usuario es consciente de la relación entre obstrucción y seguridad, y configura los mecanismos que ofrecemos en SHAD para ajustar el equilibrio entre ambas propiedades.

3.6. Metodología

Esta sección describe la planificación que se ha seguido para el diseño de la arquitectura de seguridad SHAD y para el desarrollo de los distintos prototipos que sirven como prueba de concepto.

Durante todo el proceso se ha adoptado un ciclo de desarrollo en espiral [32], en el que los prototipos incrementan su funcionalidad hasta llegar a los objetivos planteados. De esta forma, en cada iteración del ciclo de desarrollo se refina el diseño de la arquitectura y de los protocolos.

El plan general de trabajo se resume en el cuadro 3.1. El cuadro muestra las tareas realizadas para el desarrollo de SHAD englobadas en seis fases. A continuación explicaremos brevemente cada una de estas fases y sus tareas.

En un principio se estudian las propuestas de otros autores para solucionar el problema de la seguridad, tanto en sistemas distribuidos clásicos como en sistemas ubicuos. El estado del arte se actualiza constantemente durante todo el proceso de diseño e implementación de SHAD.

Una vez que se comprueba que ninguna de las arquitecturas propuestas se ajustan a las necesidades de nuestro sistema ubicuo, se comienza a diseñar la arquitectura de seguridad.

En esta primera fase se establecen establecen los principios básicos [175] para la creación de una arquitectura de seguridad. Además se estudian las distintas herramientas criptográficas disponibles hoy día. También se estudian los problemas patentes en otras arquitecturas con el fin de evitarlos en SHAD. Por último, se establecen los requisitos que debe cumplir el diseño de nuestra arquitectura.

En una segunda fase se diseñan los protocolos que usa la arquitectura para sus servicios. En esta fase también se eligen las herramientas criptográficas para los distintos protocolos.

En la tercera fase se estudia en profundidad la arquitectura de seguridad del sistema operativo Plan 9 y se diseña e implementa un prototipo que incorpora algunas de las ideas de SHAD a este sistema. En ese prototipo se analiza la posibilidad de usar distintos dominios de seguridad para cada usuario del sistema, de tal forma que cada uno de ellos posea un servidor de autenticación propio. También se experimenta con el proceso de arranque del sistema operativo con el fin de proporcionar una entrada única al sistema (*Single Sign-On*) para el espacio inteligente.

En una cuarta fase se estudia la integración de del prototipo de SSO con el sistema operativo Plan B. Después se incrementa la arquitectura con los mecanismos de compartición de terminales [176] y de compartición de dispositivos. Para ello se implementan los mecanismos para utilizar los servicios del espacio inteligente, la interfaz gráfica integrada con Omero[18], los protocolos de cesión y las librerías de control de acceso basadas en roles.

En la quinta fase se evalúa la arquitectura mediante el uso del entorno ubicuo. Para ello, se usa el prototipo final en nuestro propio espacio inteligente para comprobar que la arquitectura cubre las necesidades expuestas en el capítulo 1.

La sexta y última fase de la planificación consiste en la creación de la documentación relacionada con la arquitectura de seguridad y los distintos prototipos implementados. Después se elabora el presente documento y la documentación relacionada con su presentación. Por último, se revisan todos

los documentos y se procede a su presentación.

Fase	***Etapas***
Fase I: Identificación de las tareas	1.- Plan de requisitos y adquisición de conocimientos
	2.- Estado del arte
	3.- Identificación y definición de tareas
Fase II: Diseño	1.- Diseño de los protocolos
	2.- Elección de algoritmos de cifrado
Fase III: Implementación de los primeros prototipos	1.- Introducción a Plan 9 y primeras modificaciones
	2.- Experimentación con dominios por usuario
	3.- Modificación del arranque de Plan 9
	4.- Primer prototipo de *Single Sign-On* real para Plan 9
Fase IV: Implementación	1.- Adaptación del prototipo de SSO a Plan B
	2.- Prototipo para la compartición de terminales para Plan B
	3.- Prototipo para la compartición de dispositivos para Plan B
	4.- Creación de una interfaz gráfica para SHAD
Fase V: Evaluación y pruebas	1.- Uso diario del prototipo
	2.- Evaluación de requisitos de uso
	3.- Extracción de medidas
Fase VI: Elaboración de la tesis	1.- Elaboración de la documentación
	2.- Revisión
	3.- Presentación

Cuadro 3.1: *Planificación seguida para el desarrollo de SHAD*

Capítulo 4

Entrada única al sistema ubicuo

4.1. Introducción

Como consecuencia de la generalización del ordenador personal y la aparición de la gran cantidad de dispositivos móviles disponibles hoy en día, el número de máquinas que utiliza al mismo tiempo un usuario ha crecido sustancialmente. Estos dispositivos son básicamente ordenadores, que aunque poseen distintas cualidades físicas y funcionales, ejecutan un sistema operativo convencional y están sometidos a las mismas necesidades de configuración y administración que un ordenador común. Entre dichas necesidades se encuentra el proceso de identificación del usuario y su posterior autenticación.

El proceso de autenticación se realiza normalmente mediante la introducción de contraseñas por teclado, aunque se han propuesto métodos que proporcionan menos obstrucción al usuario, como la autenticación mediante tarjetas Smartcard[148], Ibuttons[76], USB Tokens[157] o dispositivos biométricos, tales como lectores de huellas digitales. Esos métodos, aun siendo menos incómodos para el usuario que la introducción de contraseñas, siguen aportando obstrucción. Por otra parte, dependen de dispositivos lectores específicos que normalmente no están integrados en los propios dispositivos. Esto supone un problema a la hora de usar dispositivos móviles y máquinas de propósito general.

Los sistemas tradicionales que proporcionan entrada única al sistema (*Single Sign-On*) se han diseñado para entornos cliente/servidor en los que los usuarios trabajan con un único terminal desde el que acceden a todos los servicios del entorno distribuido. De hecho, estos sistemas son sistemas de SSO *por máquina* en los que sus agentes de SSO no cooperan. Los agentes de SSO en estos sistemas suelen ser programas cliente que acceden a un servidor centralizado de claves común para todos los usuarios, y el usuario debe autenticarse al menos una vez para cada agente.

Sin embargo, los usuarios de un entorno de computación como el presentado en capítulos anteriores no trabajan en un sólo ordenador, sino que utilizan e interaccionan físicamente con un grupo de ordenadores interconectados que conforman el espacio inteligente. De esta forma, el usuario necesita autenticarse

ante cada uno de los ordenadores que integran el espacio con el que está interaccionando. Esta situación aparece como consecuencia de usar un sistema SSO *por máquina* en un entorno ubicuo.

La autenticación explícita por parte del usuario en cada una de las máquinas rompe la ilusión de que *el entorno ubicuo es un único sistema y no un grupo de sistemas interconectados*. Por tanto, los sistemas de SSO convencionales no son apropiados para espacios inteligentes y entornos ubicuos, simplemente porque no proporcionan una única entrada (o autenticación) al sistema constituido por un grupo de máquinas interconectadas.

Teniendo en cuenta que el número de nuevos dispositivos en el sistema tiende a crecer debido a la progresiva miniaturización y el abaratamiento de los dispositivos móviles según modeló Moore[111], el problema de la autenticación sin obstrucción adquiere un papel importante en la computación ubicua.

Con SHAD proponemos una solución para este problema basada en agentes que actúan como servidores personales de claves para el usuario. Estos agentes tienen como objetivo proporcionar al usuario un acceso simple y sin obstrucción al entorno de computación ubicuo, haciendo el uso de varias máquinas a la vez cómodo, seguro y parcialmente transparente en lo que respecta a la autenticación dependiendo de las necesidades y preferencias del usuario. El sistema está ideado para los dispositivos convencionales, tanto móviles como de escritorio, que puede poseer un usuario en un espacio inteligente actual. Sin embargo, el sistema no contempla dispositivos tales como redes de sensores o microcontroladores.

Mediante los agentes SHAD, los dispositivos que pertenecen a un mismo usuario pueden compartir sus secretos (claves, contraseñas, etc.), propiciando un acceso simple al entorno ubicuo y conservando la ilusión de que se está trabajando en un único sistema. A continuación, se presentarán las ideas básicas de la arquitectura de SHAD para ofrecer una entrada única al sistema.

4.2. El agente SHAD de SSO

Siguiendo las ideas que propusimos en capítulos anteriores, cada usuario tiene un agente personal que suministra los secretos necesarios para que las aplicaciones que ejecutan en otras máquinas (que también pertenecen al usuario) puedan acceder tanto a los servicios del entorno ubicuo como a servicios convencionales.

El agente personal que se encarga de distribuir los secretos se denomina **agente principal**. Este agente está ideado para ejecutar en un dispositivo móvil que el usuario siempre lleva consigo, el **UbiTerm** (ver capítulo 3). No obstante, el agente SHAD principal puede ejecutar en cualquier máquina que el usuario desee, siempre y cuando la máquina le pertenezca. En cada máquina perteneciente al usuario se ejecuta un agente SHAD común, que solicita al agente principal los secretos que necesiten las aplicaciones locales a su máquina. Los agentes convencionales proporcionan autenticación a sus aplicaciones locales de dos formas:

- Ejecutando la lógica de autenticación necesaria, de tal forma que toda la autenticación queda fuera de la aplicación.
- Proporcionando el secreto necesario a la aplicación, que se encarga de ejecutar su propia lógica de autenticación.

En ese aspecto, los agentes SHAD siguen la misma aproximación que otros trabajos presentados en la literatura [41]. SHAD difiere de esos sistemas en los mecanismos que provee para que los agentes cooperen con el fin de extender el SSO entre todas las máquinas del usuario. Como ya hemos explicado anteriormente, esos sistemas no ofrecen SSO real en un espacio inteligente, SHAD sí.

Cuando el agente principal está disponible, los agentes comunes obtienen los secretos de él. En caso de fallo del agente principal, los agentes comunes pueden solicitar e intercambiar secretos entre ellos mediante un protocolo *Peer-to-Peer*. A los agentes comunes no les está permitido compartir todos los secretos

que tienen almacenados. Sólo pueden compartir los secretos que el usuario ha especificado como compatibles en la configuración. De esta forma, el usuario puede estar seguro de que ciertos secretos nunca se compartirán entre agente comunes.

El agente SHAD ofrece los mecanismos para que el usuario pueda definir sus propias políticas para el control de acceso a sus secretos. El usuario es capaz de restringir la disponibilidad de sus secretos en base a ciertos atributos que puede asignar a cada secreto.

Por último, el agente SHAD saca partido de una infraestructura de contexto de un espacio inteligente real. En particular, de información de localización, así como de otros servicios del mismo, como por ejemplo el servicio de mensajes de voz.

4.3. Arquitectura SHAD de SSO

La arquitectura general del sistema se presenta en la figura 4.1. Como se puede observar, la arquitectura se compone de entidades principales:

- **Las aplicaciones.** Las aplicaciones son programas de usuario que acceden a los servicios disponibles en el entorno ubicuo y fuera del mismo. Comúnmente, estas aplicaciones necesitan autenticarse de alguna forma (mediante contraseñas, respuestas de retos, obtención de credenciales, etc.) ante los servidores del sistema. Para llevar a cabo dicha autenticación, los programas necesitan conocer secretos que normalmente les proporciona el usuario mediante un dispositivo de entrada. Como ya se ha comentado anteriormente, las aplicaciones pueden elegir entre intentar obtener los secretos del agente SHAD local y gestionar ellas mismas la autenticación, o delegar la autenticación en el agente SHAD local si este es capaz de llevarla a cabo[1]. En el caso de que el agente SHAD no responda o no

[1]El agente SHAD puede ejecutar la lógica de autenticación de los protocolos para los que tenga su módulo implementado.

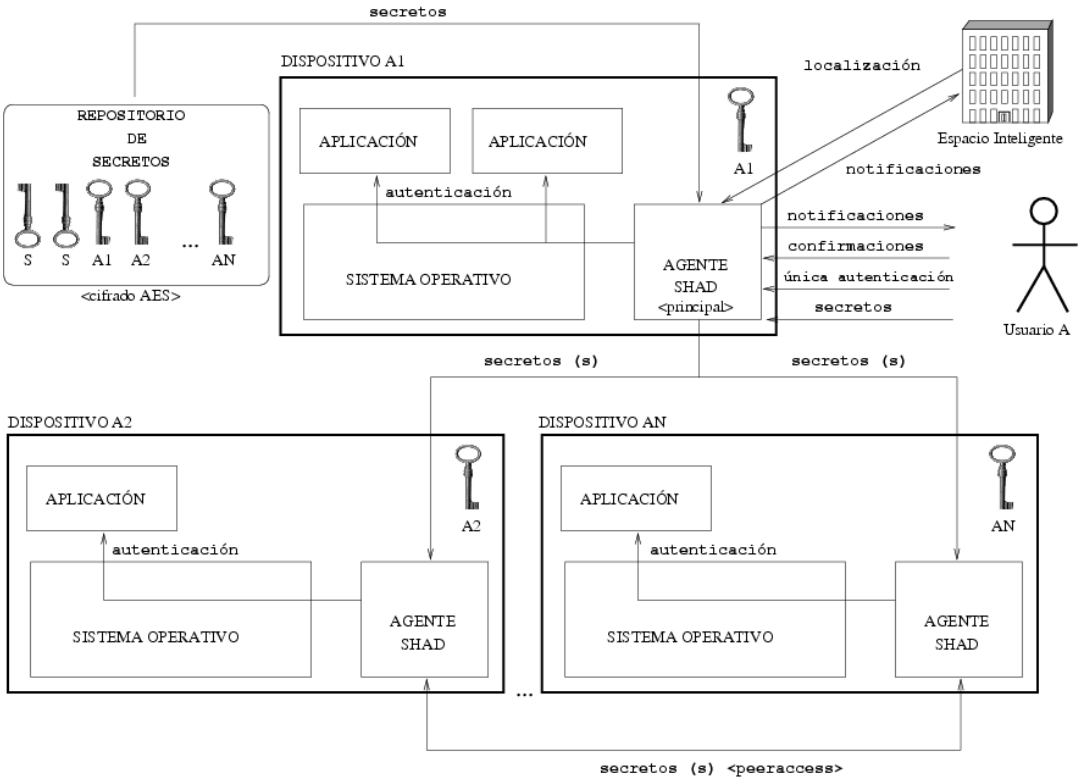

Figura 4.1: *Arquitectura global del sistema de entrada única al sistema (Single Sign-On) basada en SHAD*

pueda llevar a cabo la autenticación, las aplicaciones reclaman el secreto al usuario de la forma tradicional.

- **El sistema operativo.** El sistema operativo proporciona a las aplicaciones el acceso al agente SHAD local.

- **El usuario.** El usuario del sistema es un humano que interacciona físicamente con el entorno ubicuo. El humano es el encargado de administrar sus propios dispositivos, como ya se ha explicado en el capítulo anterior. Además, debe realizar una primera y única autenticación explicita para entrar en el sistema, que servirá para descifrar el repositorio de secretos. También puede añadir nuevos secretos (que no están almacenados en el repositorio) a sus agentes y proporcionar confirmaciones que pueden

requerirse para permitir la delegación de ciertos secretos. El usuario puede cancelar y confirmar operaciones interactuando con el agente SHAD principal.

- **El espacio inteligente** o entorno ubicuo, que puede ofrecer servicios tales como localización física de personas y objetos. El sistema de SSO puede sacar partido de la información de localización, por ejemplo para pedir confirmación al usuario en ciertas situaciones. El agente SHAD principal también puede comunicar notificaciones al usuario a través del espacio inteligente. En todo caso, la arquitectura no puede depender de los servicios suministrados por el espacio inteligente, ya que debe estar operativo en particiones de red sin conectividad con dicho espacio. Por lo tanto, el agente de SHAD debe ser capaz de operar sin los servicios que le ofrece el entorno ubicuo. El agente principal accede a los servicios usando mecanismos estándar que quedan fuera del ámbito de la arquitectura (por ejemplo a través de un sistema de ficheros). Por consiguiente, la seguridad relacionada con esos servicios no se tratará en este capítulo.

- **Los agentes SHAD.** Los agentes SHAD son los encargados de manejar la autenticación del usuario, normalmente sin la necesidad de interactuar con este (dependiendo de su configuración). Como se detallará más adelante, el usuario es capaz de configurar el comportamiento de los agentes según sus preferencias y su valoración del compromiso entre obstrucción y nivel de seguridad. En realidad, hay un único tipo de agente SHAD, que puede actuar de dos formas distintas: como *agente principal* o como agente común.

- **Las claves de los terminales.** Cada máquina perteneciente al usuario tiene asignada una *clave de terminal*. Todas las claves de terminal se almacenan en el repositorio de secretos. La clave de terminal permite al agente común que ejecuta en la máquina establecer una canal seguro con el agente principal. Por tanto, para que una máquina del usuario pueda

ejecutar una agente SHAD, es necesario que tenga asignada una clave de terminal. Esta clave se representa en la figura 4.1 mediante una llave junto con el nombre del terminal.

- **El repositorio de secretos** del usuario es un fichero cifrado que contiene los secretos de usuario en un formato legible para los humanos y las claves de los sus terminales. En la figura 4.1 los secretos del usuario se representan mediante una llave invertida y la letra S. Los secretos del usuario se almacenan junto con sus atributos, que restringen su uso y su distribución. Este fichero se encuentra cifrado con un algoritmo fuerte, y su contenido nunca se almacena en un soporte no volátil en texto claro. El repositorio se puede almacenar en un soporte externo (p.e. de una tarjeta de memoria SD Card) o en un servidor centralizado. Nótese que si se obtiene de un servidor centralizado, el usuario deberá tener interconexión con esa partición de la red en el momento de obtenerlo, pudiendo perder la conectividad después. En el caso de usar un soporte externo, como una tarjeta SD card, el repositorio se puede obtener sin tener conectividad con el resto del sistema.

Un agente de SHAD que actúa como *agente principal* ofrece los secretos del usuario a los otros agentes SHAD, dependiendo de ciertas restricciones que tratamos más adelante. Cualquier agente puede asumir el rol de agente principal en tiempo de arranque, dependiendo de los siguientes factores:

- El descubrimiento de otro agente SHAD actuando como agente principal. Cada máquina ejecuta su agente al arrancar, y este inicia un proceso de descubrimiento del agente principal. Si el proceso de descubrimiento fracasa, el agente intenta asumir el papel de agente principal.

  Puede ocurrir que, tras una partición de la red y su posterior unificación, haya dos agentes ejecutando como agente principal para un mismo usuario. En ese caso, los dos agentes principales siguen ofreciendo servicio a los agentes comunes que iniciaron la sesión con ellos. Los agentes que arrancan

a partir de la unificación establecen su sesión con uno de ellos (el más rápido en contestarles). Es tarea del usuario controlar su agente principal y reiniciarlo cuando proceda (por ejemplo, en este caso debería reiniciar uno de los dos agentes principales).

- La capacidad de obtener el repositorio de secretos del usuario. Si el repositorio de secretos se encuentra en un sistema remoto, el agente debe tener conectividad con dicho sistema. En caso de que el repositorio se encuentre en una tarjeta insertada en el dispositivo donde ejecuta el agente, este debe ser capaz de obtenerlo y copiar su contenido en la memoria principal.

- La capacidad de descifrar el repositorio de secretos del usuario. Para ello, el usuario debe realizar la **única autenticación explícita**. Si el usuario no se autentica explícitamente ante el agente SHAD, este no puede proclamarse como agente principal. Dicha autenticación explícita puede consistir en la introducción de una contraseña, o la utilización de alguna tecnología biométrica o de algún otro dispositivo de autenticación. A través de dicha autenticación, el agente es capaz de descifrar el contenido del repositorio de secretos del usuario.

Aunque cualquier dispositivo del usuario es capaz de ejecutar el agente principal, este está ideado para ejecutar en un dispositivo móvil que siempre se encuentra con el usuario. De esa forma, el usuario es capaz de atender a mensajes informativos relativos a la delegación de sus secretos y confirmar la delegación de determinados secretos. El repositorio de secretos está ideado para obtenerse de una tarjeta de memoria directamente desde el dispositivo móvil, y de esa forma no depender de la conectividad para ofrecer SSO en una partición separada del resto del sistema. No obstante, el usuario es libre de ejecutar su agente principal en cualquiera de sus máquinas, y de obtener su repositorio de secretos desde un servidor centralizado si así lo desea.

Una vez que un agente SHAD ha adoptado el papel de agente principal, entonces puede suministrar secretos a los agentes SHAD que ejecutan en otros dispositivos pertenecientes al usuario, que actuarán como clientes. Los agentes cliente solicitan al agente principal los secretos requeridos por las aplicaciones que ejecutan en su dispositivo. La delegación de los secretos del usuario se lleva a cabo dependiendo de restricciones que tienen asociados en el repositorio. En ningún caso se delegarán secretos a agentes SHAD que no puedan autenticarse ante el agente principal. Tampoco se podrán delegar secretos a agentes que no estén ejecutando en nombre del mismo usuario que el agente principal.

En caso de fallo del agente principal, los agentes clientes pueden intentar contactar entre ellos para poder obtener secretos. Si un agente SHAD común intenta contactar con su agente principal para obtener un secreto y no recibe respuesta, entonces inicia un protocolo para intentar obtener el secreto de otros agentes SHAD de su mismo tipo. Si finalmente no obtiene respuesta de ningún agente, entonces intenta obtenerlo directamente del usuario. En ese caso, se pregunta al usuario mostrando un mensaje a través de la interfaz gráfica o de la consola del sistema local a la aplicación que solicita el secreto. De esta forma, los agentes SHAD intentan cooperar e intercambiar secretos para interrumpir al usuario sólo en última instancia.

4.4. Diseño del Agente SHAD de SSO

4.4.1. Modelado de flujo de datos

El diseño del prototipo se ha llevado a cabo mediante Diagramas de Flujo de Datos [44]. Los diagramas de este tipo tienen dos funciones principales: (i) describir la forma en la que los datos se transforman según pasan por el sistema, y (ii) definir las funciones que transforman o manejan los datos [143]. Mediante estos diagramas, se pretende modelar la funcionalidad de los procesos y definir

las entidades internas de los agentes SHAD[2].

La figura 4.1 muestra las entidades externas con las que los agentes de SHAD intercambian información. Por lo tanto esa figura es equivalente a un diagrama de contexto (diagrama de flujo de datos de nivel cero).

Las entidades externas a un agente SHAD ya han sido definidas en la sección anterior.

En los diagramas de nivel uno presentados en las figuras 4.2 y 4.3 se describe con más detalle el flujo de los datos entre las entidades internas de los agentes. Se han realizado dos diagramas distintos para describir el comportamiento del agente de SHAD, debido a que aunque los agentes son exactamente el mismo programa, pueden adoptar dos funcionalidades distintas dependiendo de las circunstancias que se den en tiempo de arranque, como ya se dijo.

La figura 4.2 muestra el diagrama de flujo de datos de primer nivel de un agente SHAD común. Por claridad, la figura omite ciertas relaciones de control explicadas en la sección que sigue. Las entidades internas a un agente SHAD común son:

- **Anillo de secretos (RI).** Es un almacén de datos que contiene los secretos del usuario. También almacena las claves de las máquinas pertenecientes al usuario. La memoria en la que se almacena el anillo de secretos debe estar lo más protegida posible. Si el sistema operativo lo permite, la memoria principal del agente SHAD no se debe poder depurar ni paginar a área de intercambio.

- **Módulo de descubrimiento (DM).** Se encarga de descubrir al agente principal en tiempo de arranque. Cada dispositivo capaz de arrancar agentes de SHAD tiene que tener asignada una clave privada. Dicha clave privada ha debido asignarse en tiempo de configuración. Además, esa clave debe incluirse en el repositorio de secretos. De esta forma, el agente principal puede autenticar mensajes de cualquier dispositivo configurado

[2]Sin embargo, en esta sección no se pretende realizar un análisis o modelado exhaustivo del sistema.

para usar SHAD. La clave de la máquina se leerá de un componente hardware local. Si el módulo no es capaz de descubrir al agente principal, entonces intentará obtener el repositorio de secretos para convertirse él mismo en agente principal. Si logra descubrirlo, intentará establecer una sesión con él para poder reclamar secretos cuando sea necesario.

- **Módulo de secretos (SM).** Es la entidad encargada de proporcionar autenticación a las aplicaciones. El usuario debe ser capaz de añadir nuevos secretos en el *anillo de secretos* a través de esta entidad. El módulo necesita usar un secreto para proporcionar autenticación a una aplicación. En primera instancia, intenta obtener el secreto del anillo. Si no es capaz de obtenerlo, insta al *módulo recolector* a obtenerlo.

- **Módulo recolector (RM).** Es el encargado de conseguir los secretos necesarios del agente principal. Para ello, se pone en contacto con el agente principal e inicia el protocolo de obtención de secretos. En caso de fallo del agente principal, pide al *módulo Peer-to-Peer* que intente obtener el secreto.

- **Módulo Peer-to-Peer (P2PM).** Este módulo se encarga de conseguir secretos de otros agentes SHAD comunes. El módulo P2P se pondrá en contacto con el módulo P2P de los otros agentes SHAD con este fin. El módulo P2P sólo podrá ofrecer los secretos marcados especialmente para ello mediante un atributo especial.

En la figura 4.3 se pueden identificar las siguientes entidades internas dentro del agente principal:

- **Anillo de secretos (RI).** Cumple la misma función que en un agente SHAD común.

- **Módulo de secretos (SM).** Esta entidad cumple las mismas funciones que en un agente común, pero en el agente principal se ocupa además de

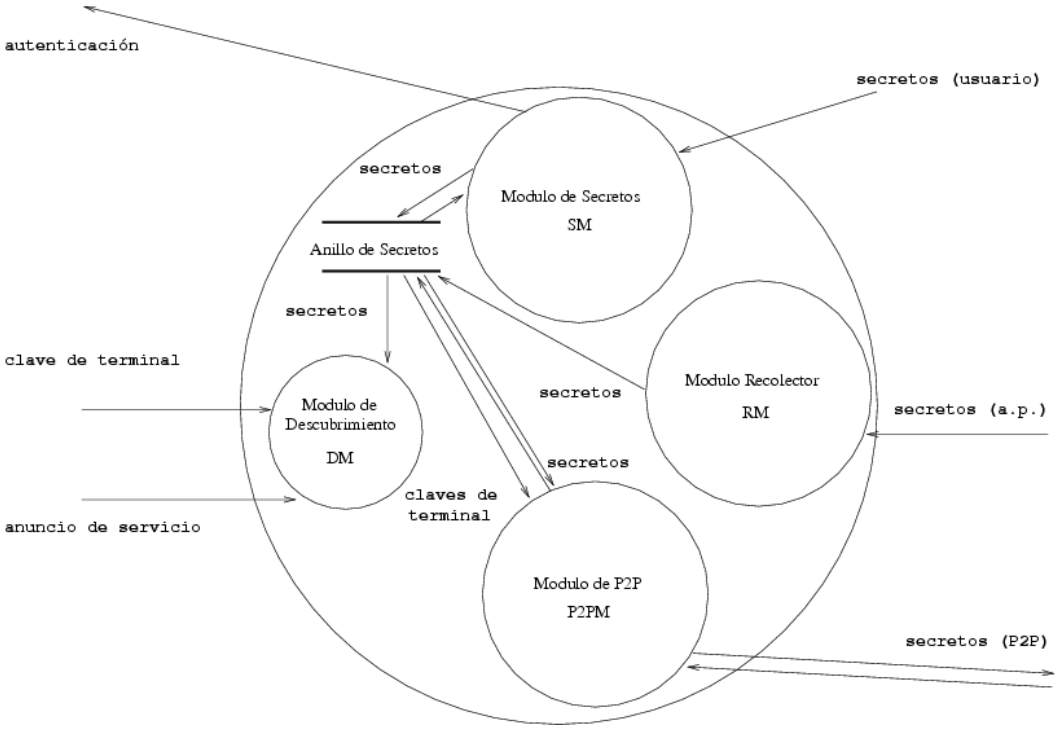

Figura 4.2: *Diagrama de flujo de datos de nivel uno correspondiente al agente SHAD de SSO común.*

obtener el repositorio de secretos y de descifrarlo mediante la información suministrada por la única autenticación explícita realizada por el usuario. A continuación, almacena su contenido en el *anillo de secretos*.

- **Módulo proveedor (PM).** Esta entidad es la encargada de suministrar secretos a los agentes SHAD comunes, dependiendo de los atributos ligados a estos, la información de localización que obtiene del espacio inteligente, y la confirmación del usuario en los casos que sea necesario.

- **Módulo de descubrimiento (DM).** Cumple la misma función que en un agente SHAD común.

- **Módulo de anuncios (AM).** Es la entidad encargada de responder a los descubrimientos de los agentes del usuario que arrancan, y de establecer la sesión con ellos.

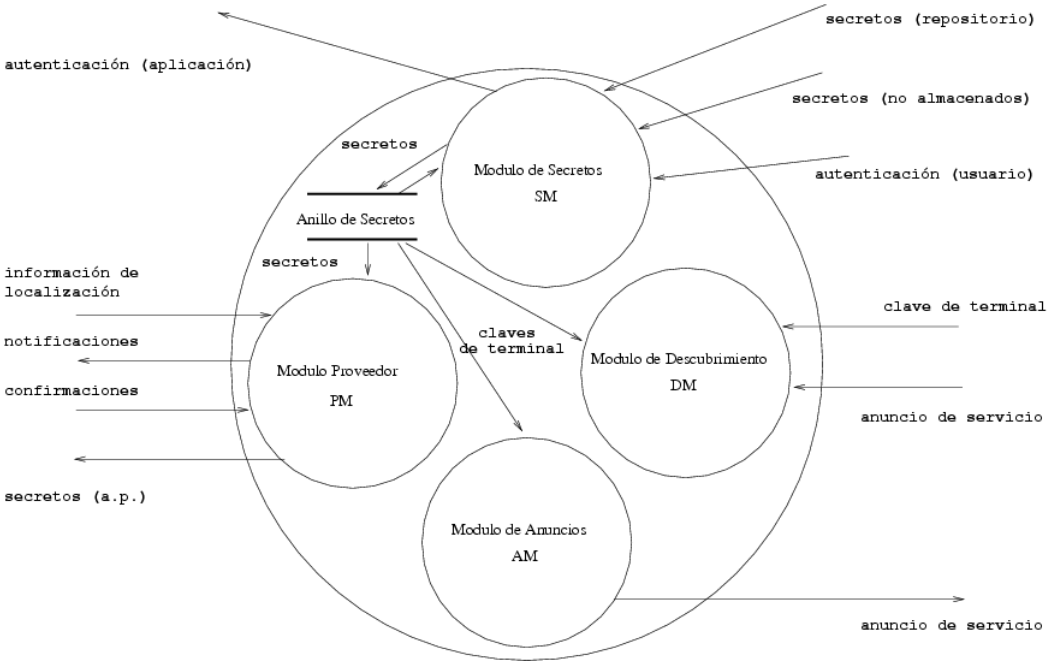

Figura 4.3: *Diagrama de flujo de datos de nivel uno para el agente SHAD de SSO principal.*

Varios de los flujos de datos mostrados en los diagramas de flujo se realizan entre entidades internas de distintos agentes:

- El descubrimiento del agente principal y establecimiento de sesión por parte del *módulo de descubrimiento*. Este flujo se establece entre el módulo de descubrimiento y el *módulo de anuncios* del agente principal, que lo procesa.
- La petición de un secreto por parte del *módulo recolector* de un agente común la atiende el *módulo proveedor* del agente principal.
- La petición de un secreto por parte del *módulo Peer-to-Peer* de un agente común la atienden los *módulos Peer-to-Peer* de los agentes comunes disponibles.

Esos flujos de datos se deben implementar con un protocolo seguro, debido a que son flujos entre distintos dispositivos interconectados mediante una

red insegura en la que puede haber terceras partes capturando, replicando o inyectando mensajes.

La propia red puede perder o duplicar mensajes. En la siguiente sección se describirá en profundidad el protocolo diseñado para la transmisión segura de los secretos entre el agente servidor y los agentes clientes.

Pero antes puede resultar ilustrativo considerar un ejemplo para ver cómo interactúan los elementos que acabamos de describir.

4.4.2. Escenario de ejemplo

Supongamos el siguiente escenario, ilustrado en la figura 4.4:

- **Paso 1:** Alice llega a su despacho y enciende su PDA. Cuando enciende la PDA, se inicia un agente SHAD. El *módulo de descubrimiento* del agente SHAD que ejecuta en la PDA radia mensajes para descubrir si existe un agente SHAD principal activo ejecutando a nombre de Alice. Ese mensaje se cifra con la *clave de terminal* del PDA.

- **Paso 2:** Tras varios intentos sin respuesta, el agente SHAD que ejecuta en la PDA supone que no hay ningún agente principal disponible e intenta adoptar ese rol. El *módulo de secretos* adquiere el fichero con el *repositorio de secretos* de Alice de una tarjeta de memoria insertada en la PDA.

- **Paso 3:** Para descifrar el contenido del repositorio, el *módulo de secretos* pide por pantalla una frase de acceso. Alice se autentica en el PDA mediante la frase de acceso. El *módulo de secretos* convierte la frase de acceso en una clave de criptografía simétrica, descifra el repositorio de secretos, y almacena su contenido en una estructura de datos en memoria principal: el *anillo de secretos*.

- **Pasos 4, 5 y 6:** Alice arranca las tres estaciones de trabajo que tiene en su despacho: T1, T2 y T3. En el proceso de arranque se inicia un agente SHAD en cada una de ellas. Los *módulos de descubrimiento* de esos agentes

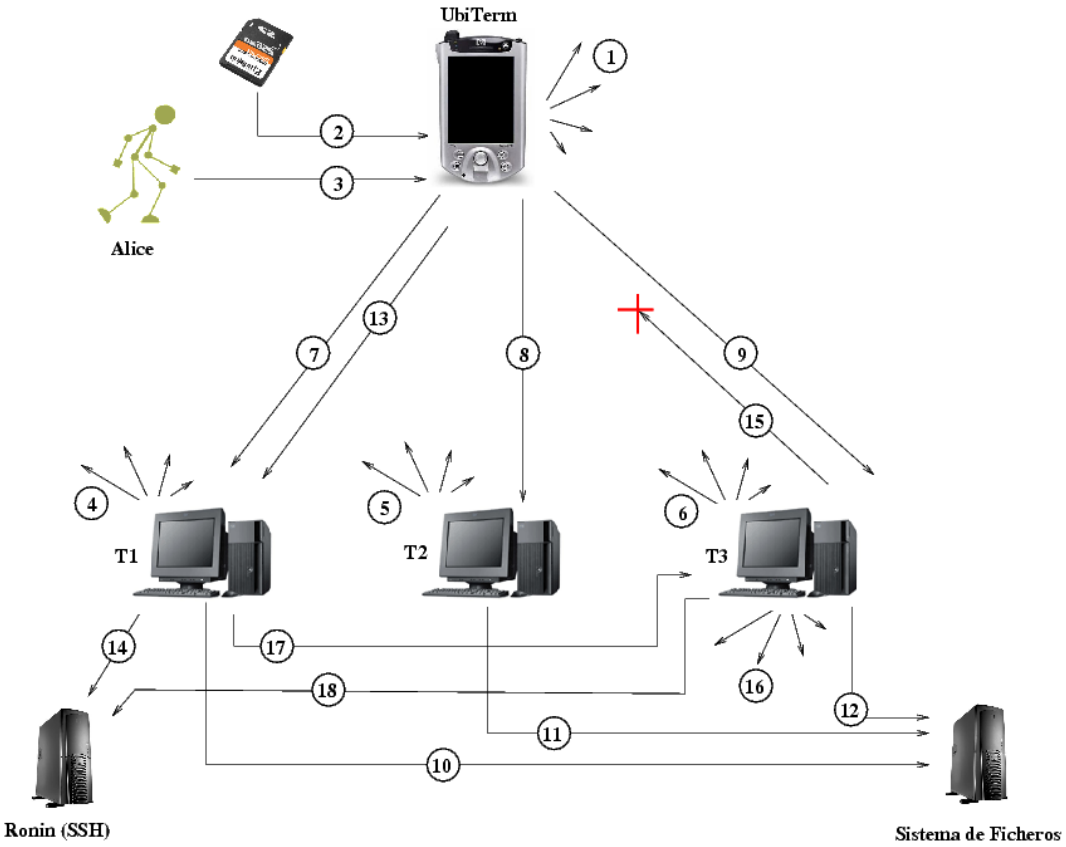

Figura 4.4: *La figura ilustra los pasos seguidos en el escenario de ejemplo.*

radian el mensaje de descubrimiento del agente principal, cifrado con sus *claves de terminal*.

- **Pasos 7, 8 y 9:** Los *módulos de descubrimiento* obtienen respuesta del *módulo de anuncios* del agente principal que está ejecutando en el PDA, que crea una sesión para cada uno de ellos.

  Los terminales continúan con la secuencia de arranque. Para montar el sistema de ficheros raíz, los terminales necesitan autenticarse como Alice ante el servidor de ficheros. El programa de arranque de cada terminal solicita autenticación al *módulo de secretos* de su agente local. El *módulo de secretos* encuentra el *anillo de secretos* vacío, y solicita el secreto al *módulo recolector*.

Los *módulos recolectores* de las estaciones solicitan el secreto al *módulo proveedor* del agente principal. El *módulo proveedor* proporciona el secreto y los módulos *recolectores* lo almacenan en su *anillo de secretos* local.

- **Pasos 10, 11 y 12:** Los *módulos de secretos* de las estaciones se autentican ante el servidor de ficheros raíz y el programa de arranque procede al montaje. Finaliza el arranque y las estaciones quedan listas para trabajar en ellas como si fuesen un único sistema.

  Alice empieza a trabajar en sus terminales. En un momento dado, una aplicación que ejecuta en el terminal T1 necesita autenticarse ante el servidor SSH de una máquina llamada Ronin.

  Para obtener el secreto, la aplicación solicita el secreto al *módulo de secretos* del agente SHAD del terminal T1. El *módulo de secretos* busca la contraseña en el *anillo de secretos* local, pero no la encuentra; el único secreto que contiene el *anillo de secretos* es la contraseña para acceder al sistema de ficheros raíz.

- **Paso 13:** El *módulo de secretos* solicita el secreto al *módulo recolector* local. El *módulo recolector* se pone en contacto con el *módulo proveedor* del agente principal. El *módulo proveedor* del agente principal busca la contraseña para acceder al servidor SSH de Ronin en su *anillo de secretos* local. Una vez encontrado, inspecciona sus atributos y los evalúa. Si no hay ningún atributo que restrinja su compartición, el *módulo proveedor* envía el secreto al *módulo recolector* del agente que ejecuta en el terminal T1.

- **Paso 14:** El *módulo recolector* lo almacena en su *anillo de secretos* local. Después, el *módulo de secretos* suministra la contraseña a la aplicación que la solicitó. La aplicación se autentica ante el servidor SSH de Ronin. La aplicación termina correctamente.

  Tiempo después, otra aplicación que ejecuta en el terminal T1 solicita al

módulo de secretos la contraseña para acceder al servidor SSH de Ronin. El *módulo de secretos* la obtiene del *anillo de secretos* local, sin necesidad de contactar con el agente principal.

- **Paso 15:** Alice se ausenta del despacho, y se lleva el PDA con ella. Recordemos que el PDA ejecuta el agente SHAD principal.

  Una aplicación que ejecuta en el terminal T3 necesita autenticarse ante el servidor SSH de Ronin. El *módulo de secretos* la intenta obtener del *anillo de secretos*, pero no puede obtenerla, ya que el único secreto que hay almacenado allí es la contraseña para acceder al sistema de ficheros raíz. El *módulo de secretos* pide al *módulo recolector* que consiga el secreto.

  El *módulo recolector* del agente de T3 intenta contactar con el *módulo proveedor* del agente principal, pero no lo consigue porque Alice se lo ha llevado fuera del edificio, donde no hay red disponible. Tras varios intentos, el *módulo recolector* desiste y comunica el error al *módulo de secretos*.

- **Paso 16:** El *módulo de secretos* solicita la contraseña al *módulo de P2P*. El *módulo de P2P* radia mensajes con la petición de la contraseña. Los *módulos de P2P* de los terminales T1 y T2 reciben la petición. Ambos inspeccionan su *anillo de secretos* en busca de la contraseña para el servidor SSH de Ronin. El módulo del agente de T1 lo encuentra, mientras que el módulo del de T2 no.

- **Paso 17:** El *módulo de P2P* del agente de T2 ignora el mensaje. El *módulo de P2P* de T1 inspecciona los atributos de la contraseña y los evalúa. Si los atributos permiten compartir el secreto a través del *módulo de P2P*, envía el secreto al *módulo de P2P* del agente de T3. El *módulo de P2P* del agente de T3 almacena el secreto en el *anillo de secretos* local.

- **Paso 18:** El *módulo de secretos* del agente de T3 pasa la contraseña a la aplicación. La aplicación se autentica ante el servidor SSH de Ronin

y ejecuta su lógica. La aplicación finaliza correctamente. Alice regresa al despacho y continúa con sus actividades.

Ninguna aplicación ha fallado en su ausencia. Alice sólo ha tenido que autenticarse explícitamente una única vez: el sistema ha ofrecido **Single Sign-On real** en un entorno distribuido en el que se usan cuatro máquinas simultáneamente.

4.5. Protocolos

A continuación se describen los protocolos que se han diseñado para el intercambio de mensajes entre agentes SHAD para el descubrimiento del agente principal, la petición de un secreto al agente principal y la petición de secretos entre agentes comunes (P2P).

Existen dos tipos de mensajes dependiendo de su estructura:

- Mensajes cortos:

CABECERA	IV	DATOS

- Mensajes largos:

CABECERA	IV'	DATOS'	IV"	DATOS"

Todos los mensajes emplean un identificador en forma de cadena de caracteres. Ese identificador se incluye en la **cabecera** del mensaje para indicar su tipo, junto con otros datos específicos del mensaje.

Los **datos** se encuentran cifrados. Para el diseño del protocolo se usa el algoritmo de cifrado de clave simétrica AES [121] en modo CBC. El **vector de inicialización (IV)** para descifrar el mensaje se incluye en el campo anterior al correspondiente a los datos. En el apéndice A se tratan los detalles relacionados con el uso de AES y y de SHA-1 [120], algoritmo utilizado para realizar los resúmenes contenidos en los datos cifrados.

Terminología

- El identificador del usuario A se representa como $uname_a$.
- El agente principal de un usuario A se representa como Ta^*.
- Un agente cliente de un usuario A se representa como $Ta_i|i : 1..n$. Usamos esta misma nomenclatura para los terminales, dado que hay a lo sumo un agente por terminal.
- El protocolo utiliza el algoritmo AES se usa en modo CBC, por lo que se necesitará utilizar un vector de inicialización. Este vector de inicialización se representa como IV, y se genera de forma aleatoria para cada mensaje enviado.
- Supuestos los datos D, $\{D\}_{K_s}$ representa dichos datos cifrados con el algoritmo AES usando K_s como clave.
- Supuestos los datos D, $\{D\}_{SHA1}$ representa el resumen SHA-1 de los mismos.
- En todos los mensajes del protocolo se insertan datos generados aleatoriamente (ver apéndice A). Esos datos se representan como $ranb_i|i : 1..n$.
- En el protocolo se usan números aleatorios que se incrementan en las respuestas. Esos números, comúnmente denominados *nonces*, son números enteros sin signo de 32 bits de longitud. Se representan como $N_i|i : 1..n$,
- Con el fin de evitar ataques de replicación de mensajes, los agentes insertan marcas de tiempo en los mensajes. Esas marcas de tiempo se representan como $tstamp_{Ta}$ para una máquina (o agente) Ta .

4.5.1. Claves

El protocolo entre agentes SHAD se compone de varios tipos de mensajes, que se apoyan en tres claves privadas:

- **Clave de terminal:** Cada dispositivo debe tener asignado un secreto que usará para autenticarse ante su agente principal. El agente principal tiene que conocer la clave del terminal para que se pueda establecer un canal seguro entre ambos. Para un terminal Ta_i, la clave secreta de ese terminal será representada como KT_{Ta_i}.

- **Clave de sesión:** Es la clave que proporciona un canal seguro entre un agente común y su agente principal. Esta clave se genera aleatoriamente y se usa sólo para una sesión. La clave la genera el agente principal de forma aleatoria, y se representa como KS_{Ta_i} para un terminal Ta_i.

- **Clave de encarnación:** Junto con la clave de sesión, el agente principal proporciona una clave de encarnación a todos los agentes que inician una sesión con él. Esta es común para todos los agentes que inician su sesión con la misma encarnación del agente principal, y se utiliza para proporcionar seguridad al protocolo *Peer-to-Peer* realizado por el módulo P2PM. La encarnación del agente principal se define mediante el identificador de encarnación, que se representa para un usuario A como E_{Ta^*}, y su clave se representa como KE_{Ta^*}. Cuando el agente principal reinicia o decide cambiar de encarnación, genera otra clave de este tipo. Los agentes que arranquen desde ese momento obtendrán la nueva clave de encarnación. Los agentes que arrancaron en una encarnación anterior no podrán pedir servicio a la nueva encarnación. Un agente tampoco podrá usar su módulo P2PM con agentes de una encarnación diferente a la suya.

Descarte de mensajes

Los mensajes se descartarán y el protocolo no prosperará siempre que:

- Los datos esperados en ciertas partes del mensaje no sean correctos. Por ejemplo, si la cadena de caracteres que identifica el tipo de mensaje no corresponde a ninguno de los tipos conocidos, se descarta el mensaje.

- El número aleatorio (*nonce*) que se ha devuelto no coincide con el valor del *nonce* que se envió incrementado.

- El *nonce* que se recibe se encuentra en la lista (*cache*) de *nonces* usados recientemente. Esa *cache* es lo suficientemente grande [3] como para albergar los números aleatorios usados en un intervalo de tiempo δ. Si en algún momento la *cache de nonces* llega a su límite, se entenderá que se está sufriendo un ataque de denegación de servicio mediante inundación y se abortará la ejecución del agente, informando antes al usuario de esta situación. Nótese que para que un *nonce* llegue a entrar en *cache*, antes se ha tenido que verificar que el mensaje se ha creado con la clave necesaria (clave de terminal, clave de sesión, o clave de encarnación, dependiendo del tipo de mensaje). Por lo tanto, un ataque de estas características tan sólo puede llevarse a cabo generando mensajes maliciosos pero correctos (y eso implica el conocimiento de la clave utilizada para ese tipo de mensaje).

- La marca de tiempo insertada en el mensaje se considere desfasada. Siendo *time* el reloj local, *timestamp* estará desfasada si y sólo si:

$$|time - timestamp| \geq \delta$$

  Por esa razón, los relojes de los dispositivos del entorno tienen que estar relativamente sincronizados. La experiencia de uso nos muestra que un valor de 1800 segundos para δ es razonable, ya que el margen de sincronización de relojes para los distintos dispositivos usados es bastante

[3] El número máximo de *nonces* en la *cache* se ha fijado en 2048, un número muy superior al que se llegaría a alcanzar en condiciones extraordinarias de uso. Ese valor se ha asignado a partir de la experiencia de uso, y es razonable teniendo en cuenta la memoria que puede estar disponible en el dispositivo, incluso si es un dispositivo móvil.

amplio (pueden estar desfasados un total de 30 minutos) y la *cache de nonces* es lo suficientemente grande como para no saturarse en ese tiempo incluso en situaciones de uso extremo.

La combinación de las marcas de tiempo con la *cache de nonces* nos permite evitar ataques de repetición (*Replay Attacks*) [26].

- Los códigos MAC que se envían junto al mensaje no sean correctos. Un MAC no es correcto si el receptor no puede crear uno similar usando los mismos algoritmos, los mismos datos y la misma clave.

4.5.2. Protocolo de descubrimiento del agente principal

Este protocolo corresponde al flujo de datos `anuncio de servicio` entre la entidad *módulo de descubrimiento* de un agente común y la entidad *módulo de anuncios* del agente principal.

El protocolo sirve para descubrir al agente principal y establecer una sesión con él. La sesión proporciona los datos necesarios para solicitar servicio al agente principal y para activar el protocolo de P2P con los otros agentes comunes.

El protocolo se ejecuta siempre que arranca un nuevo agente de SHAD.

Supongamos que el usuario A posee un terminal que actúa como agente principal (Ta^*), y arranca otro terminal Ta_i que le pertenece. Se compone de dos tipos de mensajes:

- `ineedmainagent` (INMA): mensaje de descubrimiento del agente principal del usuario. Es un mensaje corto.

  Si pasa un tiempo determinado y no obtiene respuesta, el agente considerará que no existe un agente principal e intentará adoptar este papel él mismo.

- `iammainagent` (IAMA): mensaje con la respuesta al descubrimiento.

  En este mensaje se adjunta una clave de sesión KS_{Ta_i}, que utiliza Ta_i de aquí en adelante para solicitar servicio a Ta^*.

También se proporciona el identificador de la encarnación del agente principal E_{Ta^*} y la clave de encarnación KE_{Ta^*}, que se usa para el protocolo de compartición de secretos entre agentes comunes.

Es un mensaje largo, que se compone de dos partes: (i) una parte en la que se incluye la dirección del agente principal que se ofrece a dar servicio al agente que envió el mensaje INMA, y (ii) otra parte, llamada `fromowner`, que contiene la clave de sesión que se va a utilizar para establecer contacto con el agente principal, el identificador de encarnación y la clave de encarnación.

La descomposición del mensaje en dos partes se debe a razones de diseño, para desacoplar la asignación de agente principal del servicio que procesa el mensaje INMA. De esta forma el protocolo puede seguir siendo válido en una arquitectura formada por diferentes agentes principales (este tipo de arquitectura se tratará más adelante).

Para enviar estos mensajes, Ta_i tiene que tener asignada una clave KT_{Ta_i}, y Ta^* debe conocerla. Esa clave se debe almacenar en el repositorio de claves que obtiene el agente principal al arrancar. El protocolo de descubrimiento del agente principal se detalla en el cuadro Protocolo 1.

Protocolo 1 Protocolo de descubrimiento del agente principal.

$REQ = [\texttt{ineedmainagent}, uname_a, Ta_i, N_1, N_2, tstamp_{Ta_i}]$

$Ta_i \longrightarrow BROADCAST$
$\texttt{ineedmainagent}, uname_a, Ta_i$
$IV, \{ranb_1, \{REQ\}_{SHA1}, REQ\}_{KT_{Ta_i}}$

$RES = [\texttt{iammainagent}, uname_a, Ta_i, address, N_1 + 1, tstamp_{Ta^*}]$
$CAP = [\texttt{fromowner}, uname_a, Ta_i, N_2 + 1, tstamp_{Ta^*}, KS_{Ta_0}, KE_{Ta^*}, E_{Ta^*}]$

$Ta_i \longleftarrow Ta^*$
$\texttt{iammainagent}$
$IV', \{ranb_2, \{RES\}_{SHA1}, RES\}_{KS_{Ta_i}}$
$IV'', \{ranb_3, \{CAP\}_{SHA1}, CAP\}_{KT_{Ta_i}}$

En este protocolo, los *nonces* N_1 y N_2 aseguran que las dos partes del mensaje `iammainagent` corresponden a la petición `ineedmainagent` enviada. Se utilizan dos *nonces* diferentes para desacoplar las dos partes. El protocolo usa la *cache de nonces* para evitar mensajes repetidos. Las marcas de tiempo insertadas en los mensajes aseguran que la reinyección de mensajes antiguos no tenga efecto en la *cache de nonces* usados.

4.5.3. Protocolo de servicio de secretos

Este protocolo corresponde al flujo de datos `secretos (a.p.)` entre la entidad *módulo recolector* de un agente común y la entidad *módulo proveedor* del agente principal.

El protocolo sirve para obtener información acerca de los secretos que tiene el agente principal. Se utiliza comúnmente para obtener secretos del agente principal.

El protocolo se ejecuta siempre que una aplicación solicita autenticación al agente común local, y este no posee el secreto necesario en su *anillo de secretos* local.

Este protocolo está formado por dos mensajes:

- `tomainagent` (TMA): mensaje de petición de un secreto al agente principal. Es un mensaje corto. El mensaje contiene un comando para que se ejecute en el agente principal. Ese comando puede ser:

  - **listkeys**: pide al agente principal que le envíe la lista de secretos que tiene en su anillo. En la lista se envía la descripción de los secretos que tiene el agente principal (sus atributos), pero nunca la información secreta (claves y contraseñas). Los secretos que no se pueden compartir no aparecen el el listado que genera el comando.
  - **haskey descripción**: pregunta al agente si posee algún secreto que coincida con la descripción suministrada. La descripción especifica atributos asociados al secreto deseado.
  - **givekey descripción**: solicita al agente principal que devuelva el secreto que coincida con la descripción. En caso de que haya más de un secreto que coincida con la descripción, el agente principal devuelve el que encuentra antes en su *anillo de secretos*. Si el agente no posee ese secreto, devuelve un mensaje de error.

- `frommainagent` (FMA): mensaje con la respuesta a la petición. Es un mensaje corto.

El protocolo se detalla en el cuadro Protocolo 2. El comando del mensaje `tomainagent` se representa como *cmd*. La respuesta al comando que se envía en el mensaje `frommainagent` se representa como *cmdresponse*.

Protocolo 2 Protocolo de petición de servicio al agente principal.

$REQ = [\texttt{tomainagent}, uname_a, Ta_i, N_1, tstamp_{Ta_i}, cmd]$

$Ta_i \longrightarrow Ta^*$
$\texttt{tomainagent}, uname_a, Ta_i$
$IV, \{ranb_1, \{REQ\}_{SHA1}, REQ\}_{KS_{Ta_i}}$

$RES = [\texttt{frommainagent}, uname_a, Ta_i, N_1 + 1, tstamp_{Ta^*}, cmdresponse]$

$Ta_i \longleftarrow Ta^*$
$\texttt{frommainagent}, uname_a$
$IV', \{ranb_2, \{RES\}_{SHA1}, RES\}_{KS_{Ta_i}}$

El *nonce* N_1 y la marca de tiempo tienen la misma función que en los protocolos anteriores.

4.5.4. Protocolo *Peer-to-Peer*

Este protocolo corresponde al flujo de datos `secretos (P2P)` entre las entidades *módulo de P2P* de distintos agentes comunes. El protocolo sirve para obtener un secreto de otros agentes comunes.

El protocolo se ejecuta cuando una aplicación solicita autenticación al agente común local, este no posee el secreto necesario en su *anillo de secretos* local, y no ha podido establecer contacto con el agente principal.

El protocolo está formado por dos mensajes:

- `topeers` (TPE): mensaje de petición de un secreto a los agentes comunes.
- `frompeer` (FPE): mensaje con la respuesta a la petición.

El protocolo se detalla en el cuadro Protocolo 3. En el cuadro se ilustra el protocolo suponiendo que un terminal Ta_i necesita un secreto, lo solicita

mediante un radiado, y el terminal Ta_j se lo sirve.

Protocolo 3 Protocolo *Peer-to-Peer*.

Ta_i y Ta_j son terminales de A.

$REQ = [\texttt{topeers}, uname_a, Ta_i, E_{Ta^*}, N_1, tstamp_{Ta_i}, secret]$

$Ta_i \longrightarrow BROADCAST$

$\texttt{topeers}, uname_a, E_{Ta^*}$

$IV, \{ranb_1, \{REQ\}_{SHA1}, REQ\}_{KE_{Ta^*}}$

$RES = [\texttt{frompeer}, uname_a, Ta_j, E_{Ta^*}, N_1 + 1, tstamp_{Ta_j}, tuple]$

$Ta_i \longleftarrow Ta_j$

$\texttt{frompeer}, E_{Ta^*}$

$IV', \{ranb_2, \{RES\}_{SHA1}, RES\}_{KE_{Ta^*}}$

El *nonce* N_1 y la marca de tiempo tienen la misma función que en los protocolos anteriores.

4.6. Implementación de prototipo: NetFactotum

En esta sección se comentarán detalles de una implementación del agente SSO de SHAD para el sistema operativo Plan B [20; 21].

Este prototipo se basa en Factotum [41], el agente de seguridad del sistema operativo Plan 9, cuyo diseño y funcionamiento se describe en el capítulo dedicado al estado del arte.

De ahora en adelante, nos referiremos al prototipo del agente de SSO de SHAD como **NetFactotum**.

La arquitectura de la implementación de NetFactotum se presenta en la figura 4.5.

Factotum conforma prácticamente la totalidad de la entidad *módulo de secretos* del diseño de un agente SHAD. El resto del prototipo lo forman los siguientes componentes:

- Servidor *unicast*. Sólo está activo en el agente principal y posee flujo de ejecución propio. Este componente implementa la entidad *módulo proveedor* del diseño del agente SHAD principal.

  Se encarga de servir los secretos a los agentes principales procesando las peticiones que recibe en un puerto[4] TCP bien conocido (tcp!unicast!10023).

- Servidor *broadcast*. Está activo tanto en agentes comunes como en el agente principal. Implementa la entidad *módulo de anuncios* y *módulo de P2P* del diseño de los agentes SHAD.

  El servidor posee flujo de ejecución propio y escucha en un puerto UDP bien conocido de la dirección de broadcast de la subred. En el agente principal, se encarga de procesar los mensajes de descubrimiento de los agentes comunes (udp!broadcast!10023). En los agentes comunes, se encarga de procesar las peticiones *Peer-to-Peer* (udp!broadcast!10024).

- Módulo recolector (RM). Implementa la entidad de mismo nombre especificada en el diseño del agente SHAD. Se encarga de obtener las claves de los otros agentes. Se pone en contacto con los servidores descritos anteriormente a través de conexiones TCP (agente principal) o radiando datagramas UDP en la dirección de broadcast (agentes comunes).

- Módulo de descubrimiento (DM). Implementa la entidad de mismo nombre especificada en el diseño del agente SHAD. Su función es descubrir al agente principal radiando mensajes de broadcast en un puerto (udp!broadcast!10023).

[4]Se utiliza una notación similar a la utilizada en el sistema operativo Plan 9: *protocolo!dirección!puerto*.

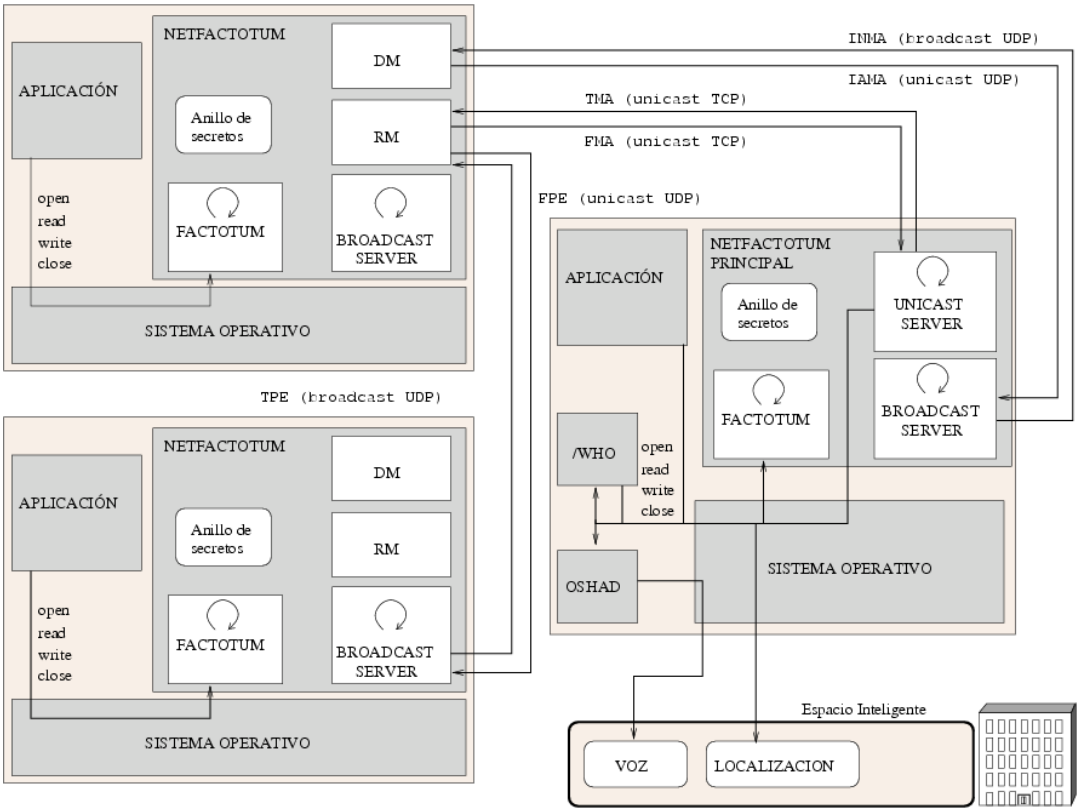

Figura 4.5: *Esquema del prototipo de agente SHAD para SSO (NetFactotum)*

Los distintos flujos de ejecución deben acceder concurrentemente a la estructura de datos que alberga las tuplas de Factotum (el anillo de secretos). Para coordinar el acceso concurrente se han utilizado cierres con espera activa (*spin locks*), debido a que la espera que deben realizar es corta. Por otra parte, la probabilidad de encontrar el cierre echado en el anillo es baja, debido a la naturaleza del propio agente y el tipo de uso para el que está diseñado.

La implementación aprovecha los servicios que ofrece en espacio inteligente del Laboratorio de Sistemas [19], en concreto la infraestructura de localización y la de mensajes de voz. El uso de estas infraestructura resulta extremadamente útil para automatizar ciertos procesos, reducir la obstrucción al usuario y emitir notificaciones al usuario. Por ejemplo, las notificaciones de voz resultan útiles a la hora de advertir al usuario de una situación determinada, y la información

de localización puede ayudar a un agente a compartir sus secretos sin pedir confirmación.

Aunque el prototipo haga uso de estos servicios, no depende de ellos para su funcionamiento. Por tanto, el prototipo tolera el fallo o la ausencia de dichos servicios.

4.6.1. Claves

Claves de terminales

La claves de los terminales se almacenan en la NVRAM junto con el nombre del dueño. De esta forma, el agente NetFactotum es capaz de leer la clave en tiempo de arranque, y así poder descubrir al agente principal y autenticarse ante él.

Las claves se almacenan en claro en la NVRAM. Por lo tanto, son vulnerables ante ataques físicos contra la máquina. Esta posibilidad se trata en el capítulo 7, dedicada a las limitaciones del modelo.

Si el usuario arranca siempre el agente principal en el mismo dispositivo (el UbiTerm), entonces ese dispositivo no necesita tener una *clave de terminal* asignada. De esta forma, en caso de pérdida del UbiTerm, no queda comprometida ninguna *clave de terminal*. Esta ventaja es significativa, dado que el UbiTerm es un dispositivo móvil que puede extraviarse.

Representación de las claves

NetFactotum utiliza el mismo método para almacenar las claves que Factotum. Las claves se almacenan en tuplas, que además contienen atributos que aportan información sobre la clave. Las claves de Factotum tienen un atributo común, `proto`, que especifica el tipo de secreto, esto es, el protocolo al que hace referencia. Los otros atributos varían según el tipo de clave (el servicio para el que sirve el secreto, o campo de una tupla) [55].

Los atributos secretos de una tupla van precedidos por el carácter `!`. Estos

atributos secretos nunca se presentan en claro por pantalla. Por ejemplo, una tupla convencional de Factotum para el protocolo `P9SK1` se representa como sigue:

```
proto=p9sk1 dom=urjc.es user=esoriano !password=foopasswd
```

En el ejemplo, el atributo `dom` indica el dominio de autenticación para el que sirve la tupla, `user` el identificador del usuario al que pertenece, y `password` la contraseña. El atributo `password` contiene un carácter `!` delante, por tanto Factotum nunca lo escribirá en claro.

Factotum almacena las tuplas en su memoria principal. La memoria principal de Factotum está protegida contra depuración y nunca se pagina a área de intercambio. NetFactotum actúa del mismo modo.

Claves para el protocolo SHAD

Para almacenar los secretos relacionados con el protocolo de NetFactotum, se ha creado un nuevo tipo de tupla que se identifica mediante el campo `proto=shad`.

Estas tuplas almacenan las claves privadas de las máquinas que pertenecen al usuario (KTa_i para un terminal Ta_i):

$$\texttt{proto=shad type=term machine=}Ta_0\ \texttt{!tsecret=}KTa_0$$
$$\texttt{proto=shad type=term machine=}Ta_1\ \texttt{!tsecret=}KTa_1$$
$$\ldots$$

Los secretos que consisten en claves criptográficas se almacenan en la tuplas aplanados en una cadena de caracteres que representa una cifra hexadecimal.

Campos para claves convencionales

Se han añadido varios atributos a las tuplas convencionales de Factotum. Mediante el uso del estos atributos, el usuario es capaz de ajustar el nivel

de seguridad para sus secretos. El usuario debe aceptar el compromiso entre comodidad de uso y nivel de seguridad de sus secretos. Por tanto, tiene que considerar que los secretos que no están disponibles desde ciertos terminales tienen que proporcionarse explícitamente, con la obstrucción que dicha acción conlleva. Por otro lado debe e asumir que los secretos que están disponibles desde terminales vulnerables a ataques están menos seguros que los que no están accesibles.

De las restricciones indicadas, SHAD aplica siempre la más restrictiva. En el caso de los atributos que utilizan información de localización, si dicha información no está disponible, el agente principal no sirve el secreto.

Por ejemplo, la siguiente tupla que contiene un secreto para el protocolo SSH incluyendo todos los posibles atributos:

`proto=pass server=s1 service=ssh user=`*usuario* `machine=`*máquina*
`noremoteaccess nopeeraccess needconfirm samelocation`
`userlocation=`*localización* `clientlocation=`*localización*
`accesiblefrom=`*máquina* `!password`

Los atributos de SHAD en dicha tupla indicarían:

- `noremoteaccess`: indica que ningún agente puede compartir ese secreto con otro. Si una tupla no contiene este atributo, el agente principal considera que puede enviarla a un agente común.
- `nopeeraccess`: significa que el secreto no puede compartirse entre agentes comunes a través del protocolo *Peer-to-Peer*. El agente principal sí puede enviar el secreto a los agentes que lo requieran.
- `needconfirm`: obliga a que la acción tenga que confirmarse explícitamente por el usuario. Las confirmaciones se realizan en la máquina que ejecuta el agente principal.
- `userlocation=`*localización*: indica que el secreto se puede compartir sólo si el usuario se encuentra en la localización especificada en el atributo.

En caso de que la tupla contenga también el atributo `needconfirm`, se pide confirmación sólo si el usuario se encuentra en esa ubicación. En caso contrario, se deniega el acceso al secreto. Puede haber más de una instancia de este atributo dentro de una tupla para especificar varias localizaciones desde donde el secreto está disponible.

- `clientlocation=`*localización*: funciona igual que el atributo anterior pero comprobando la localización de la máquina cliente. Puede haber más de una instancia de este atributo en una tupla.
- `samelocation`: significa que el secreto es accesible si la localización física del usuario y de la máquina cliente es la misma. Respecto a las confirmaciones, se aplica el mismo criterio que en los dos atributos anteriores.
- `accesiblefrom=`*máquina*: indica que el secreto sólo es accesible desde la máquina indicada en el atributo. Puede haber más de una instancia de este atributo en una tupla.

4.6.2. Control de los agentes

Factotum ofrece una interfaz de control a través de un sistema de ficheros. La interfaz consiste en un pequeño sistema de ficheros virtuales, sobre los que se pueden realizar lecturas y escrituras. Dicho sistema de ficheros se monta en un punto determinado, normalmente `/mnt/factotum`. Una vez montado, los ficheros virtuales que forman la interfaz del programa aparecen dentro de ese directorio. La interfaz consiste en lecturas y escrituras sobre esos ficheros.

Por ejemplo, para obtener la lista de secretos guardada en el anillo de secretos, se puede ejecutar

```
cat /mnt/factotum/ctl
```

y para borrar todas las tuplas que tengan como protocolo `p9sk1` se puede ejecutar

```
echo 'delkey proto=p9sk1' >/mnt/factotum/ctl
```

El prototipo de SHAD posee la misma interfaz, pero con varios ficheros de control adicionales.

El agente NetFactotum exporta un sistema de ficheros, que al igual que el de Factotum, se suele montar en el directorio /mnt del sistema. De ahora en adelante, para los ejemplos en los que se accede a la interfaz de NetFactotum se supondrá que el sistema de ficheros está montado en dicho directorio.

El fichero `getkey` proporciona una interfaz para ordenar a un agente común que adquiera un secreto. El fichero de control admite una cadena de caracteres con formato de tupla de Factotum, sin el atributo secreto especificado. Si la cadena escrita en el fichero tiene una sintaxis correcta, entonces el agente intenta conseguir la tupla completa del agente principal o de los otros agentes comunes de su misma encarnación. Si el agente consigue obtener la tupla completa, la almacena en el anillo de secretos. El fichero `getkey` no está disponible en el agente principal.

Supongamos que el usuario Nemo desea que el agente del terminal en el que está trabajando adquiera el secreto para autenticarse ante el servidor de ficheros de Plan B del dominio `lsub.org`:

```
echo 'proto=p9sk1 dom=lsub.org user=nemo'
    >/mnt/netfactotum/getkey
```

El fichero `hold` hace que los hilos servidores del agente no acepten peticiones, y por tanto, el agente SHAD actúe como un agente Factotum convencional. Para deshabilitar dicha funcionalidad, basta con escribir una cadena de control `hold` en el fichero. Para volver a habilitar la funcionalidad, se debe escribir la cadena `unhold` en el fichero. Por ejemplo, para bloquear el agente:

```
echo hold >/mnt/netfactotum/hold
```

Como medida contra el extravío del dispositivo que ejecuta el agente principal, el usuario puede definir un tiempo de inactividad tras el cual queda

bloqueado. Cuando se cumple el tiempo de inactividad indicado, el agente se queda bloqueado hasta que el usuario se vuelva a autenticar explícitamente. Por lo tanto, provoca obstrucción al usuario, pero este puede definir el tiempo de inactividad según sus preferencias. Como en casos anteriores, el prototipo ofrece los mecanismos necesarios para que el usuario establezca sus propias políticas.

El tiempo de inactividad se puede definir mediante el fichero `timeout`, que espera recibir una cadena de caracteres que especifique el tiempo en minutos. Por ejemplo, para establecer un tiempo de espera de quince minutos, deberíamos escribir:

```
echo 15 >/mnt/netfactotum/timeout
```

La cadena de control `'0'` indica que el agente no se debe bloquear por inactividad. Ese es el valor establecido por omisión.

Por último, el agente principal exporta otro fichero llamado `location`, que se describirá más adelante en la Sección 4.6.5.

4.6.3. Repositorio de claves

En el prototipo NetFactotum, el repositorio de claves se obtiene mediante Secstore[169], de la misma forma que lo hace Factotum. Cuando un agente NetFactotum arranca y asume el papel de agente principal , adquiere su almacén de claves de un servidor de autenticación a través de la red. La principal diferencia con Factotum radica en que sólo el primer agente en arrancar necesita obtener el fichero del Secstore. Por tanto, sólo se depende de una entidad centralizada (el servidor de Secstore) en el momento de arrancar el agente principal.

Para eliminar esa dependencia, se podría obtener el almacén de claves de un dispositivo local, por ejemplo de una tarjeta de memoria SD Card. La tarjeta tendría el mismo fichero de Secstore, cifrado con un algoritmo fuerte de clave simétrica. Esto puede conseguirse ejecutando el servidor de Secstore en el UbiTerm.

4.6.4. Confirmaciones: Oshad

El usuario necesita una interfaz para recibir avisos de su agente principal y poder consentir la cesión de un secreto cuando sea necesario. Para ello se ha desarrollado un programa, Oshad, que ofrece una interfaz gráfica a través del sistema gráfico de Plan B, Omero [18].

Oshad ofrece una interfaz sencilla para que el usuario pueda confirmar las operaciones tan sólo pulsando un botón, y ofrece una interfaz para su control desde NetFactotum que se basa en un pequeño sistema de ficheros virtual, siguiendo con las lineas generales de diseño de Plan B.

El sistema de ficheros de Oshad ofrece tres ficheros de control: `confirm`, `ads`, y `voice`.

`Confirm` es el fichero de control en el que se indican las operaciones que el usuario debe confirmar. Este fichero de control espera una cadena de caracteres con el siguiente formato:

`‘‘nonce cliente protocolo servidor’’`

`Nonce` es un entero sin signo de 32 bits generado de forma aleatoria que identifica la petición de confirmación que se le pasa a Oshad. `Cliente` es el nombre de la máquina donde ejecuta el agente que está pidiendo servicio al principal. `Protocolo` especifica el tipo de secreto que solicita el agente remoto. Por último, `servidor` es el nombre de la máquina que ofrece el servicio para el que se va a usar el secreto reclamado.

Cuando se escribe una cadena con esta sintaxis en el fichero `confirm`, Oshad pregunta al usuario a través de su interfaz gráfica integrada en Omero. En la figura 4.6 se muestra una captura de pantalla de Oshad pidiendo confirmación para ceder un secreto a un agente SHAD que lo solicita. Cuando el usuario se pronuncia al respecto, su decisión se puede leer del mismo fichero: la lectura del fichero de control devuelve el `nonce` junto con la decisión del usuario.

Por tanto, cuando una aplicación (que comúnmente será NetFactotum, aunque otras aplicaciones pueden usar Oshad de la misma manera, ya que están

Figura 4.6: *Captura de pantalla de Oshad pidiendo confirmación para ceder al agente SHAD de la máquina 'osiris' la contraseña de acceso al servidor SSH de la máquina 'ronin' .*

totalmente desacoplados) necesita confirmación, sigue los pasos enumerados a continuación:

1. Abre el fichero `confirm` en modo lectura/escritura.

2. Escribe la cadena de control en el fichero `confirm`.

3. Lee del fichero `confirm` en repetidas ocasiones la confirmación o cancelación de la operación, identificando en contenido mediante el `nonce` utilizado en la cadena de control. Cada aplicación definirá su tiempo de espera máximo (timeout) para considerar que la confirmación ha fallado.

El fichero `ads` espera cadenas de caracteres correspondientes a avisos destinados al usuario. Estos avisos se imprimen en la interfaz gráfica de Oshad. Además, Oshad saca partido de la infraestructura de voz de Plan B [19], avisando a través de esta al usuario. Cuando un programa escribe un aviso en el fichero `ads`, Oshad lo reenvía a la infraestructura de voz del entorno inteligente. La infraestructura de voz de Plan B redirige el mensaje a la localización actual del usuario, donde un sintetizador de voz lo reproduce por un altavoz. Oshad es capaz de usar la infraestructura de voz si esta está disponible, pero no depende de ella para su funcionamiento: el usuario puede seguir confirmando operaciones y recibiendo mensajes a través de la interfaz aunque no esté disponible ninguna infraestructura del entorno inteligente (localización o voz).

Por último, Oshad ofrece un fichero `voice` que sirve para indicar si el usuario desea utilizar la infraestructura de voz del espacio inteligente o no. El fichero admite sólo dos cadenas de control, `on` y `off`, que activan y desactivan el servicio de voz respectivamente.

4.6.5. Información de localización

Mediante el uso de información de localización y de las restricciones descritas en la sección 4.6.1, el agente principal de SHAD es capaz de ceder secretos a los demás agentes en base al contexto físico de las personas y máquinas.

El prototipo implementado accede a la información de localización que ofrece el espacio inteligente del Laboratorio de Sistemas mediante dos servicios: `/who` ofrece información de contexto sobre personas y `/what` ofrece información de localización sobre objetos. Estos servicios se describen en profundidad en otros trabajos [17; 19].

```
; ls -l /who/esoriano
--rw-rw-r-- M 11 esoriano     esoriano 1591 Jun 28  2004 /who/esoriano/face.gif
--rw-rw-r-- M 11 esoriano     esoriano    5 May 31 21:20 /who/esoriano/letters
--rw-rw-r-- M 11 esoriano     esoriano 2197 May  5 15:07 /who/esoriano/msgs
--rw-rw-r-- M 11 esoriano     esoriano    7 May 31 21:23 /who/esoriano/status
--rw-rw-r-- M 11 esoriano     esoriano    4 May 31 21:23 /who/esoriano/where
; cat /who/esoriano/status
online
; cat /who/esoriano/where
124
```

Cuadro 4.1: *Contenido y uso de* `/who` *para el usuario* `esoriano`.

El servicio `/who` ofrece una interfaz de sistema de ficheros a través de la cual proporciona información de contexto sobre usuarios.

El sistema ofrece los siguientes ficheros con información de localización para cada usuario del sistema:

- `status`: muestra si el usuario está presente en el sistema.

Figura 4.7: *Sensores, receptores y balizas de la infraestructura de localización del entorno para el que se ha desarrollado NetFactotum.*

- `where`: ofrece la localización del usuario.

El cuadro 4.1 muestra la interfaz de ficheros de `/who` para un usuario del sistema. La infraestructura de localización obtiene dicha información de distintas fuentes, como sensores de X10 [196], balizas emisoras de ultrasonidos, programas que implementan heurísticas definidas para usuarios concretos y otros métodos. La figura 4.7 muestra el hardware utilizado para la infraestructura de contexto.

Por otra parte, `/what` ofrece un sistema de ficheros con un directorio para cada una de las máquinas del entorno ubicuo que contiene una serie de ficheros con información de contexto. Dentro de ese directorio se sirve un fichero llamado `where` que indica la localización de la máquina. El cuadro 4.2 muestra la interfaz de ficheros de un terminal del espacio inteligente.

SHAD supone que la infraestructura de información de contexto es fiable y segura, aunque no tolerante a fallos. Si la información de contexto no está disponible a la hora de procesar una petición que solicita un secreto que contiene una restricción relacionada con la localización (`samelocation`, `userlocation` o `clientlocation`), el agente principal denegará la petición. Por tanto, cuando la información de contexto no está disponible desde el agente principal, los secretos que dependen de ella no están disponibles para los agentes que los soliciten.

```
; ls -l /what/isis
--rw-r--r-- M 57 esoriano esoriano  9 Jun  1 11:53 /what/isis/owner
--rw-r--r-- M 57 esoriano esoriano  8 Jun  1 11:53 /what/isis/role
--rw-r--r-- M 57 esoriano esoriano 14 Jun  1 11:53 /what/isis/vgasize
--rw-r--r-- M 57 esoriano esoriano  4 Jun  1 11:53 /what/isis/where
; cat /what/isis/owner
esoriano
; cat /what/isis/where
124
```

Cuadro 4.2: *Contenido y uso de* `/what` *para el terminal* `isis`.

El uso de las restricciones de localización provoca cierta dependencia del servicio de contexto del espacio inteligente. Si el usuario considera que dicha dependencia no es apropiada para su entorno (por ejemplo, porque sea un entorno altamente particionable), basta con eliminar de sus secretos las restricciones relacionadas con la infraestructura de contexto. El usuario puede eliminar definitivamente los atributos de este tipo en el propio almacén de secretos. Alternativamente, el usuario puede deshabilitar las restricciones de localización de forma transitoria. Nótese que esta operación consiste en eliminar un atributo dentro del anillo de secretos que reside en la memoria del propio agente principal, por lo tanto no es necesario reconfigurar el agente ni reiniciar el proceso. Basta con escribir una cadena de control en el fichero `location` que exporta la interfaz del agente principal:

```
echo disable >/mnt/netfactotum/location
```

Para habilitar de nuevo las restricciones relacionadas con la infraestructura de contexto, hay que escribir la siguiente cadena de control:

```
echo enable >/mnt/netfactotum/location
```

La escritura de esta cadena de control la puede llevar a cabo Oshad, de tal forma que el usuario sólo tiene que activar un botón en la interfaz para deshabilitar el uso de la infraestructura de contexto.

4.7. Evaluación y experiencia de uso

El prototipo es lo suficientemente rápido desde el punto de vista de un usuario normal, como muestra la tabla presentada en el cuadro 4.3. La tabla muestra el tiempo medio para establecer una conexión SSH desde un cliente que utiliza un agente de SSO. La primera columna muestra el tiempo medio de acceso utilizando un agente local de SSO, en concreto, Factotum. La segunda columna muestra el tiempo medio desde un cliente que usa SHAD, obtieniendo el secreto necesario del agente principal y establece la conexión. Por último, la tercera columna muestra el tiempo medio cuando el agente común obtiene la contraseña de otro agente común mediante el protocolo *Peer-to-Peer*.

	Factotum	SHAD	SHAD P2P
Tiempo (seg)	0.12	0.14	4.06

Cuadro 4.3: *Tiempo medio para establecer una conexión SSH.*

El tiempo medio de conexión en el tercer caso está claramente influido por el tiempo dedicado al descubrimiento del agente principal. El experimento se ha realizado con tres ordenadores comunes Pentium 4 dotados con 1Gb, 512 Mb y 256 Mb de memoria principal. Los ordenadores se encuentran interconectados mediante una red Ethernet a 100 Mbit/s.

Como se puede observar, en el caso de obtener la clave del agente principal, el tiempo de respuesta es lo suficientemente corto como para que el usuario no

aprecie que está utilizando un agente de SSO distribuido y no un agente SSO tradicional.

Sin embargo, en el tercer caso el retardo es de alrededor de cuatro segundos, tiempo que sí es apreciable por el usuario. En este caso, el usuario puede apreciar el retardo en la autenticación. No obstante, no se fuerza al usuario a introducir una contraseña o autenticarse mediante algún dispositivos especial (p.e. Smartcard), acciones que podrían haber consumido la misma cantidad de tiempo, además de aportar obstrucción al usuario y eliminar la transparencia en la autenticación. Se podría reducir el *timeout* del descubrimiento del agente principal para reducir el tiempo de acceso del protocolo *Peer-to-Peer*, pero dicha modificación podría tener un efecto negativo en el descubrimiento de agentes lentos.

Las medidas que se muestran en la cuadro 4.3 corresponden a claves que no requieren confirmación por parte del usuario ni acceso a la infraestructura de contexto. El tiempo de envío de claves que requieren una confirmación por parte del usuario depende del tiempo de reacción del usuario.

El cuadro 4.4 muestra el tiempo medio para montar un servicio de Plan B mediante el protocolo 9P. La primera columna muestra el tiempo medio cuando el secreto no tiene asignado ningún atributo que restrinja su uso en base a la localización. La segunda columna muestra el tiempo medio cuando el secreto usa un atributo de localización, en concreto `userlocation`.

	sin localización	*userlocation*
Tiempo (seg)	0.15	0.17

Cuadro 4.4: *Tiempo medio para montar un volumen de Plan B usando la infraestructura de contexto.*

Se puede observar en la tabla que la diferencia entre el tiempo medio para montar el volumen usando un secreto sin restricción de localización y usándolo es mínima. Como ya se ha explicado en secciones anteriores, la arquitectura de contexto que usa SHAD es una jerarquía de directorios y ficheros. La diferencia

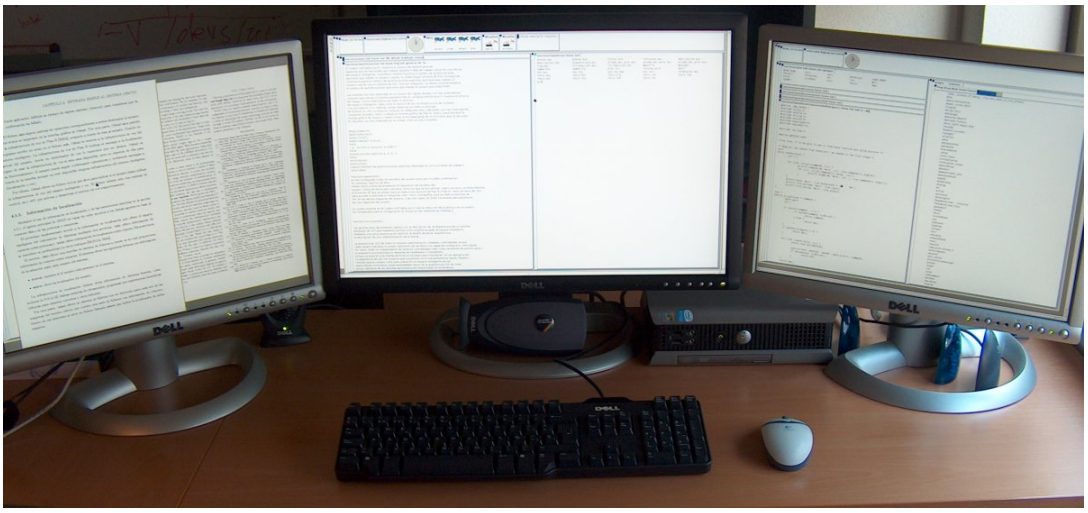

Figura 4.8: *Entorno de pruebas del sistema de SSO de SHAD.*

entre los dos tiempos medios mostrados corresponde con el tiempo que requiere acceder a la jerarquía de directorios que exporta la infraestructura de contexto y compararla con el valor que posee la tupla.

Se debe considerar que ambos experimentos se han realizado sobre una red a 100 Mbit/s, y el agente principal de SHAD está ideado para ejecutar sobre un dispositivo móvil que utiliza una red inalámbrica. Por lo tanto, debemos considerar que el retardo sobre este tipo de redes será mayor. También hay que considerar que la capacidad de procesamiento de un dispositivo móvil es más limitada. Por consiguiente, el tiempo de respuesta del agente se verá afectado.

El prototipo se ha usado durante un año. Durante este tiempo, el prototipo ha sufrido modificaciones con la intención de mejorar su funcionamiento y reducir la obstrucción al usuario. Nuestra experiencia de uso indica que el prototipo es de gran utilidad, debido a que reduce considerablemente la obstrucción en el espacio inteligente en el que se ha probado [19].

El prototipo se ha probado en el mismo entorno en el que se está desarrollando el presente documento. Este entorno se puede poner como ejemplo para remarcar la necesidad de un sistema de SSO real para sistemas distribuidos actuales.

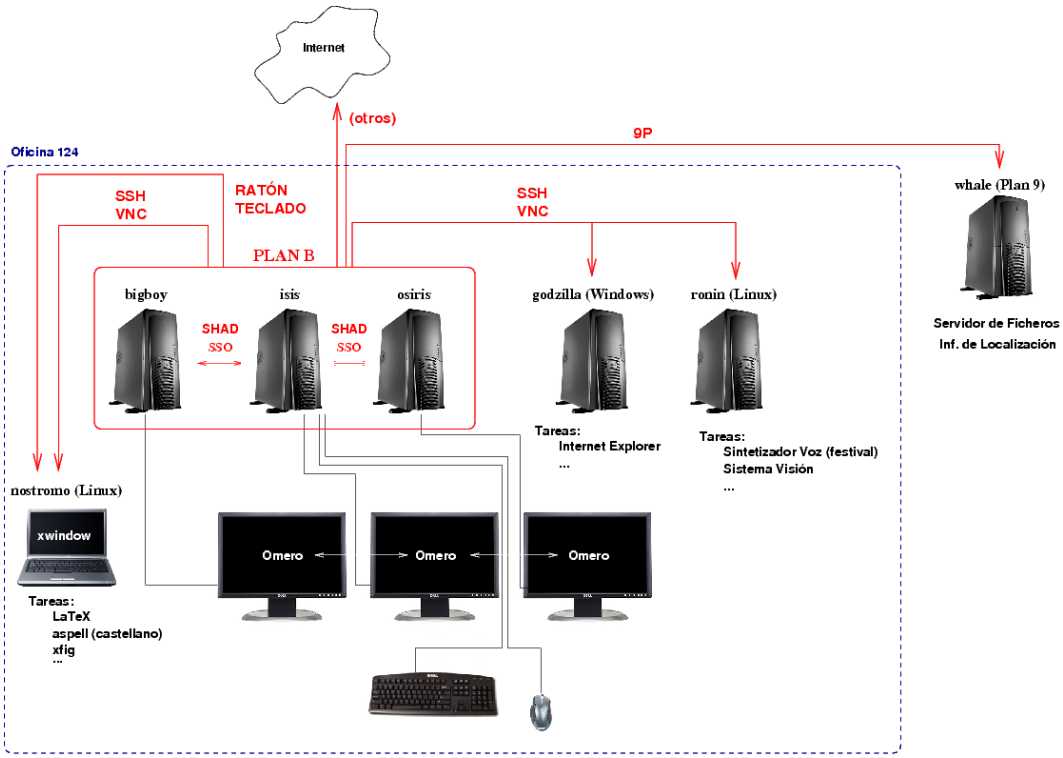

Figura 4.9: *Esquema del entorno de trabajo en el que se han realizado las pruebas.*

El entorno se compone de de un total de seis máquinas, tres de las cuales son terminales de Plan B ejecutando agentes SHAD. Las otras máquinas son un servidor Linux, un PC Windows y un ordenador portátil Linux. La figura 4.8 muestra el entorno de trabajo.

Los tres terminales de Plan B comparten teclado y ratón, y utilizan el sistema gráfico de Plan B, Omero, para distribuir la interfaz gráfica de usuario y direccionar la entrada/salida de los procesos que se ejecutan en distintas máquinas. El ratón y el teclado también se comparten con las máquinas Linux.

Los terminales de Plan B montan su sistema de ficheros raíz de un servidor remoto y utilizan distintos servicios del espacio inteligente, tales como el servicio de envío de mensajes de voz, la infraestructura de contexto, y los actuadores X10. También usan servicios comunes localizados en otras redes.

Los terminales Plan B acceden y controlan las otras máquinas Linux y

Windows mediante VNC y SSH. Los terminales de Plan B acceden a estas máquinas para realizar tareas como compilar el fuente LaTeX, emitir mensajes de voz, obtener información sobre un sistema de visión que detecta cartas en un casillero, dibujar diagramas, reproducir videos, abrir sitios WWW que necesitan un navegador específico, etc. La mayoría de estas tareas se redireccionan a las máquinas automáticamente desde Plan B. La figura 4.9 muestra el entorno de trabajo.

Como se puede observar, en un entorno como el descrito la obstrucción al usuario es un problema grave. El cuadro 4.5 muestra el número de autenticaciones explícitas que se han tenido que realizar durante 5 días de trabajo común en el entorno de desarrollo.

La primera columna muestra el número de autenticaciones que tiene que realizar un usuario cuando no utiliza ningún sistema de SSO. La segunda columna muestra el número de autenticaciones explícitas que tiene que realizar un usuario cuando usa un sistema de SSO *por máquina*. La última columna muestra el número de autenticaciones que tiene que realizar el usuario que utiliza SHAD.

Terminal	Sin SSO	Factotum	SHAD
isis	129	10	10
bigboy	73	10	0
osiris	70	16	0
Total	272	36	10

Cuadro 4.5: *Número de autenticaciones explícitas obtenidas en cinco jornadas de trabajo.*

Para este experimento se han configurado todos los secretos del usuario para que no pidan confirmación. Sin embargo, algunos de ellos utilizan restricciones de localización. El repositorio de secretos del usuario consta de 24 secretos, entre los que se encuentran cuatro secretos correspondientes al protocolo 9P que se utilizan para acceder a los servicios de Plan B y Plan 9, siete secretos de SSH para acceder a sesiones en máquinas Linux, cinco contraseñas para acceder a

sesiones de VNC en las demás máquinas del espacio, y las tres claves de SHAD necesarias para autenticar las tres máquinas del usuario.

Se puede observar en el cuadro 4.5 que la reducción de la obstrucción al usuario es considerable para la configuración en la que se han obtenido las medidas. Un entorno como el que se ha descrito no es utilizable sin un mecanismo que automatice la autenticación del usuario.

Hay que remarcar que el número de autenticaciones explícitas que muestran las medidas para SHAD es mayor que la que es habitual, debido a las tareas que se llevaron a cabo dicha semana, que incluían depuración del núcleo del sistema operativo de Plan B. Para realizar dicha tarea, es necesario reiniciar todas las máquinas que ejecutan Plan B (incluida la que ejecuta el agente principal de SHAD). Nótese que el usuario sólo se tiene que autenticar una vez para reiniciar todos sus terminales. Cuando se utiliza SHAD para tareas que no necesitan reiniciar los terminales, normalmente el usuario sólo debe autenticarse una vez por sesión (por lo general, la sesión dura toda la jornada).

Por todas estas razones, consideramos que la arquitectura SHAD para SSO es de gran utilidad para los usuarios de un espacio inteligente.

Capítulo 5

Compartición de terminales

5.1. Introducción

En el presente capítulo se detalla la arquitectura y los mecanismos necesarios para que un usuario del sistema pueda arrancar una máquina que pertenece a otro usuario con el que tiene confianza mutua.

Una vez que el usuario ha arrancado el terminal a su nombre, lo puede utilizar normalmente, como cualquiera de sus propios terminales.

La arquitectura se basa en la arquitectura de SSO descrita en el capítulo anterior. En concreto, el agente SHAD para la compartición de terminales añade una extensión al protocolo de descubrimiento del agente principal de SHAD descrito en el capítulo anterior. También incluye un nuevo secreto al esquema, y nuevos atributos para los secretos del repositorio.

El agente principal de SHAD tiene ahora dos cometidos fundamentales:

- Servir los secretos del usuario, tal y como lo hace el algente principal de SHAD en la arquitectura de SSO descrita en el capítulo anterior.

- Controlar la cesión de un terminal de un usuario a otro. La cesión de un terminal implica que pertenecerá al usuario huésped de manera transitoria.

De ahora en adelante, cuando nombremos al agente SHAD, nos estaremos refiriendo al **agente SHAD de compartición de terminales**.

5.2. Emparejamiento de UbiTerms

En el capítulo 3 se presentó el esquema básico de confianza entre los usuarios de SHAD, que consiste en *emparejar usuarios* mediante el *emparejamiento de sus UbiTerms*.

Cuando dos usuarios confían entre sí, emparejan sus UbiTerms. En la práctica, este proceso consiste en añadir un nuevo secreto en el repositorio de secretos de ambos usuarios. De esa forma, los agentes SHAD principales de

ambos usuarios pueden autenticarse mutuamente, ya que los dos comparten un mismo secreto.

El secreto que se añade al repositorio es una nueva clave de criptografía simétrica. Nos referiremos a esa clave como **clave de emparejamiento** entre dos humanos. La clave de emparejamiento entre dos usuarios A y B será representada indistintamente como $K_{a,b}$ ó $K_{b,a}$.

En la arquitectura no se contempla la transitividad de la confianza entre humanos, ya que esta no existe en el mundo real. Por consiguiente, ninguna entidad excepto los UbiTerms de los dos usuarios puede conocer la clave de emparejamiento.

El emparejamiento de los UbiTerms se puede automatizar de distintas maneras, dependiendo del nivel de seguridad y el nivel de obstrucción que se desee para los usuarios.

Por ejemplo, se podría realizar mediante un programa que adquiera el secreto por un enlace IRdA y lo almacene en el repositorio de secretos del usuario. Dado que la distancia máxima entre los dispositivos IRdA certificados como *IrReady* [79] es menor de dos metros [197], es improbable que un adversario intercepte la comunicación entre los UbiTerms sin que los usuarios se percaten de su presencia.

Otra forma menos confortable sería generar la clave a partir de un frase secreta que ambos usuarios introducen en sus UbiTerms. En nuestra opinión este método no es más seguro que la anterior, ya que está sujeto a que un tercero intercepte de la comunicación entre los dos humanos y a ataques basados en *ingeniería social* [110].

En todo caso, el emparejamiento de dos UbiTerms se basa únicamente en el intercambio de un secreto en forma de clave de criptografía simétrica. La automatización del proceso queda en manos de la implementación de la arquitectura.

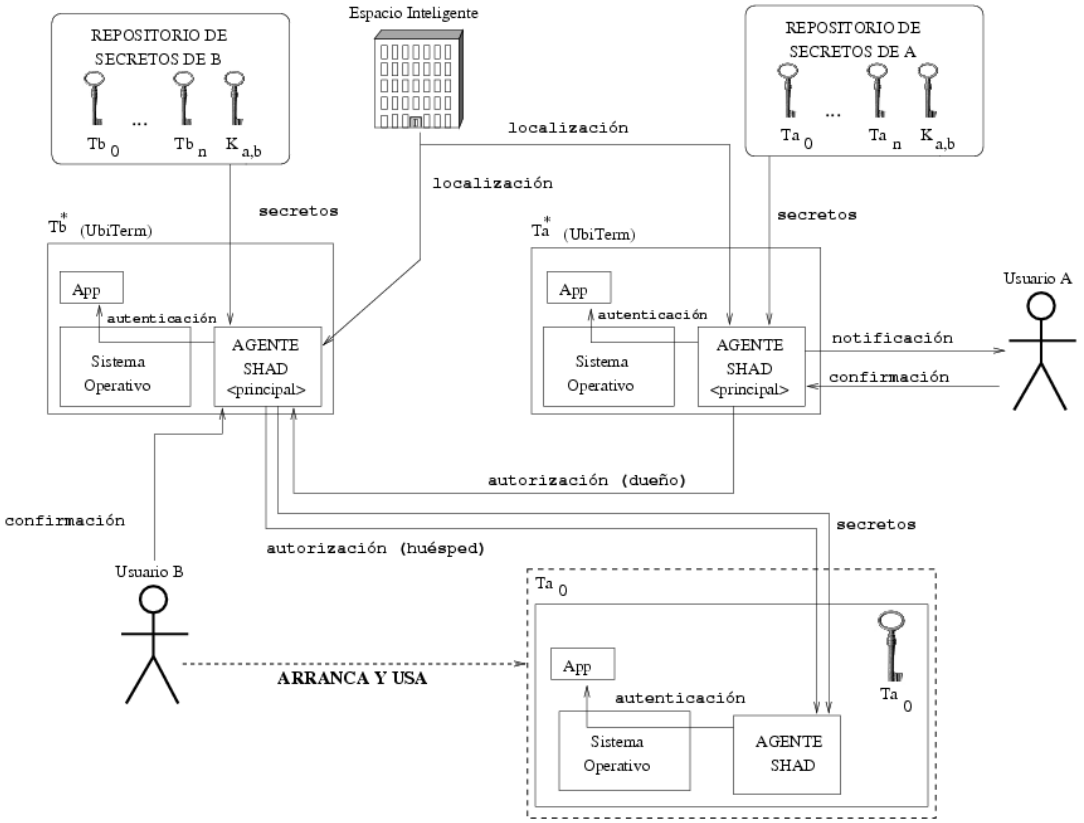

Figura 5.1: *Arquitectura global del sistema para la cesión de terminales.*

5.3. Arquitectura SHAD para la cesión de terminales

La figura 5.1 muestra la arquitectura global para la compartición de terminales entre usuarios. En la figura se pueden encontrar las siguientes entidades:

- **Los usuarios** son humanos que interaccionan entre sí en el espacio inteligente. Los usuarios proveen confirmaciones al sistema para realizar ciertas operaciones y reciben notificaciones del espacio inteligente y de sus UbiTerms. Los usuarios también deben emparejar sus agentes principales entre ellos en tiempo de configuración.

- **Los terminales** que pertenecen a los usuarios. En la figura, el terminal que actúa como UbiTerm para un usuario A se representa como Ta^*, y los demás terminales como $Ta_i|i : 1..n$.
- **El espacio inteligente** provee servicios a los usuarios del sistema, de la misma forma que en la arquitectura de SSO.
- **El repositorio de secretos** contiene todos los secretos del usuario: las claves de los terminales, los secretos para acceder a los servicios comunes y las claves de emparejamiento de SHAD.
- **Los agentes SHAD**, que controlan la cesión de un terminal. El agente principal del dueño del terminal autoriza la cesión de sus terminales. El agente principal del usuario huésped gestiona la petición de autorización y autoriza el arranque del terminal a su nombre. El agente común que ejecuta en el terminal cedido inicia el protocolo de compartición, y si prospera, permite al usuario usar el terminal.
- **Las aplicaciones** juegan el mismo papel que en la arquitectura de SSO.
- **El sistema operativo** juega el mismo papel que en la arquitectura de SSO.

5.3.1. Esquema de funcionamiento

El esquema general de funcionamiento se puede ilustrar mediante una secuencia de acciones. La secuencia está basada en el escenario presentado en la figura 5.1:

1. El usuario B arranca el terminal Ta_0, que desea usar, pero que no le pertenece.
2. Como parte de la secuencia de arranque de un terminal, Ta_0 ejecuta el agente SHAD local.

3. El agente SHAD de Ta_0 adquiere el identificador del usuario que está arrancando el terminal. Este nombre lo puede adquirir a través de varios métodos, dependiendo de la implementación.

4. Si el usuario que intenta arrancar Ta_0 no es su dueño (cuyo identificador tiene que tener almacenado en el hardware junto con la clave de terminal), el agente trata de descubrir su agente SHAD principal (Tb^*).

5. El agente de Ta_0 pide autorización al agente que ejecuta en el agente principal del usuario B.

6. El agente principal del usuario B notifica la petición al usuario mediante una interfaz gráfica en la pantalla de Tb^* y espera la confirmación. El usuario B es capaz de detectar si otro usuario está intentando arrancar el terminal a su nombre y de cancelar el arranque.

7. El usuario B confirma el arranque de Ta_0.

8. El agente principal del usuario B pide autorización al al agente principal del usuario A para arrancar el terminal Ta_0. El agente de Tb^* se autentica ante el agente de Ta^* mediante la clave de emparejamiento.

9. Este paso es opcional, y depende de la configuración del agente principal de A. El agente principal del usuario A notifica la petición de autorización en la pantalla de Ta^* y espera una confirmación.

10. Si la cesión se puede autorizar, el agente principal de A envía la autorización al agente principal de B.

11. Si la autorización es correcta, el agente principal de B envía la autorización al agente que ejecuta en el terminal Ta_0.

12. El agente del terminal Ta_0 valida la autorización y arranca el terminal a nombre del usuario B.

13. El agente del terminal Ta_0 establece una sesión con el agente principal del usuario B para adquirir secretos, siguiendo el mismo esquema que en la arquitectura de SSO.

La comunicación entre los agentes principales de Ta^* y Tb^* se basa en la **clave de emparejamiento** de A y B, $K_{a,b}$. Esta clave se encuentra almacenada en el **repositorio de secretos** del usuario A y del usuario B.

La autorización que se le envía a Ta_0 para permitir la cesión se basa en la **clave de terminal** de Ta_0, conocida tanto por el agente local de Ta_0 como por el agente principal que ejecuta en Ta^* (Ta_0 la tiene almacenada en su hardware y Ta^* en el repositorio de secretos).

5.4. Diseño

No es necesario modificar el diseño del agente SHAD presentado en el capítulo 4. Simplemente hay que añadir funcionalidad a los siguientes módulos:

- **Agente principal:** Módulo de descubrimiento (DM)
  - Autenticar agentes principales de usuarios emparejados. Debe poder autenticar tanto los mensajes del usuario como los mensajes de los usuarios con los que se ha emparejado. La autenticación es necesaria para procesar una petición de cesión de un terminal.
  - Solicitar la autorización para arrancar un terminal para obtener la autorización tratada en el punto anterior.
  - Crear datos que certifiquen la autorización, que se cifran mediante la *clave de emparejamiento* de los dos usuarios involucrados.
  - Enviar al usuario notificaciones relacionadas con la cesión de terminales.
  - Recibir del usuario confirmaciones relacionadas con la cesión de terminales.

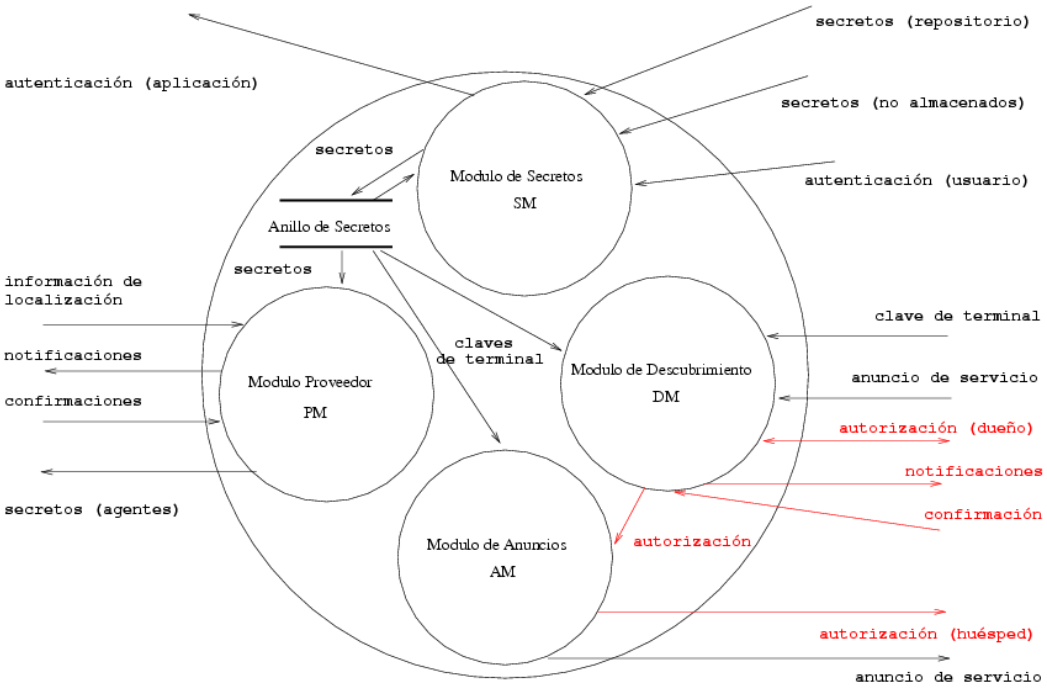

Figura 5.2: *Diagrama de flujo de datos de nivel uno correspondiente a un agente principal para la compartición de terminales.*

- **Agente común:** Módulo de descubrimiento (DM)

  - Detectar que el usuario que intenta arrancar el terminal no es el dueño. No se trata de autenticar al usuario que intenta arrancar el terminal, sino de obtener su identificador.
  - Establecer contacto el agente principal del usuario que arranca el terminal. Esa comunicación es la que desencadena el protocolo de compartición de terminales.
  - Validar la autorización generada por los agentes principales. El DM tiene que ser capaz de validar la autorización generada por el agente principal del dueño del terminal, y de esa forma poder autorizar el arranque del terminal a nombre del usuario huésped.

Las figuras 5.2 y 5.3 muestran los diagramas de flujo de datos de nivel 1 para los agentes SHAD para compartición de terminales (agente principal y agente

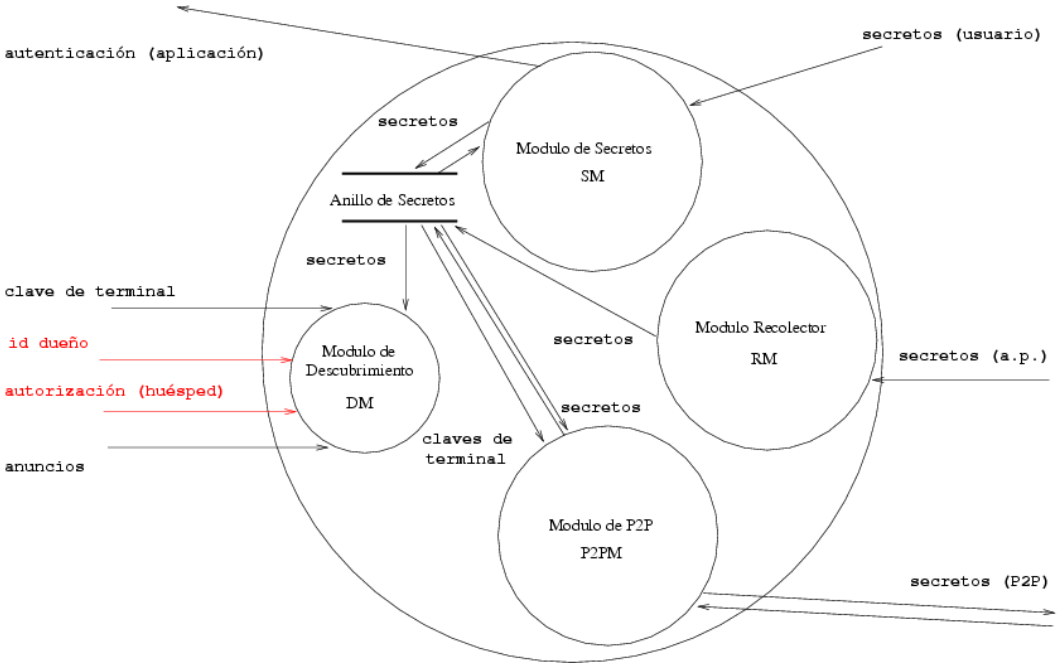

Figura 5.3: *Diagrama de flujo de datos de nivel uno correspondiente a un agente común para la compartición de terminales.*

común respectivamente). En las figuras se representan en color rojo los flujos de datos que se han añadido al diseño del agente SHAD de SSO.

Hay dos flujos de datos básicos: la autorización del dueño del terminal para que el usuario huésped pueda arrancarlo a su nombre, y la autorización del huésped que intenta arrancar el terminal. Los otros flujos corresponden a las notificaciones y confirmaciones que realizan los dos usuarios involucrados durante el proceso de cesión del terminal.

5.5. ¿SSO para los terminales cedidos?

Cuando se cede un terminal, el usuario huésped puede usarlo como un terminal de su propiedad por un tiempo indeterminado.

El uso de hardware que no nos pertenece, en el que además se ejecuta un sistema operativo y otro tipo de software sobre el que no tenemos control, supone

un gran riesgo. Si un adversario tiene control sobre la máquina cedida, entonces tiene la capacidad de controlar y espiar el uso de cualquier dispositivo (p.e. el teclado), volcar el contenido de la memoria principal de cualquier programa que ejecuta en su máquina (el agente SHAD incluido), etc. De esta forma, cualquier secreto que se teclee en la máquina cedida puede quedar comprometido. Cualquier operación realizada puede resultar incorrecta.

Por tanto, **cualquier secreto del usuario que acabe en una máquina controlada por un adversario queda comprometido en el acto**, indistintamente del método que se utilice para hacer llegar el secreto allí (mecanismos de SSO, introducción manual de contraseñas, etc.).

Entonces, ¿Es razonable que un usuario acceda al servicio de SSO desde una máquina que no le pertenece? La arquitectura podría adoptar distintas estrategias para abordar esta cuestión:

- No permitir el uso del servicio de SSO en la máquina cedida. El usuario debería proporcionar los secretos manualmente cuando fuera necesario.
- Permitir el acceso total al servicio de SSO como si el terminal cedido fuera un terminal más del usuario. Este enfoque supone la confianza total en el hardware y el software del terminal cedido.
- Permitir un acceso al servicio de SSO limitado, de tal forma que el usuario pueda especificar los secretos que están disponibles desde máquinas que no le pertenecen y en las que no confía plenamente.

La primera estrategia es muy restrictiva y añade demasiada obstrucción al sistema. El usuario debe introducir al menos una contraseña en el terminal cedido, ya que debe acceder al sistema de ficheros en el que se encuentran sus datos y su configuración (que normalmente requiere autenticación).

La segunda estrategia es demasiado laxa, teniendo en cuenta que cualquier secreto que tenga restricciones que permitan su envío a un agente común puede acabar en la máquina cedida. También implica que un terminal cedido puede

activar el protocolo P2P de SSO para conseguir secretos directamente de los agentes comunes y no del agente principal. La estrategia no parece razonable porque ofrece el mismo tipo de servicio a una máquina cedida que a una máquina propia, cuando la primera es implícitamente menos confiable que la segunda.

La tercera estrategia es un compromiso entre las dos anteriores. Una vez arrancado el terminal a nombre del usuario huésped, se permite un uso restringido del servicio de SSO. A través del servicio de SSO el agente de la máquina cedida puede acceder a un subconjunto de los secretos del agente principal.

Para ello, los secretos deben incluir nuevas restricciones que controlen el acceso a los secretos desde terminales que no son propiedad del usuario.

Por omisión, los secretos del usuario no deben ser accesibles desde un terminal que no le pertenezca. Si el usuario desea que un secreto esté disponible desde un terminal que le han cedido, debe especificarlo explícitamente en el repositorio de secretos mediante un atributo.

Esta estrategia es la más flexible, ya que en la práctica puede actuar como las dos anteriores, con la excepción del uso del protocolo P2P de SSO. Mediante la configuración del repositorio de secretos, se puede restringir o permitir el acceso a todos los secretos del usuario.

El uso del protocolo P2P de SSO no parece razonable en ningún caso. No creemos que sea prudente activar los mecanismos para obtener secretos directamente de los agentes SHAD comunes cuando el usuario no está presente en el sistema para poder cancelar ciertas operaciones. Si el usuario ya no se encuentra delante del terminal cedido, entonces no debería ser necesario enviar nuevos secretos a dicho terminal. En el caso de que el usuario haya dejado aplicaciones ejecutando en el terminal cedido, esas aplicaciones ya no son fiables. Por estas razones, el agente común que ejecuta en un terminal cedido no puede activar el protocolo P2P de SSO.

La arquitectura SHAD para la compartición de terminales parte de la siguiente hipótesis: **los usuarios emparejados actúan de buena fe, y por lo**

tanto, no pueden adoptar un comportamiento malicioso o negligente.

Es razonable considerar que los usuarios emparejados hacen buen uso del sistema y no pueden convertirse en adversarios. **No es posible compartir recursos de forma segura cuando no se confía ni en ellos ni en sus dueños**.

Si no se acepta esta hipótesis, es inútil intentar asegurar el uso de cualquier recurso que no pertenezca al usuario, dado que el recurso en sí puede ser malicioso.

También es razonable considerar que los usuarios no van a actuar de forma negligente y van a prestar atención al dispositivo que alberga todos sus secretos y ofrece los mecanismos para autenticarse ante los demás usuarios.

Como consecuencia de la hipótesis:

- La clave de emparejamiento sólo la conocen los UbiTerms de ambos usuarios.
- La pérdida de un UbiTerm se comunica inmediatamente para que los demás usuarios rechacen las operaciones que se hagan en nombre del usuario comprometido y se revoquen todas sus claves de emparejamiento.
- Se supone que el hardware que se cede a los usuarios es correcto y no ha sido modificado con intenciones maliciosas.
- El núcleo del sistema operativo y el agente SHAD que ejecuta tanto en el hardware cedido como en el UbiTerm de los usuarios es correcto, y no ha sido modificado con intenciones maliciosas.

Por lo tanto, el usuario que usa un terminal que no le pertenece confía plenamente en el software y el hardware que usa. Por lo contrario, no confía en las redes de comunicaciones mediante las cuales se opera en el sistema ni en los usuarios con los que no está emparejado.

En la siguiente sección se describe el protocolo que hemos diseñado para la arquitectura SHAD para la compartición de terminales. No obstante, hemos

diseñado otro protocolo para proteger los secretos del usuario cuando el UbiTerm del dueño del terminal o la clave de emparejamiento quedan comprometidos. Este protocolo se describe en la sección 5.8.

5.6. Protocolo

Este protocolo corresponde a los siguientes flujos de datos presentados en los diagramas de flujo de datos correspondientes al diseño del agente:

- `autorización (dueño)` entre las entidades *módulo de descubrimiento* de dos agentes principales de diferente dueño. El flujo de datos representa la autorización del dueño del terminal para la cesión.

- `autorización (huésped)` entre las entidades *módulo de anuncios* del agente principal y el *módulo de descubrimiento* del agente común. El flujo de datos representa la autorización del usuario para que el agente del terminal pueda arrancar su nombre. Incluye autorización del dueño del terminal y los datos necesarios para establecer la sesión con el agente principal, con el fin de usar el protocolo de SSO.

El protocolo se inicia siempre que arranca un agente de SHAD en nombre de un usuario que no es el dueño del terminal.

En el cuadro Protocolo 4 se detalla el protocolo de compartición de terminales, suponiendo el esquema mostrado en la figura 5.1: el usuario B intenta arrancar un terminal Ta_0 que pertenece al usuario A. Para ello, los agentes principales de A y B, Ta^* y Tb^* respectivamente, necesitan autenticarse mutuamente y autorizar la operación.

El protocolo de cesión de terminales es una extensión del protocolo de descubrimiento de agente principal descrito en la sección 4.5.2. El protocolo se compone de cuatro mensajes:

- `ineedmainagent` (INMA): Cuando el agente Ta_0 solicita arrancar en nombre del usuario B, el agente inicia la búsqueda del agente principal de B.

  Para ello radia un mensaje `ineegmainagent` similar al protocolo de descubrimiento del agente principal de SSO, excepto por la cabecera. El primer campo de la cabecera indica el tipo de mensaje, el segundo indica el dueño del terminal, el tercero indica el nombre del usuario que intenta usarlo, y el cuarto indica el nombre del terminal.

  El agente de Tb^* recibe el mensaje, pero no conoce la clave de terminal de Ta_0, KT_{Ta_0}. Examinando la cabecera del mensaje, que no se encuentra cifrada, puede deducir que el usuario B está intentando arrancar un terminal que pertenece al usuario A.

  Si el usuario B no se encuentra emparejado con el usuario A (no se encuentra la clave de emparejamiento $K_{a,b}$) se ignora el mensaje INMA y el protocolo no prospera.

  Antes de continuar con el protocolo, **el usuario B debe confirmar la operación en *Tb**** . Esta confirmación es obligatoria. Si la confirmación no tiene lugar en un tiempo determinado, se desecha el mensaje sin enviar ninguna respuesta a Ta_0.

- `ineedowner` (INO): El agente principal del usuario B pide la autorización al agente principal del dueño del terminal mediante un mensaje `ineedowner`.

  La petición de autorización se basa en $K_{a,b}$, por tanto Ta^* es capaz de autenticar el mensaje. Este mensaje lleva ensamblado el mensaje cifrado correspondiente al mensaje `ineedmainagent`, para que Ta^* pueda validarlo.

- `iamowner` (IAO): Ta^* autentica el mensaje `ineedowner` mediante $K_{a,b}$. También autentica el mensaje `ineedmainagent` reensamblado mediante

KT_{Ta_0}, Si la configuración permite la cesión, Tb^* genera un mensaje `iamowner`, que consta de dos partes:

- Una parte que contiene la clave de sesión KS_{Ta_0} y que autoriza la cesión del terminal. Dicha parte se encuentra cifrada con $K_{a,b}$, por tanto Tb^* puede acceder a su contenido y a la vez constatar que procede de Ta^*.

- Una parte que contiene datos para el terminal Ta_0 cifrados mediante la clave de terminal KT_{Ta_0}. Por tanto, esta parte no es accesible para Tb^*. Esta parte se debe ensamblar en el mensaje `iammainagent` descrito a continuación. Esta parte del mensaje se denomina `fromowner`.

- `iammainagent` (IAMA): El mensaje está compuesto por dos partes:

  - La parte `fromowner` que se ensambló en el mensaje. Contiene la clave de sesión generada por Ta^* para la cesión del terminal, KS_{Ta_0}. Permite comprobar que Ta^* autorizó la cesión al usuario B. También permite comprobar que la clave de sesión ha sido generada por Ta^*.

  - El contenido del mensaje `iammainagent` que ha generado Tb^* y que contiene la dirección del agente principal que debe usar el agente que arranca en el terminal. Esta parte del mensaje se descifra mediante la clave de sesión que se ha asignado al terminal, KS_{Ta_0}, que se ha obtenido de la parte `fromowner`. Permite comprobar que Tb^* conoce KT_{Ta_0}, por tanto corrobora que Ta^* autorizó la cesión a Tb^*.

Protocolo 4 Protocolo para la compartición de terminales.

$REQ = [\ \texttt{ineedmainagent}, uname_b, Ta_0, N_1, N_2, tstamp_{Ta_0}]$

$Ta_0 \longrightarrow BROADCAST$

$\texttt{ineedmainagent}, uname_b, uname_a, Ta_0$

$IV_1, \{ranb_1, \{REQ\}_{SHA1}, REQ\}_{KT_{Ta_0}}$

Confirmación explícita obligatoria por parte de B

$AUTREQ = [\texttt{ineedowner}, uname_b, Ta_0, tstamp_{Tb^*}, N_3]$

$Tb^* \longrightarrow Ta^*$

$\texttt{ineedowner}, uname_b$

$IV_2, \{ranb_2, \{AUTREQ\}_{SHA1}, AUTREQ\}_{K_{a,b}}$

$IV_3, \{ranb_3, \{REQ\}_{SHA1}, REQ\}_{KT_{Ta_0}}$

$AUTHRES = [\texttt{iamowner}, uname_b, Ta_0, KS_{Ta_0}, tstamp_{Ta^*}, N_3 + 1, N_1 + 1]$

$CAP = [\texttt{fromowner}, uname_b, Ta_0, N_2 + 1, tstamp_{Ta^*}, KS_{Ta_0}]$

$Tb^* \longleftarrow Ta^*$

$\texttt{iamowner}$

$IV_4, \{ranb_4, \{AUTHRES\}_{SHA1}, AUTHRES\}_{K_{a,b}}$

$IV_5, \{ranb_5, \{CAP\}_{SHA1}, CAP\}_{KT_{Ta_0}}$

$RES = [\texttt{iammainagent}, uname_b, Ta_0, address, N_1 + 1, tstamp_{Ta^*}]$

$Ta_0 \longleftarrow Tb^*$

$\texttt{iammainagent}$

$IV_6, \{ranb_6, \{RES\}_{SHA1}, RES\}_{KS_{Ta_0}}$

$IV_7, \{ranb_7, \{CAP\}_{SHA1}, CAP\}_{KT_{Ta_0}}$

En el protocolo se utilizan los siguientes *nonces*:

- N_1: Lo crea Ta_0 y lo incrementa Tb^*. Ta_0 lo utiliza para comprobar que la primera parte del mensaje `iammainagent` corresponde a la petición `ineedmainagent` que se envió y no a otra petición anterior del mismo tipo.

- N_2: Lo crea Ta_0 y lo incrementa Ta^*. Ta_0 lo utiliza para comprobar que la segunda parte del mensaje `iammainagent`, `fromowner`, corresponde a la petición `ineedmainagent` que se envió y no a otra petición anterior del mismo tipo.

- N_3: Lo crea Tb^* y lo incrementa Ta^*. Tb^* lo utiliza para comprobar que la primera parte del mensaje `iamowner` corresponde a la petición `ineedowner` que se envió y no a otra petición anterior del mismo tipo.

El protocolo aplica las misma medidas de protección basadas en la *cache de nonces* y marcas de tiempo en los mensajes.

5.6.1. Encaminamiento de los mensajes del protocolo

El protocolo consta de cuatro mensajes que involucran a los UbiTerms de los dos usuarios (el dueño del terminal y el usuario que desea utilizarlo) y al propio terminal. En el protocolo descrito en la sección anterior, el terminal cedido se pone en contacto con el UbiTerm del usuario huésped. Después, el UbiTerm del usuario se pone en contacto con el UbiTerm del dueño del terminal.

El protocolo podría haberse diseñado para que el terminal cedido se pusiera en contacto con el UbiTerm de su dueño en primera instancia. La figura 5.4 muestra las dos opciones: la figura situada a la izquierda (1) muestra el diseño del protocolo actual, y la figura situada a la derecha (2) muestra el diseño alternativo que se podría haber seguido para el protocolo.

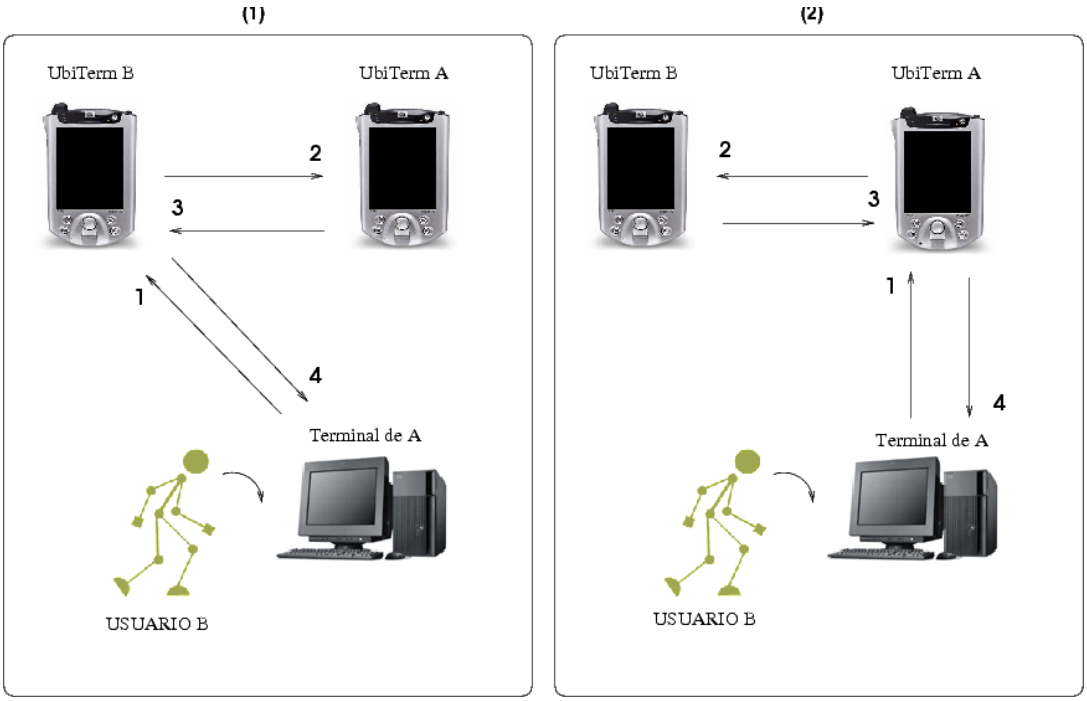

Figura 5.4: *Los dos esquemas posibles para el protocolo de compartición de terminales. La figura (1) ilustra el esquema que se ha seguido para el protocolo de SHAD. La figura (2) ilustra la otra alternativa.*

La razón por la que se ha optado por el encaminamiento de la figura 5.4(1) es que el protocolo se puede abortar en una etapa más temprana en caso de que un adversario suplante a un usuario para arrancar un terminal.

Cuando el terminal inicia el protocolo, sólo conoce el identificador del usuario que supuestamente quiere arrancarlo. En ese momento, el terminal no tiene constancia de que el identificador que le han suministrado para arrancar el protocolo sea correcto.

Siguiendo el esquema propuesto en la figura 5.4(1), el siguiente paso en el protocolo corresponde al usuario que se supone está arrancado el terminal. El usuario debe confirmar en su UbiTerm que realmente está arrancado el terminal que ha iniciado el protocolo. Por lo tanto, si el terminal intenta arrancar en nombre de un usuario que realmente no está arrancando el terminal, el protocolo termina tras el primer mensaje.

En caso de que el adversario intente suplantar también el UbiTerm del usuario suplantado, el protocolo tampoco prosperaría, debido a que el adversario no posee la clave de emparejamiento entre el dueño del terminal y el usuario suplantado.

Ahora supongamos el esquema mostrado en la figura 5.4(2). Si el terminal intenta arrancar en nombre de un usuario que no es el que intenta usarlo, el protocolo prosperaría hasta el segundo mensaje, ya que el usuario que se supone está arrancado el terminal no rechazaría la acción hasta ese momento.

En este caso, se aumenta la efectividad de ataques de denegación de servicio cuyo objetivo sea agotar la batería de los UbiTerms (en este caso, el UbiTerm del dueño del terminal). El UbiTerm del dueño del terminal tendría que recibir el mensaje 1, crear el mensaje 2 (con el gasto de procesamiento que implica el cifrado simétrico) y enviarlo. Aunque la arquitectura siempre estará sujeta a este tipo de ataques, es conveniente minimizar la efectividad de los mismos.

El dueño del terminal no puede abortar el protocolo después del primer mensaje, porque probablemente no sepa quién está arrancado su terminal. Aunque lo supiera, no se puede obligar al dueño del terminal a confirmar cada cesión de un terminal de su pertenencia. Si obligamos al dueño a confirmar todas las cesiones, incrementamos su nivel de obstrucción aun cuando él no es el interesado en la operación. El usuario que quiere usar el terminal es el interesado en la operación, y como tal, estará dispuesto a confirmar la operación en su UbiTerm. Sin embargo, el dueño del terminal no tiene porqué estar interesado en confirmar la cesión de una de sus máquinas. SHAD ofrece mecanismos para realizarla, pero no de forma obligatoria.

Los dos encaminamientos tienen problemas si no se acepta la hipótesis descrita en la sección 5.5: si el UbiTerm del dueño del terminal actúa de forma maliciosa, entonces puede capturar todos los secretos que se transmitan entre Tb^* y Ta_0.

5.6.2. Revocación de un terminal

El dueño de un terminal tiene que ser capaz de revocar desde su UbiTerm el uso del terminal cedido en cualquier momento. Para ello, hemos creado un nuevo mensaje: `revoqueterminal` (RT). Este mensaje se envía desde el UbiTerm al terminal cedido y no requiere de respuesta.

El mensaje se cifra con la clave de terminal KT_{Ta_0}, que sólo conocen el UbiTerm Ta^* y el terminal Ta_0.

El cuadro Protocolo 5 muestra el mensaje completo.

Protocolo 5 Mensaje de revocación de un terminal

$REQ = [\texttt{revoqueterminal}, Ta_0, N_4, tstamp_{Ta^*}, Timeout]$

$Ta^* \longrightarrow Ta_0$
$\texttt{revoqueterminal}, Ta_0$
$IV_1, \{ranb_1, \{REQ\}_{SHA1}, REQ\}_{KT_{Ta}}$

El campo *Timeout* especifica el tiempo que debe transcurrir desde que el agente del terminal cedido recibe el mensaje RT y el reinicio del terminal.

Cuando el agente SHAD del terminal cedido recibe un mensaje RT y valida su contenido, presenta un mensaje por pantalla avisando de la revocación. El aviso que aparece por pantalla comunica el tiempo que le queda al usuario huésped para cancelar las tareas que está realizando y salvar sus documentos. Después del tiempo especificado, el agente reinicia el terminal

Si *Timeout* tiene el valor 0, el terminal reiniciará al recibir el mensaje, por tanto sin aviso previo.

El mensaje incluye un *nonce* N_4 y una marca de tiempo $tstamp_{Ta^*}$ para que el agente SHAD de Ta_0 pueda comprobar que no es un mensaje antiguo reinyectado. Para ello, se sigue la misma táctica descrita para los mensajes anteriores (*cache de nonces*).

5.7. Implementación de prototipo

La implementación del prototipo se basa en NetFactotum, al que se añaden las nuevas funcionalidades y los protocolos descritos en las secciones anteriores.

5.7.1. Claves para el protocolo SHAD

Clave de emparejamiento

Para representar el secreto $K_{a,b}$ dentro del repositorio de secretos, se añade una tupla como la que sigue:

`proto=shad type=ubiterm propietor=`*Usuario* `!secret=`$K_{a,b}$

Las tuplas que representan secretos compartidos entre usuarios son del tipo `ubiterm`. El atributo `propietor` identifica al usuario con el que nos hemos emparejado. Por último, el atributo `secret` almacena la clave de emparejamiento, que es una clave AES de 256 bits.

En la figura 5.5 se muestra el esquema entre tres usuarios A, B y C[1]. A tiene emparejado su UbiTerm con B y con C. Sin embargo, B y C sólo tienen emparejados su UbiTerm con A. Por tanto, existe confianza mutua entre A y B, y entre A y C. No obstante, C no confía en B y viceversa.

Clave de terminal

Se han añadido nuevos atributos a las claves de terminal de SHAD con el fin de permitir al usuario fijar las políticas de cesión de sus terminales. Para un terminal Ta_0, la representación de la clave de terminal posee los siguientes atributos:

`proto=shad type=term machine=`Ta_0 `handover needconfirm ownerinlocation userinlocation voicenotification !tsecret=`KT_{Ta_0}

[1]En la figura, la clave es de menos longitud que la utilizada realmente en el sistema.

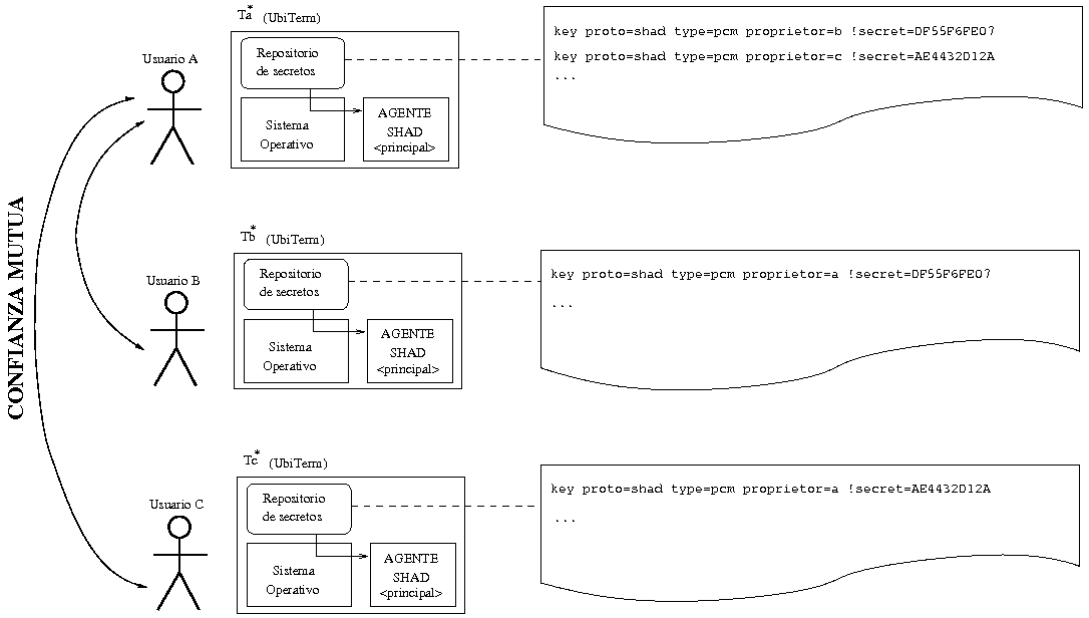

Figura 5.5: *El usuario A tiene confianza mutua con los usuarios B y C. Sin embargo, B y C no confían entre sí.*

Se han añadido cinco nuevos atributos a las claves de terminal explicadas en la Sección 4.6.1.

Por omisión, un terminal no se puede ceder a otros usuarios. Para indicar que el terminal sí puede cederse, hay que incluir el atributo `handover` en la tupla.

El atributo `needconfirm` indica que es necesario que el dueño de la máquina confirme la cesión en su UbiTerm. Siguiendo el mismo escenario que en secciones anteriores, el usuario A debería confirmar en Ta^* la cesión del terminal Ta_0 al usuario B. Por defecto, la confirmación no es necesaria.

El atributo `ownerinlocation` hace que sólo se requiera la confirmación si el dueño del terminal y el usuario que intenta usarlo se encuentran en distintas localizaciones del espacio inteligente. Este atributo depende de la infraestructura de contexto del espacio inteligente. De la misma forma que en la arquitectura de SSO, en caso de que la infraestructura de contexto no esté disponible y haya que evaluar un atributo que depende de ella, siempre se ejecuta la acción más restrictiva. Si no se puede acceder a la infraestructura de contexto, el dueño

debe confirmar la cesión.

El atributo `userinlocation` hace que el terminal sólo se pueda ceder si el usuario que lo intenta usar se encuentra en la misma localización que el terminal. Si la infraestructura de contexto posee información de localización de todos los usuarios del espacio inteligente, este atributo no tiene sentido, ya que un usuario no podría arrancar un terminal sin estar en la misma en sala . Pero en la práctica cabe la posibilidad de que la infraestructura de contexto no posea información acerca de todos los usuarios del espacio inteligente. Puede haber usuarios que no dispongan de hardware para que la infraestructura les monitorice, usuarios a los que se les olvide activar las balizas que sirven para obtener su localización, usuarios que valoren negativamente la monitorización porque consideran que viola su privacidad, etc. Por lo tanto, aunque la infraestructura de contexto se encuentre operativa, puede haber usuarios sobre los que no se obtenga información de contexto.

Por tanto, el atributo `userinlocation` restringe el uso del terminal a los usuarios que tienen disponible información de localización que indica que se encuentran en la misma sala que el terminal. En caso de que la información de localización no esté disponible, se deniega el acceso.

Por último, el atributo `voicenotification` indica que siempre que se autorice la cesión de un terminal, se notifique la operación a través del sistema de voz del espacio inteligente.

5.7.2. Secretos comunes

Los secretos comunes que se sirven a través del protocolo de SSO necesitan un nuevo atributo que especifique si se pueden enviar a un terminal cedido.

El nuevo atributo se llama `alienaccess`. Si el atributo está presente en un secreto, entonces el agente principal puede enviar ese secreto a un terminal cedido. En caso de ausencia del atributo, el secreto no se podrá enviar a un terminal cedido.

5.7.3. Control del agente

Hemos añadido un fichero de control adicional con el fin de ofrecer una interfaz para la revocación de los terminales cedidos. El nombre del fichero es `revoque` y recibe una cadena de control que especifica el terminal que se desea revocar y el tiempo en segundos deseado para llevar a cabo la revocación.

Por ejemplo, supongamos que el usuario desea revocar el uso del terminal con nombre `rusty` en 120 segundos:

```
echo rusty 120 >/mnt/netfactotum/revoque
```

Si el valor del segundo campo de la cadena de control es `0`, el terminal se reinicia en cuanto recibe el mensaje `revoqueterminal`.

5.7.4. Identificación del usuario

Los terminales tienen almacenado el nombre de su dueño junto con la clave de terminal en la NVRAM. Cuando un usuario desea utilizar un terminal que encuentra en el espacio inteligente, necesita identificarse [2] para que el agente SHAD pueda iniciar el protocolo de descubrimiento de agente principal al comprobar que alguien que no es el dueño intenta usarlo.

El método más sencillo es la identificación manual, que consiste en teclear el identificador de usuario en el terminal. Dicha acción añade obstrucción al uso del sistema.

Existen otros métodos que provocan menos obstrucción. Podemos sacar partido de la infraestructura de contexto que nos suministra información de localización. Cuando alguien arranca el terminal, su agente SHAD puede obtener el identificador del usuario que está más cercano al terminal.

La infraestructura de contexto sobre la que se ha implementado el prototipo ofrece información de localización poco precisa, ya que sólo informa de la sala en la que se encuentra el usuario. Cuando el agente SHAD comprueba que en la sala

[2] Nótese que se trata de identificación, no de autenticación. El usuario se autentica mediante su UbiTerm en el transcurso del protocolo de descubrimiento y cesión.

donde se encuentra el terminal sólo hay un usuario del sistema, entonces supone que el identificador del usuario coincide con el que le suministra la infraestructura de contexto.

En ese caso, el usuario no tiene que suministrar su identificador para arrancar el terminal a su nombre. Sin embargo, si hay más de una persona en la sala, el agente no puede deducir el identificador del usuario que está arrancado el terminal. En ese caso, se requiere la identificación manual del usuario.

En espacios inteligentes que dispongan de información de localización más precisa, simplemente se debería comprobar quién es el usuario más cercano al terminal. En todo caso, el problema persiste pero con distinta granularidad. En esta situación habría incertidumbre sobre el identificador del usuario en el caso de que hubiera más de un usuario delante del terminal. En ese caso, también se requeriría la identificación manual.

El problema se puede solucionar mediante el uso de RFIDs pasivas de contacto. Este tipo de RFIDs necesitan mucha proximidad (casi contacto) con su lector. Por tanto, si el usuario lleva consigo una RFID de este tipo, por ejemplo pegada en su reloj de pulsera, bastaría con apoyar el reloj sobre el lector de RFIDs (que podría estar al lado del teclado) para identificarse ante el terminal.

5.7.5. Arranque del terminal

NetFactotum se ha implementado para el sistema operativo Plan B. En Plan B, el usuario que arranca la máquina es el dueño del terminal hasta que vuelva a reiniciar.

Por tanto, el agente NetFactotum lo único que debe hacer es impedir el arranque en caso de que la cesión no se haya autorizado. En caso de que sí se haya autorizado, el terminal arranca de forma habitual pero a nombre del usuario autorizado.

Este hecho no supone ningún riesgo, ya que normalmente los terminales de Plan B no poseen un sistema de ficheros local. En su lugar, se monta un servidor de ficheros remoto.

El usuario que usa el terminal cedido no se puede autenticar como el verdadero dueño del terminal ante el servidor de ficheros, ya que no conoce la contraseña.

Por tanto, aunque un usuario utilice el terminal cedido, no puede acceder a los ficheros del dueño del terminal. Eso no es así si el terminal posee un sistema de ficheros en el disco local. Queda en manos del usuario la configuración de NetFactotum compartir terminales con sistemas de ficheros locales que no requieran autenticación.

Después de que NetFactotum permita continuar con el proceso de arranque de Plan B, el sistema necesita montar un sistema de ficheros raíz.

Para ello, necesita autenticarse ante el servidor de ficheros que lo sirve. El agente NetFactotum local obtiene ese secreto del agente principal del usuario a través del servicio de SSO.

Una vez autenticado en el servidor de ficheros, el terminal finaliza el proceso de arranque. El usuario puede usar el terminal como cualquiera de sus máquinas: puede importar/exportar el ratón y el teclado, montar los volúmenes que sus otras máquinas exportan, compartir la pantalla del terminal, redireccionar la entrada/salida de los comandos, acceder a servicios comunes, etc.

La implementación de la arquitectura para otro sistema debería tener en consideración el acceso al sistema de ficheros local, la creación de cuentas invitadas, los detalles del arranque y la configuración del entorno para el usuario transitorio.

5.7.6. Evaluación y experiencia de uso

El prototipo NetFactotum se ha usado en el Laboratorio de Sistemas como prueba de concepto, obteniendo un resultado positivo en lo que respecta al protocolo y confirmaciones por parte de los usuarios.

Sin embargo, la arquitectura de compartición de terminales no ha resultado tan útil en nuestro espacio inteligente como se había pensado antes de su desarrollo. En comparación, la arquitectura de SSO ha resultado mucho más

útil en nuestro entorno que la arquitectura de compartición de terminales.

Creemos que este hecho se debe a dos peculiaridades del espacio inteligente en el que se ha probado:

- El número de usuarios que conforman el espacio es muy reducido.
- Los usuarios poseen una gran cantidad de terminales, tanto fijos como móviles. Por tanto, los usuarios no suelen tener la necesidad de tomar prestado un terminal de otros. Esta situación es habitual en la actualidad, debido al constante abaratamiento del hardware.
- El espacio inteligente en el que se ha probado consta de pocas salas comunitarias, que además no suelen usarse regularmente. Precisamente, en estas localizaciones es donde los terminales compartidos suelen tener más relevancia.

No obstante, pensamos que la arquitectura de compartición de terminales puede resultar útil en espacios inteligentes con características diferentes a las del espacio en el que se ha implementado y probado el prototipo.

5.8. Protocolo alternativo para la compartición de terminales

Anteriormente se ha descrito un protocolo para la compartición de terminales que supone que los UbiTerms de los usuarios involucrados en la cesión (el dueño del terminal y el usuario huésped que pretende usarlo) actúan correctamente. Por lo tanto, ese protocolo se basa en que la clave de emparejamiento entre los usuarios involucrados sólo la conocen los UbiTerms involucrados.

En el caso de el UbiTerm del propietario del terminal actúe de forma maliciosa, o la clave de emparejamiento entre los dos usuarios sea conocida por un adversario, todos los secretos que se compartan entre el UbiTerm del huésped y el terminal cedido pueden quedar expuestos.

A continuación presentamos un nuevo protocolo de compartición de terminales que asegura los secretos transferidos al agente SHAD común que ejecuta en un terminal cedido en estas situaciones.

Las hipótesis sobre las que se construye el protocolo son:

- El terminal cedido es correcto: tanto su software como su hardware no se han manipulado y su comportamiento no es malicioso.
- El UbiTerm del usuario huésped también es correcto.

El protocolo pretende asegurar los secretos del usuario huésped cuando utiliza el servicio de SSO desde un terminal cedido. El adversario puede denegar el servicio u obstaculizarlo, pero en ningún caso puede capturar los secretos que se envían a través del servicio de SSO.

5.8.1. Uso de criptografía asimétrica

Supongamos el esquema seguido en la figura 5.1 (Sección 5.3), en el que un usuario B pretende utilizar un terminal que pertenece al usuario A.

Dado que suponemos que el agente SHAD principal de A (Ta^*) puede ser malicioso, hay que establecer algún mecanismo para asegurar la comunicación entre el agente principal de B (Tb^*) y el terminal que se quiere utilizar (Ta_0), El principal problema es que Tb^* y Ta_0 no comparten ningún secreto entre ellos:

- Ta_0 y Ta^* comparten la clave de terminal KT_{Ta_0}.
- Ta^* y Tb^* comparten la clave de emparejamiento $K_{a,b}$.

Por tanto, usando las claves disponibles en el esquema de SHAD no se puede establecer una conexión segura basada en criptografía simétrica entre Tb^* y Ta_0 sin que Ta^* sea capaz de interceptarla.

Por otra parte, incluir nuevas claves simétricas para asegurar Tb^* y Ta_0 no es viable, ni aporta una solución al problema. No es viable porque el número

de claves a almacenar en el repositorio de secretos crece exponencialmente al insertar un secreto compartido con cada terminal de los usuarios en los que se confía. En todo caso, no aporta seguridad si dicha clave se encuentra en el repositorio de secretos de A, ya que suponemos que Ta^* se comporta de forma maliciosa.

Mediante el uso de criptografía de clave asimétrica podemos construir un protocolo que asegure confidencialidad a la comunicación entre Tb^* y Ta_0.

Cuando el usuario B arranca el terminal Ta_0, este genera un par de claves de criptografía asimétrica. Nos referiremos a la clave pública como PUK_{Ta_0} y a la clave privada como PRK_{Ta_0}. Dichas claves deben generarse en tiempo de arranque, y en ningún caso deben almacenarse de forma persistente en Ta_0.

El tiempo requerido para generar un par de claves de criptografía asimétrica es lo suficientemente corto como para no obstaculizar significativamente el arranque del terminal cedido. Hemos medido el tiempo medio para generar un par de claves RSA de 1024 bits de longitud en un terminal de Plan B con procesador Pentium 4 a 3 Ghz y 1 Gb de memoria principal. El tiempo medio obtenido a partir de la generación de 50 claves es de 1,17 segundos.

En caso de que el terminal cedido fuera un dispositivo móvil cuya alimentación sólo depende de una batería, la generación de claves supondría un problema. Sin embargo, los terminales compartidos suelen ser estaciones de trabajo o PCs sencillos (Mini-TX) que se encuentran dispersos en el entorno inteligente. Por tanto, el tiempo de procesamiento (y el consecuente consumo de energía) que provoca la generación de las claves no tiene impacto sobre su tiempo de funcionamiento.

5.8.2. Protocolo

El protocolo de compartición de terminales se ejecuta cada vez que un usuario intenta arrancar un terminal que no le pertenece.

Protocolo 6 Protocolo alternativo para la compartición de terminales.

$REQ = [\texttt{ineedmainagent}, uname_b, Ta_0, N_1, N_2, tstamp_{Ta_0}]$

$Ta_0 \longrightarrow BROADCAST$
$\texttt{ineedmainagent}, uname_b, uname_a, Ta_0$
PUK_{Ta_0}
$IV_1, \{ranb_1, \{REQ\}_{SHA1}, REQ\}_{KT_{Ta_0}}$

Confirmación explícita obligatoria por parte de B

$AUTREQ = [\texttt{ineedowner}, uname_b, Ta_0, tstamp_{Tb^*}, N_3]$

$Tb^* \longrightarrow Ta^*$
$\texttt{ineedowner}, uname_b$
$IV_2, \{ranb_2, \{AUTREQ\}_{SHA1}, AUTREQ\}_{K_{a,b}}$
$IV_3, \{ranb_3, \{REQ\}_{SHA1}, REQ\}_{KT_{Ta_0}}$

$AUTHRES = [\texttt{iamowner}, uname_b, Ta_0, tstamp_{Ta^*}, N_3 + 1, N_1 + 1]$
$CAP = [\texttt{fromowner}, uname_b, Ta_0, N_2 + 1, tstamp_{Ta^*}]$

$Tb^* \longleftarrow Ta^*$
$\texttt{iamowner}$
$IV_4, \{ranb_4, \{AUTHRES\}_{SHA1}, AUTHRES\}_{K_{a,b}}$
$IV_5, \{ranb_5, \{CAP\}_{SHA1}, CAP\}_{KT_{Ta_0}}$

$RES = [\texttt{iammainagent}, uname_b, Ta_0, N_1 + 1, tstamp_{Ta^*}, address, KS_{Ta_0}]$

$Ta_0 \longleftarrow Tb^*$
$\texttt{iammainagent}$
$IV_6, \{ranb_6, \{RES\}_{SHA1}, RES\}_{PUK_{Ta_0}}$
$IV_7, \{ranb_7, \{CAP\}_{SHA1}, CAP\}_{KT_{Ta_0}}$

El protocolo se describe en el cuadro Protocolo 6 y se compone de cuatro mensajes similares a los descritos en el capítulo 5: `ineedmainagent` (INMA), `ineedowner` (INO), `iamowner` (IAO) y `iammainagent` (IAMA).

El par de claves generado (PUK_{Ta_0} y PRK_{Ta_0}) se utiliza para asegurar la transmisión de la clave de sesión KS_{Ta_0}. El mensaje de respuesta `iammainagent` que Tb^* envía a Ta_0 se cifra mediante la clave pública PUK_{Ta_0}, de tal forma que sólo Ta_0 pueda descifrarla usando PRK_{Ta_0}. Una vez descifrado el mensaje, Ta_0 puede pedir secretos a Tb^* usando el servicio de SSO.

Los *nonces* que se utilizan proporcionan la mismas comprobaciones que en el protocolo original (descrito en la sección 5.6).

5.8.3. Ataque de *hombre en el medio*

El protocolo anterior tiene un problema importante, derivado de la falta de un secreto compartido entre Tb^* y Ta_0 previo al inicio del protocolo de compartición de terminales. El mensaje `ineedmainagent` incluye la clave pública que Ta_0 ha generado en tiempo de arranque (PUK_{Ta_0}) con el fin de que posteriormente Tb^* le comunique de una forma segura la clave simétrica de sesión (KS_{Ta_0}).

Pero, ¿Cómo puede Tb^* comprobar que la clave pública que ha recibido en el mensaje es realmente la que ha generado Ta_0? Todo protocolo de intercambio anónimo de claves está sujeto a un ataque de *hombre en el medio*, debido a la falta de una autenticación previa. Este problema ha sido ampliamente tratado en la literatura. Sin compartir un secreto previo con Ta_0, Tb^* no puede autenticar el mensaje y constatar que la clave recibida pertenece realmente a Ta_0.

La figura 5.6 ilustra un escenario en el que se realiza un ataque de este tipo. Supongamos que un atacante M se hace con el control de un dispositivo por el que pasan obligatoriamente todos los mensajes entre Ta_0 y Tb^*, por ejemplo un punto de acceso que comunica la red cableada con la red inalámbrica del espacio inteligente. En este escenario, el adversario saca partido del problema del intercambio anónimo de claves para hacerse con la clave de sesión KS_{Ta_0} y así poder robar todas los secretos que se transmitan mediante el protocolo de

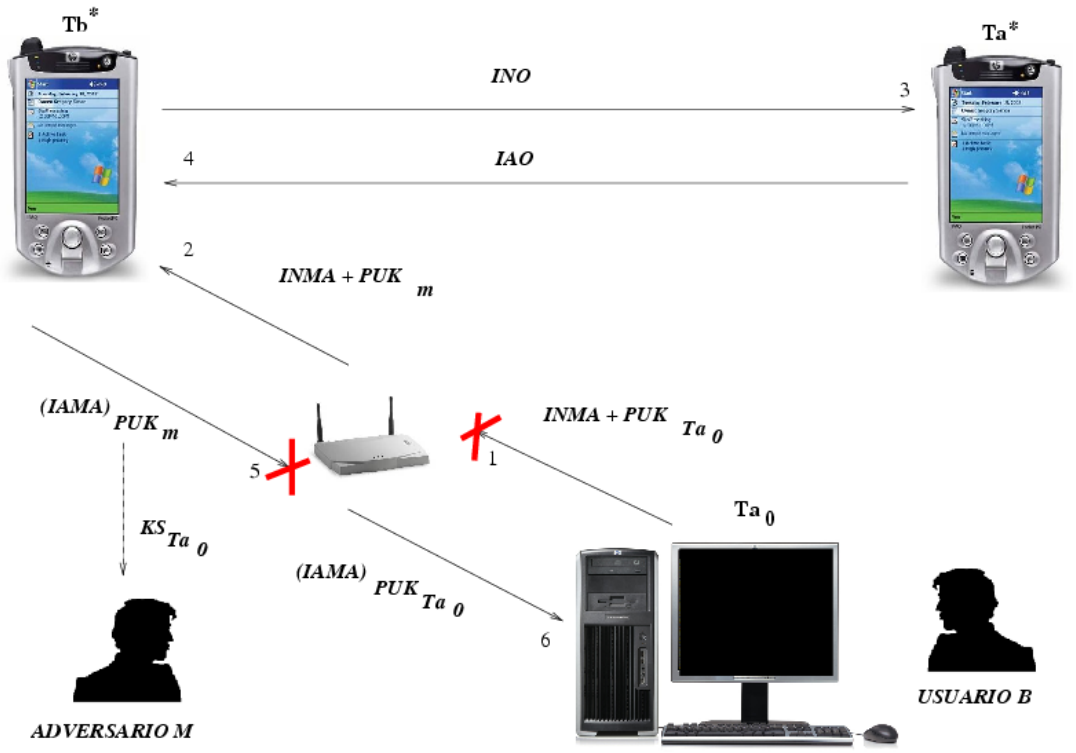

Figura 5.6: *Ataque de hombre en el medio para el protocolo basado en criptografía asimétrica sin confirmaciones comparativas. El adversario M es capaz de capturar la clave de sesión KS_{Ta_0} y robar los secretos de B que se pueden solicitar desde Ta_0.*

SSO.

5.8.4. Solución: confirmaciones comparativas

Como hemos visto, el intercambio anónimo de claves es vulnerable a ataques que ponen en riesgo los secretos del usuarios. Sin embargo, en nuestro esquema podemos sacar partido del hecho de que **el usuario B se encuentra delante del terminal** Ta_0. El usuario B es capaz de ver la pantalla del terminal y la pantalla de su UbiTerm, y comparar su contenido. Por lo tanto, existe otro canal de información además de la red de computadoras que conforma el espacio inteligente: **la visión del propio usuario B**. Mediante este nuevo canal, podremos verificar que la clave pública que se ha recibido es en realidad PUK_{Ta_0}

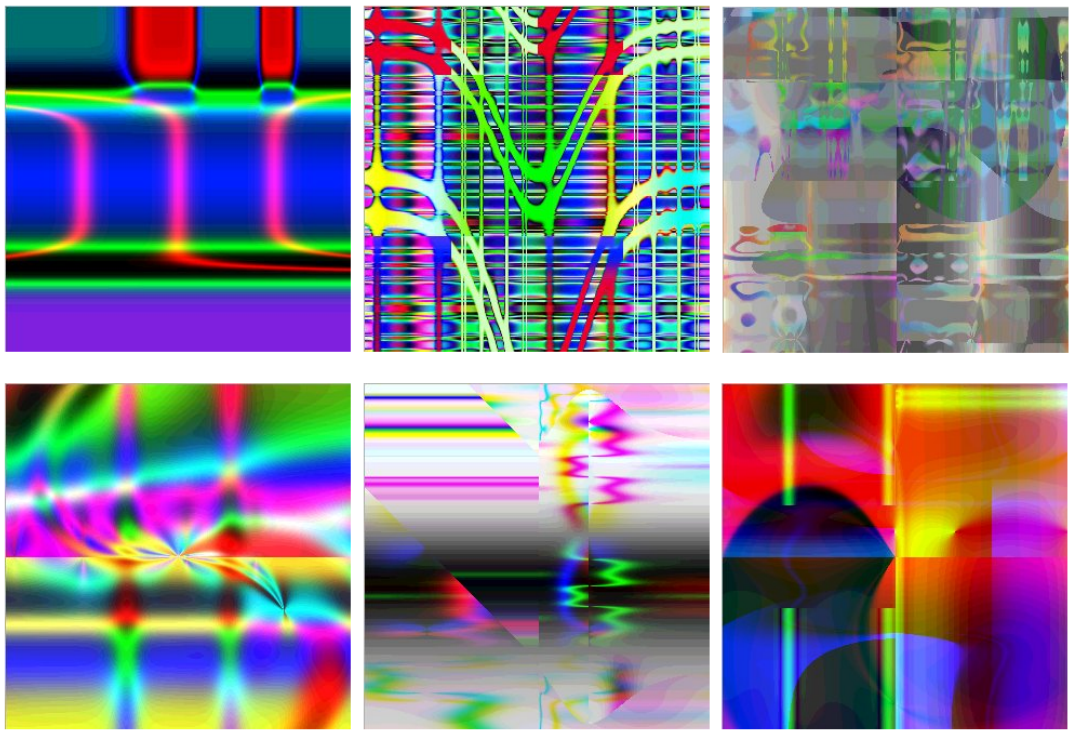

Figura 5.7: *Imágenes generadas a mediante* Random Art.

y no una clave generada por un impostor.

Como en el protocolo original, **el usuario B está obligado a confirmar en su UbiTerm que está arrancando el terminal** Ta_0 . La diferencia radica en que la confirmación que debe realizar en este protocolo requiere la comparación entre los datos que aparecen en la pantalla del UbiTerm (Tb^*) y los datos que aparecen en la pantalla del terminal cedido (Ta_0).

Cuando Ta_0 genera el par de claves pública/privada, extrae un resumen (*key fingerprint*) de la clave pública PUK_{Ta_0} y lo representa por su pantalla. De la misma forma, cuando Tb^* recibe el mensaje `ineedmainagent`, extrae la clave PUK_{Ta_0}, genera su resumen y lo representa por pantalla mientras que requiere la confirmación del usuario. En ese momento, el usuario es capaz de comparar el contenido de ambas pantallas y confirmar la operación si considera que es la misma información.

Ahora el problema reside en el método para representar el resumen de la clave PUK_{Ta_0} en las pantallas del UbiTerm y del terminal cedido.

El resumen de la clave puede ser un hash SHA-1, cuya longitud es de 160 bits. Si aplanamos su contenido en formato hexadecimal, obtenemos una cadena de 40 cifras. Si convertimos esa información en caracteres ASCII, obtendríamos un total de 20 caracteres. De la misma forma, si convertimos los datos a UNICODE podríamos obtener una cadena más reducida[3].

Creemos que estos métodos no son razonables. Por lo general, los humanos no reconocen bien las cadenas de caracteres que carecen de significado [37]. En estos casos, la comparación de las cadenas de caracteres puede llegar a provocar más obstrucción al usuario que la introducción de una contraseña en el terminal.

Čagalj et al. [34] proponen un protocolo que se basa en confirmaciones de mensajes (*Message Commitment*). El protocolo acorta la longitud de las cadenas a comparar en los dos nodos. El uso de dicho protocolo requiere aumentar el número de mensajes de nuestro actual protocolo para intercambiar los parámetros necesarios para conformar los datos de confirmación. Por tanto, su uso no es adecuado si se desea mantener el esquema de mensajes descrito en el protocolo de compartición original. En todo caso, el protocolo requiere la comparación de cadenas de caracteres que carecen de sentido para el humano.

Otra solución al problema expuesta en [34] es la conversión del resumen de la clave a una lista de palabras del lenguaje común, extrayéndolas de un diccionario. Para direccionar las entradas del diccionario RAE de 2001 ($\approx$ 88000 lemas) [146] sin colisiones se necesitan 16 bits. Por tanto, el resumen de 160 bits de la clave genera 10 palabras de diccionario. Esta solución no parece apropiada, ya que comparar 10 palabras resulta demasiado costoso para el usuario.

Otras soluciones proponen la comparación de imágenes generadas a partir del resumen de la clave. Esta técnica se denomina *Hash Visualization* [135] y se basa en la generación de imágenes de arte aleatorio (*Random Art*) [147]. Las imágenes se producen a partir de números generados mediante un generador de

[3]Habría que tener en consideración los códigos de carácter no válidos

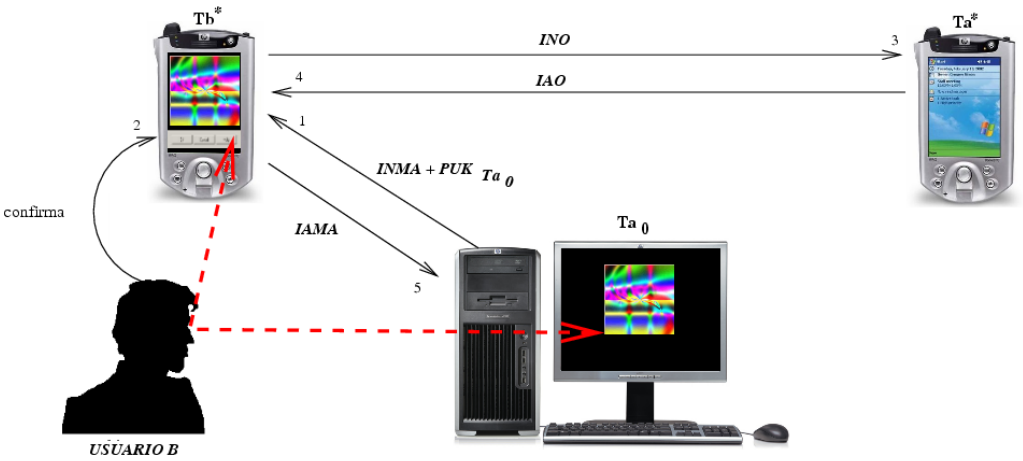

Figura 5.8: *Fases del protocolo de compartición de terminales basado en criptografía asimétrica.*

números pseudo-aleatorios seguro que se inicializa usando el resumen de la clave como semilla.

Creemos que esta aproximación es más apropiada que la anterior, ya que parece probado que los humanos son capaces de comparar y recordar imágenes con más facilidad que cadenas de caracteres [135; 47; 70].

Sin embargo, el método de *Hash Visualization* tiene el siguiente problema: las imágenes generadas a partir del resumen de la clave poseen las propiedades pre-imagen[4] de la función de resumen utilizada (en este caso SHA-1, que es una función segura), pero no poseen su propiedad de ausencia de colisiones [5]. Otro problema relacionado con este método es que aunque dos imágenes i y i' no sean iguales, pueden ser lo suficientemente cercanas para parecerlo al ojo humano ($i \approx i'$). Los autores de *Hash Visualization* ofrecen una estimación de la probabilidad de colisión basada en la experimentación [135].

La figura 5.7 muestra imágenes generadas mediante funciones *Random Art*.

[4]Propiedades que garantizan que a partir de cualquier salida de la función no es factible la deducción de la entrada a la función, y que dada una entrada x no es factible encontrar una entrada x' que genere la misma imagen.

[5]Propiedad que garantiza que no es factible encontrar dos entradas x y x' distintas que generen el mismo resumen.

En todo caso, hay que valorar el riesgo en la práctica. Para llevar a cabo un ataque de hombre en el medio, el adversario tendría que realizar las siguientes tareas:

1. Capturar el mensaje INMA y eliminarlo de la red para que no llegue hasta Tb^*. Las hipótesis sobre las que se construye el protocolo permiten al atacante llevar a cabo estas acciones.

2. Generar un par de claves de criptografía asimétrica PUK_m y PRK_m válidas tales que:

   $$\{PUK_m\}_{SHA1} = \{PUK_{Ta_0}\}_{SHA1}$$
   ó
   $$image(\{PUK_m\}_{SHA1}) \approx image(\{PUK_{Ta_0}\}_{SHA1})$$

   Siendo $image(hash)$ la imagen generada por una función de arte aleatorio que utiliza el resumen $hash$ como semilla.

3. Reenviar el mensaje **"ineedmainagent"** falso a Tb^*.

La primera opción del punto 2 no es factible, ya que la propiedad pre-imagen de SHA1 lo impide.

La segunda opción del punto 2 depende de las cualidades del propio usuario para diferenciar imágenes y a la probabilidad de encontrar dos imágenes cercanas o similares para la función de arte aleatorio utilizada. Hay que tener en cuenta que el adversario debe generar un par de claves **válidas** cuyo resumen de la clave pública genere una imagen de arte aleatorio cercana o similar a la generada por la clave original.

Hay que tener en consideración que la ventana de ataque es relativamente pequeña. Estas tareas deben realizarse en un tiempo $T < timeout - trest$, siendo $timeout$ el tiempo de espera máximo para la recepción del mensaje INMA y $trest$ el tiempo mínimo necesario para que prospere el resto de mensajes del protocolo

(mensajes INO, IAO y IAMA). El tiempo *timeout* tiene que ser suficiente para que el usuario B pueda confirmar la operación en su UbiTerm, teniendo en cuenta que si está arrancando el terminal, el usuario ya espera la petición de confirmación y está predispuesto a realizarla.

Creemos que la seguridad que aporta este método es suficiente para su uso en el ámbito para el que se propone, y por tanto su uso es razonable para este protocolo en concreto.

Capítulo 6

Compartición de dispositivos

6.1. Introducción

Un espacio inteligente resulta de la sinergia del conjunto de máquinas y dispositivos dispersos por el entorno de computación. El usuario realiza sus tareas combinando distintos dispositivos, y algunos de ellos pueden no pertenecerle. Dado que esos dispositivos pueden pertenecer a distintos usuarios o grupos de usuarios, se debe controlar el acceso a los mismos a través de algún mecanismo de seguridad con el fin de evitar su uso no autorizado.

Los humanos siempre llevan consigo el UbiTerm, que le permite obtener *Single Sing-On* (SSO) en el entorno y compartir sus terminales. En esta sección proponemos una nueva arquitectura basada en SHAD que permite al usuario compartir cualquier recurso de forma segura con la gente en la que confía. Esto es, en SHAD proponemos seguir el mismo esquema *Peer-to-Peer*, o humano a humano, descrito en el capítulo anterior.

El objetivo es que los usuarios que confían entre sí puedan compartir sus recursos de forma espontánea cuando ambos están presentes en el espacio inteligente, soportando desconexiones de forma controlada.

La arquitectura de compartición de dispositivos se basa en el agente de SSO de SHAD, y añade distintos mecanismos para controlar el acceso a los recursos de los usuarios. El agente SHAD tiene ahora tres cometidos esenciales:

- Ofrecer *Single Sing-On* al usuario.
- Controlar la cesión de terminales entre los usuarios.
- Controlar la cesión de dispositivos entre los usuarios.

De ahora en adelante, cuando citemos al agente SHAD, nos estaremos refiriendo al **agente SHAD de compartición de dispositivos**.

6.2. El problema

Los usuarios del espacio inteligente quieren trabajar en él como si fuera un único sistema. Además, los usuarios eventualmente necesitan utilizar dispositivos que se encuentran dispersos por el entorno y que no les pertenecen.

Por ejemplo, un usuario que se mueve en el entorno con su ordenador portátil puede necesitar en cualquier momento tomar prestado un ratón inalámbrico de cualquier sala para poder manipular un gráfico cómodamente. Por supuesto, el usuario quiere poder usarlo sin tener que desconectar la base receptora del ratón del PC en el que está conectado. Lo único que desea es especificar a su portátil (mediante un menú, por ejemplo) el ratón que quiere utilizar, cogerlo y empezar a usarlo. Para él, la ilusión de que el sistema es uno sigue presente. Sólo ha tenido que seleccionar el ratón y empezar a usarlo.

El problema es que el ratón tiene un dueño, y es posible que no esté dispuesto a prestárselo a cualquier usuario del sistema. Más aun, es posible que sí esté dispuesto a prestárselo, pero que en un momento dado lo quiera reclamar para poder usarlo él.

Por esa razón, tiene que existir una arquitectura que controle el acceso a los dispositivos. Esta arquitectura también debe desaparecer en el entorno en la medida de lo posible para que el usuario mantenga la ilusión de que el sistema es un único computador disponible ubícuamente.

6.3. Importar y exportar dispositivos

Las máquinas del espacio inteligente tienen que ejecutar software que permita exportar e importar dispositivos entre máquinas. Para ello, se pueden seguir dos estrategias:

1. Usar un sistema operativo que permita estás operaciones [21; 20].

2. Usar una capa *middleware* entre las aplicaciones y el sistema operativo [156; 104; 65; 66; 82].

El esquema general y las ideas principales propuestas en la arquitectura de SHAD para la compartición de dispositivos es independiente del tipo de software que permita importar/exportar dispositivos entre máquinas. Estas ideas se pueden aplicar a cualquier tipo de sistema, independientemente del sistema subyacente.

No obstante, la arquitectura se ha diseñado e implementado para un espacio inteligente en el cual las máquinas ejecutan un sistema operativo que permite importar/exportar dispositivos a través de sistemas de ficheros en red [21; 20; 17; 15].

A partir de este momento, se considera que el sistema operativo ofrece los mecanismos necesarios para compartir los dispositivos de las máquinas.

6.4. Arquitectura SHAD de compartición de dispositivos

Esta sección ofrece un diseño genérico para la arquitectura de compartición de dispositivos, de tal forma que se pueda implementar para cualquier tipo de sistema. En secciones posteriores se ofrece el diseño y los detalles de implementación para el sistema operativo Plan B.

6.4.1. Entidades

En la figura 6.1 se muestra a grandes rasgos la arquitectura global de la arquitectura de compartición de dispositivos. En la figura podemos encontrar las siguientes entidades:

- **Los usuarios** emparejan sus UbiTerms, crean roles y se los asignan entre sí para definir el control de acceso a los dispositivos.
- **El espacio inteligente** provee servicios a los usuarios del sistema, de la misma forma que en la arquitectura de SSO que proponemos.

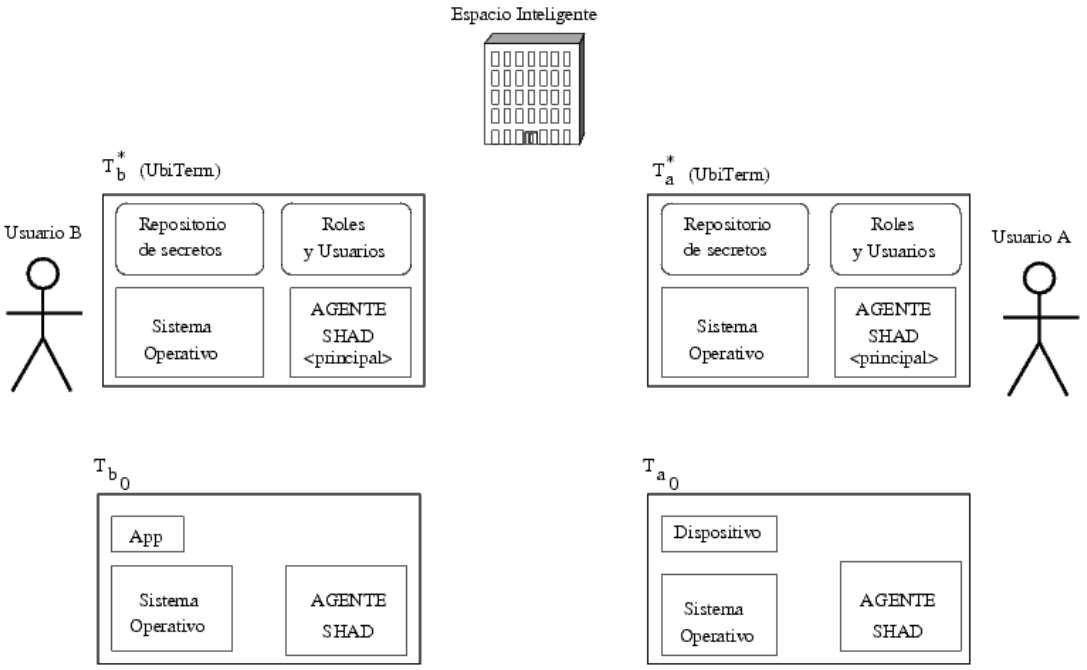

Figura 6.1: *Entidades de la arquitectura de compartición de dispositivos.*

- **El repositorio de secretos** que cumple la misma función que en la arquitectura de SSO propuesta.
- **Los dispositivos** que se controlan a través de un manejador, al que se puede acceder a través de la red mediante el sistema operativo. El acceso al dispositivo está controlado a por medio de una serie de permisos. Esos permisos se pueden modificar a través del sistema operativo local.
- **Roles y usuarios** en una base de datos que se mantiene en el UbiTerm de cada usuario. La base de datos contiene los roles creados por el usuario. También contiene una lista de los usuarios con los que el dueño ha emparejado su UbiTerm, así como los roles que tiene asignado cada uno de ellos.
- **Los agentes SHAD**, que se coordinan para permitir el acceso de los usuarios a dispositivos compartidos, además de ofrecer el servicio de SSO y el de compartición de terminales. Los usuarios reciben notificaciones del

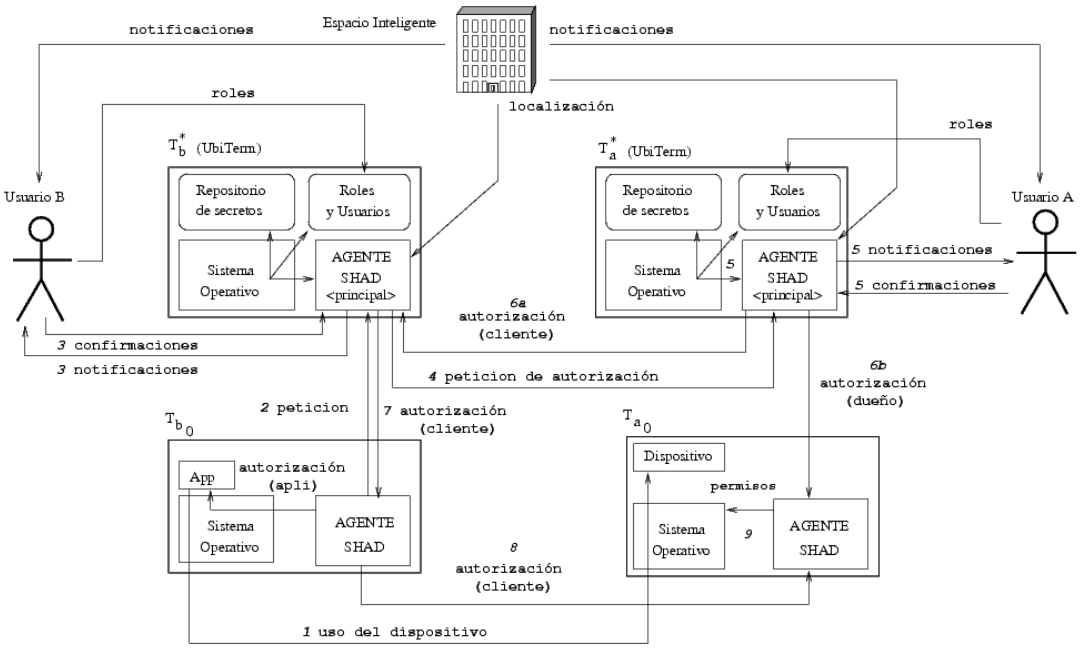

Figura 6.2: *Arquitectura global del sistema de compartición de dispositivos.*

agente principal. También pueden confirmar las operaciones en él, al igual que en las arquitecturas anteriormente propuestas.

- **Las aplicaciones** que ejecuta el usuario y que pueden acceder a dispositivos. Estas no residen en su máquina local. Las aplicaciones acceden a los dispositivos remotos a través de mecanismos ofrecidos por el sistema operativo.

- **El sistema operativo** ofrece los mecanismos para utilizar dispositivos remotos y mecanismos básicos de control de acceso a los recursos.

6.4.2. Esquema de funcionamiento

El esquema de funcionamiento de la arquitectura se puede ilustrar mediante una secuencia de operaciones. Tomemos como modelo el escenario presentado en la figura 6.2, en el que el usuario B tiene un terminal Tb_0 en el que se está

ejecutando una aplicación que solicita usar un dispositivo que está controlado por el terminal Ta_0, que pertenece al usuario A. La secuencia es la siguiente:

1. La aplicación que ejecuta en Tb_0 solicita al agente SHAD local que obtenga la autorización para usar el dispositivo que controla Ta_0.

2. El agente SHAD local se pone en contacto con su agente principal y le solicita autorización para que la aplicación acceda al dispositivo deseado.

3. El agente SHAD principal Tb^* pide confirmación al usuario B. Esta confirmación no es obligatoria como en el caso de la arquitectura de compartición de terminales, sino que depende de la configuración.

4. El agente principal de B, Tb^*, se pone en contacto con el agente principal de A, Ta^*. Se pide autorización para acceder al dispositivo indicado.

5. El agente principal de A autentica y verifica la petición. Después consulta la base de datos de usuarios y roles. Comprueba qué roles tiene asignados B. Después comprueba si dichos roles tienen garantizado el acceso al dispositivo especificado.

   Si es necesario (según la configuración), Ta^* pide confirmación al usuario.

6 a. Si se aprueba el acceso, Ta^* envía la autorización necesaria a Tb^*.

6 b. Concurrentemente, Ta^* envía un mensaje a Ta_0 que aprueba el permiso para que la aplicación utilice el dispositivo.

7. Tb^* envía la autorización al agente Tb_0.

8. El agente Tb_0 entrega la autorización al agente Ta_0. Si la autorización es correcta, el agente Ta_0 modifica el control de acceso al dispositivo en el sistema operativo.

9. El sistema operativo de Ta_0 comprueba los permisos del dispositivo y la autorización que presenta la aplicación remota. Si la autorización y los permisos son compatibles, se permite el acceso al dispositivo.

10. La aplicación comienza a usar el dispositivo.

    Si A desea revocar el uso del dispositivo, basta con actualizar los permisos para que la aplicación ya no pueda acceder a él (similar al paso 6.4.2).

Cuando hablamos de autorización, entendemos cualquier mecanismo que permita autenticar y autorizar una acción, como por ejemplo mecanismos como *capabilites* [161], certificados de atributos (PAC) [86] o una autenticación seguida de control de acceso por listas (ACL) que regule los modos de acceso al dispositivo.

La comunicación entre los agentes principales y los terminales se basa en sus *claves de terminal*. La comunicación entre los agentes principales de Ta^* y Tb^* se basa en la *clave de emparejamiento* de A y B, $K_{a,b}$. Todas esas claves se encuentran almacenadas en el *repositorio de secretos* del usuario A y del usuario B. La comunicación entre la aplicación de Tb_0 y el dispositivo de Ta_0 se basa en la autorización y el permiso emitidos.

Las confirmaciones son opcionales por dos motivos:

- Puede que Tb_0 intente usar el dispositivo de Ta_0 sin que el usuario sepa que lo hace, ya que el acceso al dispositivo es de muy poco nivel de abstracción.
- Si el uso de dispositivos es intensivo, las confirmaciones pueden provocar mucha obstrucción al usuario.

Por omisión, las confirmaciones no son necesarias. Sin embargo, añadiendo la posibilidad de configurarlas, el sistema gana en flexibilidad pero no en complejidad, ya que los mecanismos necesarios para implementar las confirmaciones ya existen en las arquitecturas anteriormente descritas.

6.4.3. Diseño

No es necesario modificar drásticamente el diseño del agente SHAD para incluir los mecanismos necesarios para hacer posible la compartición segura de dispositivos.

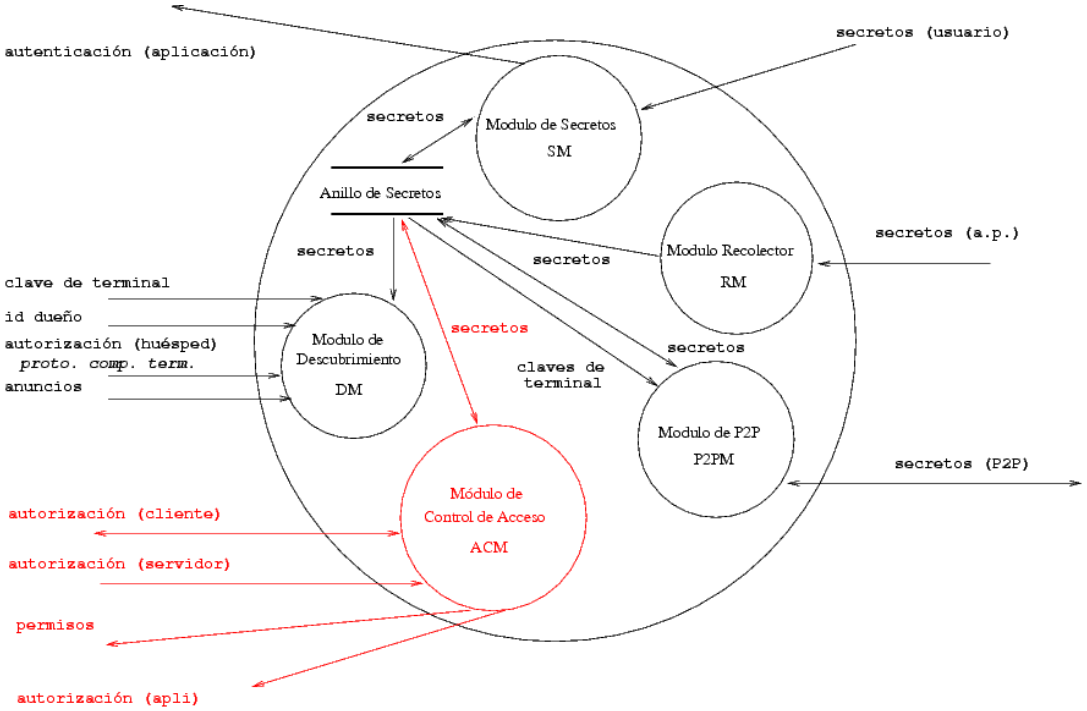

Figura 6.3: *Diagrama de flujo de datos de nivel uno correspondiente a un agente común para la compartición de dispositivos.*

Es necesario incluir un nuevo módulo que gestione el control de acceso a los dispositivos del usuario. Este módulo es necesario tanto en el agente principal como en el agente común. El módulo se denomina **Módulo de Control de Acceso** (ACM).

Las figuras 6.4 y 6.3 ofrecen el diagrama de flujo de datos de primer nivel del agente principal y el agente común respectivamente. Las diferencias respecto los diagramas mostrados en el capítulo anterior se remarcan en color rojo.

Los agentes principales de los distintos usuarios (que estén emparejados) deben ser capaces de descubrirse y autenticarse mutuamente. Para ello, los agentes principales tienen un flujo de datos en sí llamado `anuncio de servicio`. Dicho flujo ya existía en anteriores capítulos, se remarca en rojo porque es necesario ampliar el protocolo para descubrir agentes principales de distintos usuarios.

Nótese que dicho flujo existía para que los agentes SHAD pudieran encontrar a su agente principal. Ahora, además de dicha funcionalidad, los agentes principales de distintos usuarios se pueden descubrir entre sí para poder ofrecer el servicio de compartición de dispositivos.

El módulo de control de acceso tiene la siguiente funcionalidad:

- **Agente principal:**
  - Solicitar la autorización para que una aplicación pueda utilizar un dispositivo que pertenece a otro usuario. Esta función la realiza el agente principal del usuario que intenta usar un dispositivo de otro. El flujo correspondiente es `petición de autorización`.

    Para ello, debe poder acceder a las *claves de emparejamiento* almacenadas en el *anillo de secretos*. Para realizar las funciones que siguen también es necesario tener acceso a dicha clave en el anillo.
  - Proporcionar la autorización solicitada en el punto anterior. Esta función la realiza el agente principal del dueño del dispositivo que se intenta utilizar. El flujo correspondiente es `autorización (cliente)`. No confundir con el flujo de `autorización (huésped)` del protocolo de compartición de terminales.
  - Proporcionar la autorización necesaria para que la máquina que controla el dispositivo acceda a servírselo a la aplicación cliente. El flujo correspondiente es `autorización(servidor)`. No confundir con el flujo `autorización (dueño)` del protocolo de compartición de terminales.
  - Adquirir y evaluar la lista de roles y de usuarios. Los roles definen el acceso a los recursos. Los usuarios emparejados tienen roles asociados. Cuando se recibe una petición de autorización, se evalúan los roles asociados al usuario y se concede la autorización en base a ellos. El flujo asociado con esta acción es `roles y usuarios`. La base de datos de roles y usuarios se encuentra fuera del agente principal.

- El módulo almacena la autorización en el `anillo de secretos` para su posible reutilización. También adquiere de allí las `claves de terminal` de los agentes comunes que le solicitan servicio.
- Enviar al usuario notificaciones relacionadas con la cesión de un dispositivo.
- Recibir del usuario confirmaciones relacionadas con la cesión de un dispositivo.

▪ **Agente común:**

- El agente común debe reclamar a su agente principal la autorización necesaria para poder acceder a un dispositivo que no pertenece al usuario. El flujo de datos correspondiente es `autorización (cliente)` entre el agente principal y el agente común,

  Para ello el módulo necesita conocer la *clave de sesión* con su agente principal, que la obtiene del **anillo de secretos**. Ese flujo se representa como `secretos` entre el módulo y el anillo.
- El agente común de la máquina que intenta usar el dispositivo tiene que enviar la autorización obtenida al agente de la máquina que sirve el dispositivo. Esa máquina debe procesar la `autorización (apli)` y validarla con la autorización que le envió su agente principal `autorización (servidor)`.
- El agente de la máquina que controla el dispositivo debe poder cambiar los mecanismos de control de acceso del sistema operativo, de tal forma que la aplicación externa pueda acceder a él. El flujo de datos asociado es `permisos`.
- El módulo almacena la autorización en el `anillo de secretos` para su posible reutilización. Ese flujo también se representa mediante el flujo `secretos` entre el módulo y el anillo.

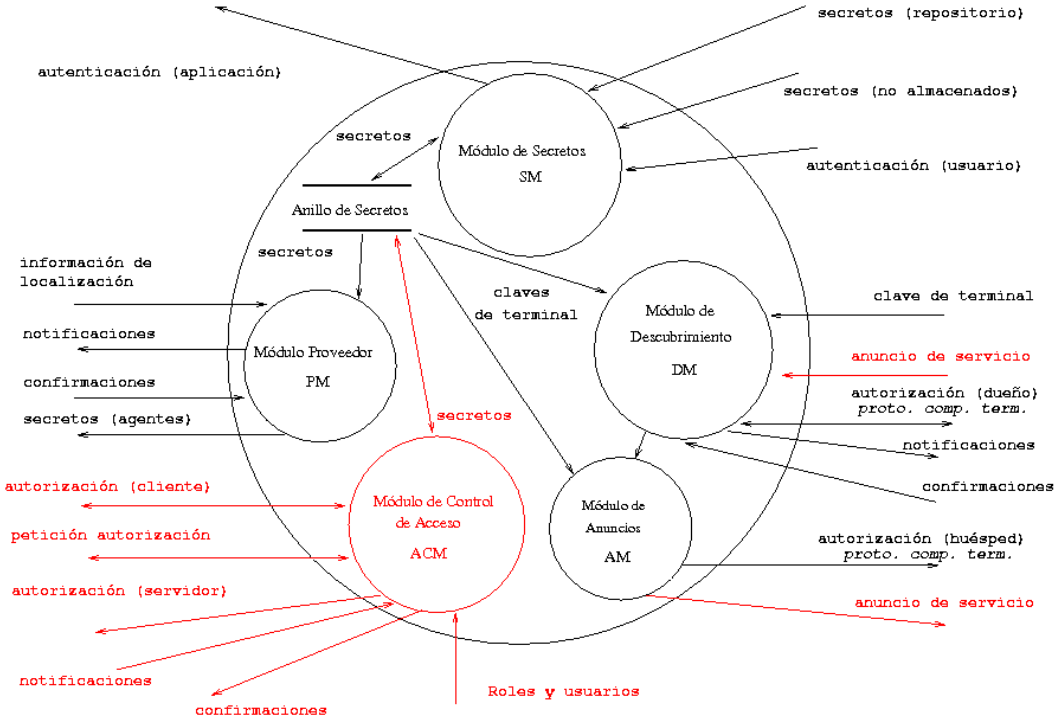

Figura 6.4: *Diagrama de flujo de datos de nivel uno correspondiente a un agente principal para la compartición de dispositivos.*

El flujo de datos correspondiente a `roles y usuarios` se obtiene de una base de datos externa. Dicha información es la que define el control de acceso a los recursos.

Se propone un esquema basado en roles (RBAC) porque se ajusta al modelo centrado en el humano en el que se fundamenta el sistema. Se propone que dicha información incluya la descripción de los propios roles, así como como los roles asignados a cada humano emparejado con el usuario. Los detalles de esta base de datos dependen del propio sistema subyacente. En la sección dedicada a la implementación de esta arquitectura para el sistema operativo Plan B se detalla en profundidad la base de datos de usuarios y roles que se ha implementado para el prototipo.

6.5. Protocolos

Los protocolos necesarios para poder compartir dispositivos dependen en gran medida del sistema en el que se van a implantar, ya sea un sistema operativo o un *middleware* en concreto. Estos protocolos también dependen del tipo de autorización que se disponga a utilizar (credenciales, *capabilities*, etc.). Por tanto, no hay protocolos genéricos que puedan ser implementados para distintos sistemas, como era el caso en las dos arquitecturas presentadas en los capítulos 4 y 5.

En el caso del trabajo que se presenta, que se diseña e implementa para el sistema operativo Plan B, sólo es necesario añadir un único protocolo para hacer posible la compartición de dispositivos, ya que el prototipo se apoya en los protocolos de la arquitectura de SSO. El protocolo se describe en la sección 6.7.3.

6.6. Usando dispositivos en Plan B

En el sistema operativo Plan B, los dispositivos se exportan a través de sistemas de ficheros virtuales. Para ello se utiliza un protocolo de sistemas de ficheros en red basado en llamadas a procedimiento remoto (RPC), **9P** [3].

En el protocolo 9P, cuando un cliente necesita acceder a un dispositivo que sirve un servidor de ficheros en concreto, tiene que realizar las siguientes tareas:

- Arrancar el protocolo y consultar si necesita autenticación. Los dispositivos que se comparten por SHAD necesitan autenticación.
- Ejecutar un protocolo de autenticación llamado P9SK1.
- Atarse al raíz del sistema de ficheros.
- Buscar el fichero indicado.
- Abrir el fichero que representa la funcionalidad deseada del dispositivo.

- Leer y/o escribir en el fichero.
- Cerrar el fichero.

El agente SHAD de SSO lleva a cabo el protocolo de autenticación P9SK1 de forma automática. Cuando una aplicación necesita montar un sistema de ficheros, contacta con el agente SHAD local. El agente local se encarga de ejecutar la lógica de autenticación.

El protocolo P9SK1 [13] involucra al agente SHAD del cliente, el agente SHAD del servidor de ficheros y un servidor de autenticación (AS). El protocolo es similar al utilizado en Kerberos [182] y se basa en resguardos (*tickets* y *authenticators*). La autenticación ante el AS se realiza mediante una clave (derivada de una contraseña) asociada al dominio de autenticación escogido. La autenticación es recíproca. Los pasos en la autenticación P9SK1[1] son:

1. El agente del cliente envía una lista de dominios de 9P en los que se puede autenticar (servidores en los que tiene una cuenta).
2. El agente del servidor escoge uno de ellos. Debe elegir un dominio para el que él también tenga cuenta y se pueda autenticar.
3. El agente del cliente envía un *nonce* al servidor de ficheros.
4. El agente del servidor responde con una petición de *ticket*, los dominios en los que se puede realizar la autenticación y otro *nonce*.
5. El agente del cliente necesita ponerse en contacto con el servidor de autenticación para el dominio elegido. Para ello, consulta la base de datos `ndb` [122]. Esta base de datos contiene una descripción de las máquinas conocidas para el terminal y las conocidas en toda la red. Entre esa información se puede encontrar el servidor de autenticación de un dominio concreto.

[1] En realidad, los pasos corresponden a dos protocolos distintos, `p9any` y `p9sk1`, pero se presentan como uno con el fin de simplificar el esquema.

El agente cliente contacta con el servidor de autenticación y le entrega la petición de *ticket* y el *nonce* que le entregó el agente del servidor. Con ese *nonce*, el servidor de autenticación genera el *ticket* que el agente del cliente debe entregar al agente del servidor de ficheros. También proporciona al agente del cliente la clave de sesión que le permite crear un *authenticator* con el *nonce* que le envió el agente del servidor (que permite saber al agente del servidor que el cliente conoce la clave de sesión).

6. El *ticket* que recibe el agente del servidor de ficheros contiene la clave de sesión, y le permite crear otro *authenticator* con el *nonce* del agente del cliente para que la autenticación sea mutua.

Una vez montado un dispositivo, el acceso a los ficheros que representan su interfaz se controla a través de una lista de permisos (ACL). El cliente se ha autenticado a nombre del dueño del terminal en el que ejecuta. El control de acceso se establece en base a ese nombre, y la gestión del control de acceso queda en manos del servidor de ficheros.

Los ficheros ofrecidos por el servidor tienen un dueño y un grupo asignado. Los permisos de los ficheros se establecen para el dueño del dispositivo, el grupo y el resto de usuarios, de la manera tradicional en los sistemas UNIX [89, Capítulo 4].

6.7. SHAD para la compartición de dispositivos en Plan B

El objetivo de este capítulo es la creación de una arquitectura que permita compartir los dispositivos en el sistema operativo Plan B con otros usuarios de una forma segura a la vez que intuitiva y confortable, siguiendo las ideas generales de SHAD.

SHAD delega la confidencialidad de la transmisión de los datos entre dispositivos en el protocolo 9P. Al construir la arquitectura sobre este protocolo,

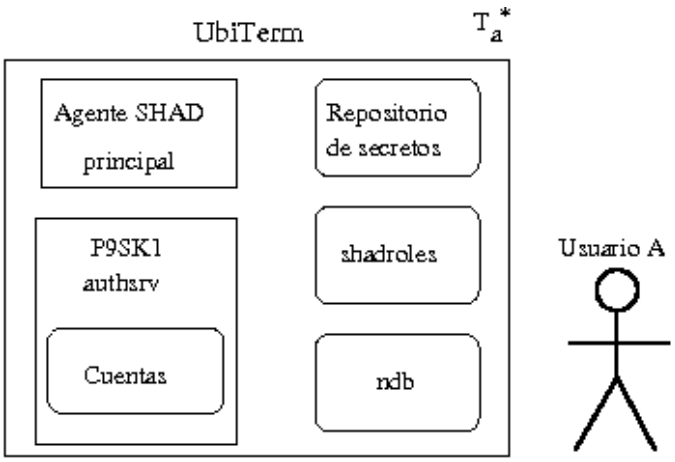

Figura 6.5: *Componentes que forman el UbiTerm del prototipo.*

suponemos que sus mecanismos para proteger los datos transmitidos son correctos y seguros.

Es posible ofrecer dicha arquitectura si situamos en el UbiTerm los siguientes componentes, que junto a ciertas modificaciones en el prototipo NetFactotum ofrecerían la funcionalidad del módulo ACM descrito anteriormente.

- **El agente principal de SHAD** (NetFactotum). Ofrece SSO a los terminales del usuario. Por tanto, ofrece acceso con autenticación automática a servidores de ficheros 9P. Si el agente principal posee el secreto necesario para autenticarse ante un servidor de autenticación, podrá acceder a todos los servidores de ficheros que acepten dicho servidor de autenticación en la negociación.

  El único factor que debe tener en cuenta el usuario es que los secretos P9SK1 se distribuyen entre sus terminales en base a los atributos y restricciones tratadas en el capítulo 4.

- **Un servidor de autenticación P9SK1**. Cada UbiTerm ejecuta un servidor de autenticación que sirve la autenticación al dominio *uname*`-shad`, donde *uname* es el nombre del dueño del UbiTerm.

Los servidores de ficheros que importen dispositivos que se quieran compartir a través de SHAD **sólo podrán aceptar dicho dominio en la negociación**.

- **Una base de datos**, `ndb`, en la que se describen los dominios de autenticación SHAD para los usuarios emparejados.

- **La base de datos de usuarios y roles**, mediante la cual el usuario puede controlar el acceso a sus dispositivos. Los servidores de ficheros de los dispositivos compartidos tienen mecanismos especiales de control de acceso implementados, de tal forma que controlan el acceso mediante los roles asignados en la base de datos. Por tanto, el control de acceso a los dispositivos que adopta SHAD deja de ser puramente basado en ACLs para convertirse en una mezcla entre ACLs y RBAC. El control de acceso se trata en la sección 6.7.4. La base de roles y usuarios toma el nombre de `shadroles`.

6.7.1. Servidor de autenticación P9SK1

Las autorizaciones que aparecen en el diseño se implementan mediante la autenticación P9SK1 y el control de acceso basado en roles.

Cada UbiTerm ejecuta un servidor de autenticación P9SK1, de tal forma que la autenticación consiste en asociar la aplicación a un nombre de usuario. Posteriormente, se aplicará el control de acceso al recurso en base a este nombre asociado. Por lo tanto, la autenticación consiste únicamente en convencer a la máquina servidora de que la aplicación ejecuta en nombre de un usuario en concreto.

El servidor de autenticación para el protocolo P9SK1 es un programa llamado `authsrv`. Este programa escucha en el puerto 567 de TCP y comienza el protocolo cuando se le abre una conexión. Para llevar a cabo el protocolo, el servidor de autenticación necesita mantener cuentas para sus usuarios. La clave (derivada de la contraseña) del usuario se encuentra guardada en una base de

datos que está cifrada con la *clave del AS*.

En Plan 9 y Plan B la *clave de AS* se guarda en la NVRAM de la máquina para que el servidor de autenticación pueda arrancar sin asistencia humana. Pero en el esquema que proponemos, el servidor de autenticación va a ejecutar en la **única máquina en la que el usuario va a autenticarse explícitamente: su UbiTerm**. Por tanto, no es necesario que la *clave de AS* se encuentre almacenada en texto claro en el hardware. La contraseña introducida por el usuario puede servir para descifrar la base de datos del servidor de autenticación cuando se reclame este servicio.

En el proceso de emparejamiento de dos usuarios, el agente SHAD de cada UbiTerm debe crear una cuenta para el otro en la base de datos del servidor de autenticación. La contraseña asignada a ese usuario se genera aleatoriamente, y no se le comunica en el proceso de emparejamiento. Esta tarea es fácilmente automatizable.

La base de datos del servidor de autenticación también tiene que tener una cuenta creada para el dueño del dispositivo. En el repositorio de secretos del dueño se debe almacenar el secreto para esa cuenta. Esta tarea se puede automatizar también, y se debe realizar en tiempo de configuración del UbiTerm. Cuando un cliente intenta acceder a un servidor de ficheros y es necesaria la autenticación, su agente SHAD local debe conocer el secreto para acceder al dominio necesario. El agente SHAD local usará los mecanismos de SSO para conseguir esa clave del agente SHAD principal.

El dominio de autenticación para los dispositivos SHAD es **obligatoriamente** el dominio *uname*`-shad` (siendo *uname* el nombre del dueño del dispositivo a utilizar). Por tanto, cuando un usuario quiere usar un dispositivo ajeno, el agente SHAD del terminal cliente (el terminal que ejecuta la aplicación que necesita usar el dispositivo en cuestión) se debe autenticar ante el servidor de autenticación P9SK1 que ejecuta en el UbiTerm del dueño del dispositivo.

Para que dicha autenticación pueda llevarse a cabo, el agente principal del

usuario cliente necesita conocer la contraseña para el protocolo P9SK1 en el dominio del dueño del dispositivo. Entonces se pueden dar dos situaciones:

1. Que el agente principal del usuario cliente conozca la contraseña para el dominio. En ese caso, cuando la aplicación cliente intenta acceder al dispositivo, el agente SHAD local intentará autenticarse. Para ello, solicitará la contraseña necesaria a su agente SHAD principal mediante el protocolo de SSO. Una vez obtenida, el agente SHAD local comenzará el protocolo P9SK1 con el servidor de autenticación del UbiTerm del dueño del dispositivo. En este punto, pueden darse dos casos:

   1a. La contraseña sigue siendo válida. La contraseña que consigue el agente local de la máquina cliente sigue siendo válida para autenticarse ante el servidor P9SK1 del UbiTerm del dueño. En este caso, la autenticación se lleva a cabo y la aplicación cliente queda autenticada a nombre del usuario cliente.

   1b. La contraseña ha caducado. Se puede dar el caso de que la contraseña que tiene almacenada el agente principal de SHAD del usuario cliente haya caducado. En este caso, el agente SHAD principal del cliente tiene que ejecutar un protocolo para obtener una nueva contraseña para acceder al dominio del usuario dueño del dispositivo. El protocolo se ejecuta entre los dos agentes SHAD principales de los usuarios involucrados (cliente y dueño).

2. Que el agente principal del usuario cliente no conozca la contraseña para ese dominio. Cuando el la aplicación cliente solicita autenticación, el agente local solicita la clave necesaria a su agente principal. Si la clave suministrada no es correcta, el protocolo P9SK1 fracasa. Entonces, el agente común de la máquina cliente solicita al agente principal la clave de nuevo y se debe ejecutar el protocolo citado en el punto anterior para obtener una clave fresca para autenticarse ante ese dominio.

Para que un agente común pueda solicitar el refresco de la cuenta P9SK1 de un dominio SHAD, se ha añadido un nuevo comando al protocolo de SSO (Capítulo 4, sección 4.5.3). El comando se llama `obtainkey`.

Nótese que la autenticación se realiza al inicio de una conexión 9P. Una vez autenticada la conexión, no es necesario volver a autenticarla hasta que esta acabe. La misma autenticación puede utilizarse para usar distintos ficheros a través de la misma conexión [12].

Por esa razón, el uso del dispositivo soporta desconexiones de ambos UbiTerms. Una vez autenticada la conexión, la aplicación cliente no depende del agente SHAD (común o principal) ni del servidor de autenticación del dominio en concreto.

6.7.2. Dominios de autenticación

Como ya se ha comentado, cada usuario tiene su propio dominio de autenticación P9SK1. Los dominios de autenticación se definen en el fichero de configuración de `ndb` [122]. En dicho fichero se especifica la dirección del servidor de autenticación para cada dominio, y también se especifica el dominio de autenticación necesario para acceder a las máquinas conocidas.

Su contenido es dinámico: cuando el agente principal ejecuta el protocolo de actualización de la cuenta, refresca la dirección del servidor de autenticación del dominio de dicho usuario con la dirección de la que obtuvo respuesta.

No obstante, el esquema deja la posibilidad de fijar las direcciones de los UbiTerms en la base de datos `ndb` y prescindir de la actualización. Si se opta por esa opción, el UbiTerm de los usuarios siempre tendrá que ser la misma máquina.

El cuadro 6.1 muestra un ejemplo de un fichero `ndb` en el que se especifican tres usuarios del sistema.

Las tres primeras tuplas (que comienzan por `authdom`) definen los dominios de autenticación para los tres usuarios. Las tuplas definen el dominio de autenticación (`authdom`) y la dirección del UbiTerm del usuario.

```
authdom=esoriano-shad
     auth=212.128.4.126
authdom=nemo-shad
     auth=212.128.4.132
authdom=paurea-shad
     auth=193.147.71.88

dom=mosquito.esoriano-shad sys=mosquito ip=212.128.4.133
     authdom=esoriano-shad auth=212.128.4.126

dom=sargazos.nemo-shad sys=sargazos ip=212.128.4.139
     authdom=nemo-shad auth=212.128.4.132

dom=sailu.paurea-shad sys=sailu ip=212.128.4.155
     authdom=paurea-shad auth=193.147.71.88
```

Cuadro 6.1: *Ejemplo de un fichero* `ndb` *para tres usuarios del sistema (*`esoriano`*,* `paurea` *y* `nemo`*). Cada uno exporta dispositivos mediante SHAD desde un terminal.*

Las tres últimas tuplas (que comienzan por `dom`) definen tres terminales, cada uno correspondiente a un usuario. El atributo `dom` define el nombre completo del terminal, incluyendo su dominio. El atributo `sys` define el nombre del terminal. `Ip` define su dirección IP. El atributo `authdom` define el dominio de autenticación para esa máquina. Por último, el atributo `auth` define la dirección del servidor de autenticación del dominio. Este último atributo no es necesario, ya que la dirección del servidor de autenticación para el dominio ya está definido en las tuplas que definen los propios dominios, pero se utiliza para fijar el servidor de autenticación en el caso de que haya varios para un mismo domino (Plan B soporta ese caso).

Lo habitual en una configuración de Plan B es que los terminales y el UbiTerm monten el mismo sistema de ficheros raíz, y por tanto compartan la base de datos `ndb`. Si esto no fuera así, los terminales tendrían que montar la base de datos `ndb` del sistema de ficheros del UbiTerm, para poder adquirir los dominios de autenticación para los otros usuarios actualizados.

6.7.3. Protocolo de actualización de cuenta

El protocolo se ejecuta entre los dos agentes principales de los usuarios involucrados. Se ejecuta cada vez que una aplicación intenta acceder a un dispositivo y no se puede llevar a cabo la autenticación para el dominio correspondiente al dueño.

El protocolo consta de dos mensajes: petición y respuesta. Es similar al protocolo de SSO en el que un agente común le solicita al agente principal una tupla que corresponda a un secreto en concreto. En este protocolo, sólo se puede solicitar la tupla correspondiente al usuario cliente del protocolo P9SK1 para el dominio correspondiente al dueño del dispositivo. El protocolo se describe en el cuadro Protocolo 7, suponiendo que el usuario B quiere solicitar una nueva clave al usuario A para autenticarse en su dominio.

Los mensajes son:

- `refreshaccount` (RFA): reclama una nueva contraseña P9SK1 para el dominio SHAD de un usuario al agente principal del mismo.
- `newaccount` (NAC): respuesta al mensaje RFA. Sólo hay respuesta al mensaje si los usuarios están emparejados y el mensaje recibido es correcto.

Si el mensaje de petición es correcto (se puede autenticar, descifrar su contenido y comprobar su integridad) y no hay restricciones que impidan que el usuario acceda al sistema (no se ha cancelado su cuenta, sigue emparejado, etc.), entonces el agente SHAD principal del dueño procede a cambiar la contraseña en el servidor de autenticación P9SK1. Para ello, genera una cadena de caracteres aleatoria y actualiza la cuenta del usuario.

El mensaje de respuesta contiene la nueva contraseña asignada a la cuenta del usuario cliente en el servidor de autenticación. Una vez que el agente principal del usuario cliente tiene la nueva clave, sus aplicaciones se podrán autenticar para usar los dispositivos del otro.

Los mensajes se basan en la *clave de emparejamiento* de los dos usuarios.

Protocolo 7 Protocolo para actualización de cuenta

$REQ = [\texttt{refreshaccount}, uname_a, uname_b, N_1, tstamp_T b]$

$Tb^* \longrightarrow BROADCAST$

$\texttt{refreshaccount}, uname_a, uname_b$

$IV, \{ranb_1, \{REQ\}_{SHA1}, REQ\}_{K_{a,b}}$

$RES = [\texttt{newaccount}, uname_a, uname_b, address, N_1 + 1, tstamp_T a, tuple]$

$Tb^* \longleftarrow Ta^*$

$\texttt{newaccount}$

$IV', \{ranb_2, \{RES\}_{SHA1}, RES\}_{K_{a,b}}$

El protocolo utiliza las mismas protecciones que los protocolos propuestos en otros capítulos. Los relojes de los UbiTerm tienen que estar razonablemente sincronizados para que las marcas de tiempo no impidan la comunicación. El *nonce* N_1 asegura que el reenvío del mensaje por parte de un adversario no tenga efecto.

El campo *tuple* del mensaje de respuesta contiene la descripción de la nueva contraseña. Por ejemplo:

```
proto=p9sk1 dom=a-shad user=b !password=di43fdwp
```

El campo *address* sirve para asegurar que el cliente obtiene la dirección correcta de Ta^*. Si se fiara sólo de las cabeceras del mensaje de respuesta, el protocolo estaría expuesto a ataques de suplantación de direcciones de red (*spoofing*). Este campo sirve para actualizar la base de datos `ndb` de Tb^* con la dirección del Ta^*, que será el servidor de autenticación para el dominio `a-shad`.

Si el agente principal de B valida el mensaje de respuesta, entonces actualiza la contraseña en su *anillo de secretos*.

6.7.4. Control de acceso

El control de acceso se basa en una base de datos de usuarios y roles. En la implementación, la base de datos es un fichero llamado `shadroles` que se encuentra en el sistema de ficheros del UbiTerm, situado en el directorio `$home/lib`. El fichero se encuentra dividido en dos partes fundamentales:

- Los roles que se pueden asignar a los usuarios. Se pueden crear tantos roles como se necesiten. Un rol consiste en un nombre mas una lista de dispositivos a los que tiene acceso. Los roles se definen mediante un antecedente y un consecuente separados por el carácter `'='`.

  El antecedente de un rol debe comenzar con la cadena `role`, seguida de su nombre. El consecuente está formado por una lista de servidores de ficheros a los que puede acceder dicho rol, separados por espacios en blanco.

  Después del nombre de un servidor de ficheros se pueden asignar modificadores que restringen el modo de apertura de los ficheros de la interfaz del dispositivo. El modificador puede ser `-R` ó `-W`, que significan restricción de operaciones de lectura y de escritura sobre los ficheros que exporta el sistema de ficheros. En caso de que el usuario que accede tenga distintos roles que permiten el acceso a un mismo sistema de ficheros, se aplica el menos restrictivo.

  Después del modificador se puede especificar el nombre de un fichero de control al que se aplica. Si no se especifica un fichero concreto, se aplica a todos los ficheros de la interfaz.

- Los usuarios emparejados junto a los roles que tienen asignados. Un mismo usuario puede tener varios roles asignados. La asignación de roles también se expresa mediante un antecedente y un consecuente separados por `=`. El antecedente lo forma únicamente el nombre del usuario. El consecuente lo forma una lista de roles, que se han tenido que definir previamente. También se puede usar el carácter comodín `*` para indicar que ese usuario

puede acceder a todos los sistemas de ficheros exportados mediante los mecanismos que ofrece SHAD.

```
role input  =  kbdfs micfs scanfs camerafs mimiofs -W
role output  =  mfs -Wvolume printerfs voicefs
role omero = omero
role lab = hxfs x10fs -Wpwr:124term -Rwho:outside camerafs -R
role guest = projectorfs  mousefs

paurea  =  output lab
katia  =  input output
foo  =  guest
nemo  =  input lab
elf  =  *
```

Cuadro 6.2: *Ejemplo de un fichero* `shadroles` *para el usuario* `esoriano`

El cuadro 6.2 muestra un ejemplo del fichero `shadroles`. El fichero de ejemplo muestra las dos secciones separadas por una linea en blanco.

En la primera sección se definen cinco roles. El rol `input` ofrece acceso a los dispositivos de entrada del usuario, tales como el ratón y teclado (`kbdfs`), micrófono (`micfs`), cámaras (`camerafs`) o pizarras electrónicas [109] (`mimiofs`). Los usuarios que tienen asignados este rol sólo pueden realizar operaciones de lectura sobre cualquier fichero de control de la pizarra electrónica, ya que su sistema de ficheros tiene asociada una restricción `-W`.

`Output` permite el acceso a dispositivos de salida, como los sistemas de audio (`mfs`) o los sintetizadores de voz (`voicefs`). El fichero de control de volumen de audio (cuyo nombre es `volume`) tiene la restricción de que sólo puede leerse. Por consiguiente, los usuarios que tengan el rol `output` pueden obtener el estado del volumen pero no pueden modificarlo.

El rol `omero` da acceso a los interfaces gráficos, por ejemplo para poder tomar prestada una columna de una pantalla perteneciente a `esoriano` y replicar allí los elementos que el usuario desee (notificador de correo, etc.).

El cuarto rol, `lab`, da acceso a los servicios de contexto que pertenecen al usuario `esoriano`, por ejemplo el sistema de ficheros de X10 [17; 19] o las balizas HX [19]. En el caso de los dispositivos de X10, los usuarios con rol `lab` no pueden modificar el estado del interruptor que controla la alimentación del terminal de la oficina 136 (`pwr:136term`) ni pueden consultar el estado del sensor que detecta si hay alguien delante de la puerta del despacho (`who:outside`).

Por último, `guest` permite al usuario usar los proyectores de `esoriano` (`projectorfs`) y sus ratones.

En la sección dedicada a la asignación de roles a los usuarios se puede definir el control de acceso personalizado. De esta forma, el dueño de los recursos puede modelar su noción de confianza en los usuarios con los que ha emparejado su UbiTerm. Por ejemplo, el usuario `esoriano` confía plenamente en el usuario `elf`, y le permite acceder a todos los dispositivos que exporta mediante el comodín `*`. Sin embargo, su confianza en el usuario `foo` es mucho menor, y por esa razón sólo le permite acceder a los dispositivos destinados a los visitantes del espacio inteligente.

Como hemos remarcado anteriormente, los sistemas de ficheros de Plan B controlan el acceso mediante listas de acceso (ACL) tradicionales. Hemos modificado la biblioteca de control de acceso del sistema para añadir el nuevo esquema. Los dispositivos que se exportan mediante SHAD tienen como dueño al usuario que arranca el servidor de ficheros (normalmente el mismo que arranca la máquina, su dueño) y como grupo al grupo `shad`.

Los permisos pertenecientes al grupo son los que controlan el acceso de un usuario de SHAD. Estos permisos se generan dinámicamente para el usuario cuando intenta acceder al recurso.

De esta forma, el control de acceso se aplica en dos capas:

1. El control de acceso personalizado para el usuario, basado en los roles que tiene asociados en el fichero `shadroles`. Cuando un usuario intenta usar un fichero exportado por SHAD, lo primero que hace la biblioteca es resolver los roles asociados. Después comprueba si esos roles le permiten

acceder al sistema de ficheros. Si no se lo permiten, se trata al usuario como si no estuviera incluido en el grupo `shad`. Por tanto se le aplican los permisos que se aplican a *los otros* (quienes no son el dueño ni están en su grupo).

2. Si el usuario sí tiene un rol que le permite acceder al recurso, se le aplican los permisos que aparezcan para el grupo `shad`. Siempre se escoge el rol menos restrictivo en el caso de que varios permitan el acceso. Los permisos para el grupo se generan dinámicamente, según las restricciones que tengan asociados los ficheros de la interfaz del dispositivo para el rol del usuario y la propia semántica de la interfaz (puede haber ficheros para los que no hay implementada la lectura o la escritura porque no es semánticamente adecuado).

```
; echo $user
paurea
; mount /srv/vol /n/audio '/devs/audio user=esoriano loc=124'
; cd /n/audio
; ls -l
---w--w---- M 95 esoriano shad 0 Jun 28 13:32 audio
--rw-r----- M 95 esoriano shad 0 Jun 28 13:32 volume
; cat volume
audio out 47
treb out 0
bass out 0
speed out 44100
; echo audio out 50 > volume
echo: can't open volume 'volume' permission denied
; cp /n/music/heavy/ACDCHard.mp3 audio &
;
```

Cuadro 6.3: *Ejemplo de acceso a un dispositivo de audio.*

El cuadro 6.3 muestra un ejemplo de uso de un dispositivo. Supongamos el fichero `shadroles` mostrado en el cuadro 6.2. El usuario `paurea` quiere montar algún sistema de ficheros `mfs` que exporte un dispositivo de audio que se

encuentre en la oficina 124 y que pertenezca al usuario `esoriano`. Para realizar el montaje el usuario necesita autenticarse ante el servidor de autenticación del dominio `esoriano-shad`. El agente SHAD realiza la autenticación ante el servidor de autenticación de forma totalmente transparente para el usuario. Para usar el dispositivo, `paurea` se sitúa en el directorio donde ha montado el sistema de ficheros del dispositivo de audio.

El dispositivo de audio tiene una interfaz que se compone de dos ficheros de control: `audio` y `volume`. Para reproducir audio, sólo es necesario escribir en el fichero `audio` un flujo en formato MP3. El fichero `volume` sirve para ajustar el volumen del dispositivo escribiendo cadenas de caracteres que indican el volumen (de 0 a 100).

El usuario `paurea` puede obtener el estado de volumen del dispositivo de audio, pero no puede cambiarlo porque, para él, el grupo `shad` no tiene permisos de escritura sobre el fichero `volume`. Eso es consecuencia de que el rol `output` tenga asociada una restricción `-Wvolume`.

Para usar el dispositivo de audio, se copia una canción en formato MP3 sobre el fichero `audio`. Como el control de acceso ha puesto para `paurea` los permisos grupo `shad` con el modo de escritura activado, se puede reproducir el fichero en el dispositivo.

6.7.5. Esquema de funcionamiento

Con el fin ilustrar el funcionamiento global del prototipo, describiremos un caso de uso. Supongamos que el usuario B quiere utilizar desde una aplicación de reproducción de MP3 que ejecuta en su terminal Tb_0 los altavoces que está situados en el despacho donde se encuentra en ese momento. Los altavoces están enchufados al terminal Ta_0 que pertenece al usuario A. En Ta_0 se exporta el audio a través de `mfs`, el sistema de audio de Plan B.

La figura 6.6 ilustra el caso. Las flechas continuas representan los protocolos de SHAD. Las flechas discontinuas representan los ficheros importados por los terminales desde su UbiTerm. Las flechas punteadas representan el protocolo

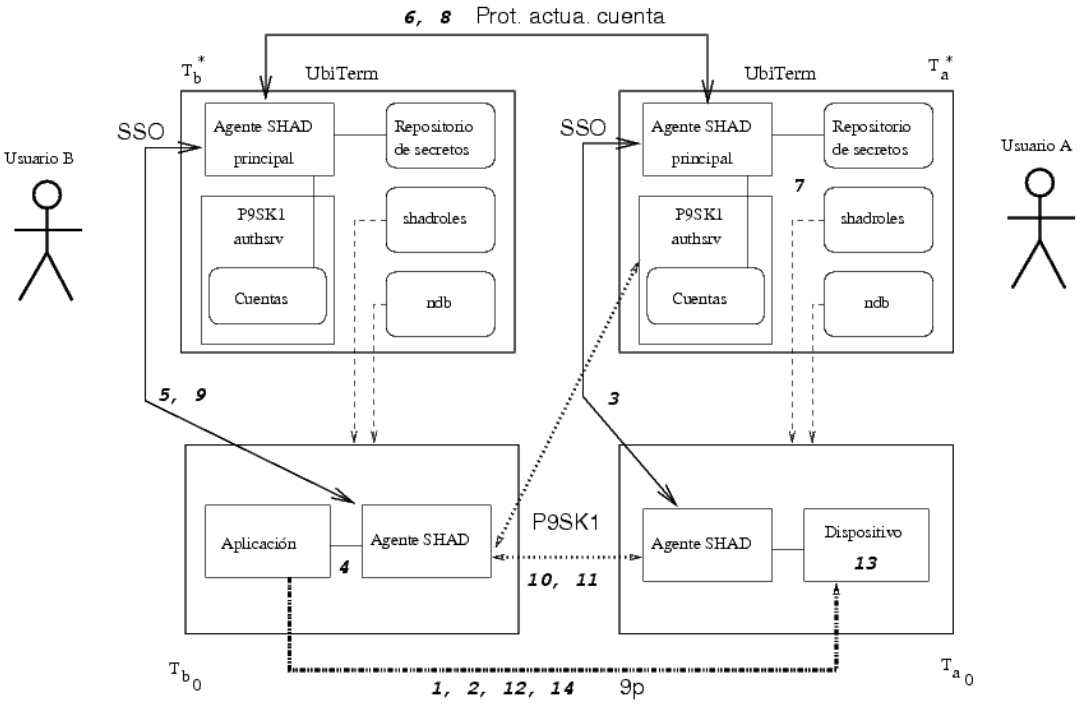

Figura 6.6: *Esquema global del prototipo de compartición de dispositivos.*

P9SK1. Las flechas mixtas representan al protocolo 9P para el uso del dispositivo. Las lineas sin flecha representan flujos entre entidades internas de las máquinas.

Los pasos para utilizar el dispositivo son:

1. La aplicación de reproducción de MP3 solicita a Plan B (mediante `mount`) el montaje de cualquier sistema de audio que se encuentre en este despacho. Se intenta montar el dispositivo de audio del despacho que exporta la máquina Ta_0.

2. Al iniciar el protocolo 9P para montar el dispositivo de audio, `mfs` (el servidor de ficheros que exporta el dispositivo en Ta_0) informa de que necesita autenticación para el dominio `a-shad`.

3. Si el agente de Ta_0 no tiene el secreto para autenticarse ante el dominio de su dueño (`a-shad`), entonces usa el mecanismo de SSO para solicitar dicho

secreto al agente principal. Este paso únicamente es necesario cuando se accede por primera vez a un dispositivo SHAD en una máquina después de reiniciarla. Después queda almacenada en el *anillo de secretos* del agente de la máquina que exporta el dispositivo.

4. En Tb_0, se pide al agente SHAD local que proporcione autenticación. El agente Tb_0 comienza el protocolo P9SK1 con el agente SHAD que ejecuta en la máquina que sirve el sistema de ficheros del dispositivo, Ta_0.

5. El agente de Tb_0 comprueba que no tiene ningún secreto para

   `proto=p9sk1 dom=`*`a-shad`* `user=`*`b`* `!password?`

   Entonces, se ejecuta el protocolo de SSO para obtener dicho secreto de su agente principal, Tb^*. Tb^* comprueba que no tiene dicho secreto y comienza el protocolo de actualización de cuenta. Antes, pedirá confirmación a través de Oshad si es necesario.

6. El agente principal de B ejecuta el protocolo de actualización de cuenta con el agente principal de A, Ta^*.

7. Ta^* valida el mensaje y actualiza la cuenta. Antes, pide la confirmación a través de Oshad si es necesario.

   Crea una nueva contraseña aleatoria y la escribe en las cuentas de su servidor de autenticación P9SK1. Para que este paso prospere, los usuarios deben estar emparejados.

8. Ta^* responde a Tb^*, proporcionándole la nueva contraseña.

9. Tb^* almacena el secreto en su anillo de secretos, y después responde a la petición de SSO que envió Tb_0.

10. Para llevar a cabo el protocolo P9SK1, el agente Tb_0 debe contactar con el servidor de autenticación del dominio `a-shad`. Para ello consulta la base de datos `ndb` y adquiere la dirección de Ta^*, el UbiTerm de A.

11. Se lleva a cabo el protocolo P9SK1 y se autentica a la aplicación en nombre de B. La conexión 9P queda autenticada.

12. La aplicación intenta abrir en modo escritura el fichero de control del sistema de ficheros del dispositivo necesario para reproducir el audio (cuyo nombre es `audio`).

13. En ese momento, el sistema de ficheros realiza el control de acceso. Las rutinas de control de acceso consultan los roles del usuario y comprueban si alguno de ellos permite el acceso al servidor de ficheros `mfs`. Si hay varios que lo permiten, se escoge el menos restrictivo. Se aplica el control de acceso en base a las restricciones del rol (restricciones de lectura o escritura). Supongamos que el usuario B tiene asignado un rol que permite el acceso en modo escritura al fichero `audio`. Entonces, las rutinas permiten la apertura del fichero.

14. Se ha superado el control de acceso. La aplicación empieza a escribir el flujo de audio en formato MP3 en el fichero de control, y el contenido del fichero empieza a sonar por los altavoces del despacho.

Si los usuarios no desean confirmaciones, los pasos 3, 6, 7, 8 y 9 no son necesarios cuando el terminal Tb_0 ya haya accedido anteriormente a algún dispositivo exportado por SHAD perteneciente al usuario A y la cuenta no haya caducado. Si el agente SHAD local ya tiene en su *anillo de secretos* la contraseña para el dominio, primero prueba a realizar la autenticación con ella. En caso de que la autenticación falle, el agente local pide al agente principal que solicite una actualización de la cuenta y se ejecutarían los pasos 6, 7, 8 y 9.

6.7.6. Confirmaciones

El estatus de las confirmaciones se pueden especificar en la tupla que representa a la *clave de emparejamiento* en el repositorio de secretos. Se incluyen dos nuevos atributos llamados `serverconfirm` y `clientconfirm`:

`proto=shad type=ubiterm propietor=`*Usuario* `clientconfirm serverconfirm !secret=`$K_{a,b}$

Si el atributo `serverconfirm` está presente, cada vez que el UbiTerm emparejado empiece el protocolo de actualización de cuenta, tendrá que haber una confirmación explícita en la interfaz gráfica Oshad que ejecuta en el UbiTerm del dueño. Si la cuenta del usuario se actualiza justo después de su autenticación, la confirmación será necesaria cada vez que se cree una conexión 9P.

La confirmación en el UbiTerm del usuario que intenta usar el dispositivo es necesaria si en la tupla de emparejamiento con el dueño del dispositivo aparece el atributo `clientconfirm`. Si los agentes SHAD comunes no almacenan en su *anillo de secretos* los secretos P9SK1 de dominios SHAD de usuarios cuya tupla de emparejamiento requiera confirmación, la confirmación será necesaria cada vez que se establezca una conexión 9P.

Si los atributos no están presentes, la confirmación no es necesaria en ningún caso.

6.7.7. Soporte de desconexiones

Los terminales del usuario deben tener acceso al fichero `shadroles`. Para ello, los terminales montarán el fichero `shadroles` del UbiTerm. Cuando el UbiTerm no esté presente en el sistema, no se podrá aplicar el control de acceso a un dispositivo exportado mediante SHAD. Hay que tener en cuenta que tampoco se podría llevar a cabo la autenticación ante el dominio SHAD porque el servidor de autenticación P9SK1 ejecuta en el UbiTerm.

Sin embargo, **sí se soporta la desconexión del UbiTerm** porque las aplicaciones que ya estaban utilizando un dispositivo pueden seguir haciéndolo. Esas aplicaciones ya se autenticaron y el control de acceso se produjo cuando abrieron el fichero de control necesario del dispositivo en el modo adecuado (lectura, lectura/escritura, etc.). Entonces, pueden seguir realizando lecturas/escrituras en el fichero de control del dispositivo.

No obstante, estas aplicaciones no podrán seguir usando el dispositivo en ausencia del UbiTerm del dueño en el caso de que se cierre la conexión 9P entre la aplicación cliente y el servidor de ficheros del dispositivo, ya que sería necesario autenticarse de nuevo ante ese dominio. La aplicación cliente tampoco podría cerrar el fichero y volver a abrirlo de nuevo, ya que sería necesario aplicar el control de acceso y el fichero de `shadroles` no se encuentra disponible.

De esta forma soportamos la desconexión del UbiTerm, pero de una forma controlada. Las aplicaciones pueden seguir usando el dispositivo de la forma en que se les permitió cuando abrieron su fichero de control, pero no pueden realizar otras tareas mientras el dueño esté ausente.

Si fuera necesario permitir nuevos accesos en ausencia del UbiTerm para las aplicaciones cuya conexión 9P ya estuviera autenticada, se podrían mantener copias locales del fichero `shadroles` en los terminales. Una conexión 9P autenticada podría abrir y cerrar ficheros de control del dispositivo en ausencia del UbiTerm del dueño.

Pero también se podrían dar situaciones de incoherencia entre el fichero `shadroles` del UbiTerm y las copias locales de los terminales. Las copias se tendrían que sincronizar periódicamente, y los terminales tendrían que usar el fichero del UbiTerm siempre que fuera posible mediante uniones de directorios [28]. Esto sería posible porque Plan B tolera fallos en los elementos de las uniones que se sitúan encima de otros [21; 20].

El mismo problema ocurre con la base de datos `ndb` y el UbiTerm del usuario cliente. La aplicación cliente soporta la desconexión del UbiTerm si la conexión 9P ya está autenticada. Sin embargo, no se pueden establecer nuevas conexiones 9P sin poder obtener la dirección actualizada del servidor de autenticación para el dominio del dueño del dispositivo.

6.7.8. Revocación

Un usuario puede revocar el acceso a sus dispositivos mediante el fichero `shadroles`. Basta con eliminar (o comentar mediante un carácter `#`) la linea

perteneciente a un usuario. En el momento en que el usuario borra un usuario del fichero de roles, este ya no puede acceder a ningún dispositivo exportado mediante SHAD. No obstante, para eliminar totalmente a un usuario, se tiene que:

- Eliminar la *clave de emparejamiento* con ese usuario del repositorio de secretos.
- Eliminar la cuenta en el servidor de autenticación P9SK1 del UbiTerm.
- Eliminar la linea correspondiente a este usuario en el fichero `shadroles`.

Estas tareas son fácilmente automatizables mediante un script de shell.

Hay un caso en el que la revocación no es inmediata. Como hemos visto anteriormente, cuando la conexión se ha establecido y el control de acceso se ha superado (se ha abierto el fichero deseado en el modo adecuado) , la aplicación cliente puede seguir leyendo y escribiendo en él aunque se elimine por completo al usuario (mediante los tres pasos descritos anteriormente).

Para solucionar este problema, se puede añadir un cuarto paso que consista en cerrar cualquier conexión que tenga establecida un dispositivo exportado mediante SHAD con un cliente autenticado como el usuario eliminado. Para ello habría que modificar el sistema operativo para que cierre las conexiones 9P que tienen abiertas los sistemas de ficheros exportados mediante SHAD. Nótese que esta limitación la tiene Plan B y la hereda desde su ancestro más remoto, UNIX.

6.8. Experiencia de uso

Se ha configurado un escenario en el que dos usuarios ejecutan servidores de ficheros para exportar sus ficheros a través de SHAD y utilizar las bibliotecas de control de acceso modificadas, el fichero `shadroles` y la base de datos `ndb` que define sus dominios de autenticación.

Para realizar las pruebas, se ha modificado el prototipo NetFactotum para que pueda actualizar las cuentas en los servidores de autenticación de los UbiTerms, y se han añadido los dos protocolos descritos en este capítulo.

Para las pruebas se han configurado los UbiTerms de los usuarios sobre dos máquinas virtuales VMWare [191] ejecutando sobre un ordenador Pentium 4 (cada una con 256 Mb de memoria). Los terminales son ordenadores Pentium 4 con 1 Gb y 512 Mb de memoria respectivamente. Las máquinas están conectadas a través de una red a 100 Mbit/s.

El resultado de las pruebas ha sido satisfactorio en lo que respecta al funcionamiento de la arquitectura. Los dos usuarios son capaces de acceder a los recursos del otro de una forma controlada.

Las medidas demuestran que el prototipo es lo suficientemente rápido como para que un usuario corriente no aprecie la latencia provocada por el protocolo de autenticación y los mecanismos de control de acceso. El cuadro 6.4 muestra el tiempo medio para montar un sistema de ficheros remoto, abrir un fichero en modo lectura, copiar su contenido (12 bytes) en un fichero local, cerrar el fichero y desmontar el sistema de ficheros. El tiempo que mostramos en las tres columnas es la media aritmética de cincuenta ejecuciones.

La primera columna muestra el tiempo medio para realizar las tareas en un sistema de ficheros sin autenticación y con el control de acceso de Plan 9, basado en ACLs. La segunda columna muestra el tiempo medio para realizar las tareas usando el prototipo de SHAD cuando no es necesario ejecutar el protocolo de actualización de cuenta para el usuario cliente (típicamente, cuando no se requieren confirmaciones en los UbiTerms). La tercera columna muestra la media en el peor caso, esto es, cuando se ejecuta el protocolo de actualización de cuenta entre los UbiTerms de los usuarios involucrados.

Se puede observar que el retardo es inapreciable para un usuario común. Nótese que la primera columna muestra el tiempo cuando no se necesita autenticación, y por tanto, no se ejecuta ningún protocolo (P9SK1 o SHAD) antes de realizar las operaciones sobre el fichero.

	Sin autenticación y ACLs	Con autenticación SHAD/P9SK1 y roles (mejor caso)	Con autenticación SHAD/P9SK1 y roles (peor caso)
Tiempo (seg)	0.024	0.207	0.308

Cuadro 6.4: *Tiempo medio para montar, utilizar y desmontar un sistema de ficheros.*

Al igual que en el caso de la arquitectura de compartición de terminales, la arquitectura de compartición de dispositivos no se ha usado intensivamente (al contrario que la arquitectura de SSO). Este hecho se puede achacar a las necesidades de nuestro entorno inteligente, y a la disposición de nuestro hardware y nuestras oficinas. Se tratan de oficinas fijas en las que los usuarios tienen disponible todo el hardware que necesitan, y además les pertenece. En estas condiciones, la arquitectura de compartición de dispositivos puede ser prescindible.

Sin embargo, creemos que esta arquitectura puede ser realmente útil en entornos más dinámicos en los que los usuarios carezcan de hardware propio y tengan que compartir sus recursos [57]. También puede resultar más útil en oficinas compartidas en las que los usuarios no tienen un puesto fijo, y por tanto estos no puedan disponer de una infraestructura que les pertenezca (sistemas de audio, ratones, pantallas, etc.).

Capítulo 7

Conclusiones y trabajo futuro

Este capítulo tiene como propósito exponer las principales conclusiones, aportaciones del trabajo efectuado y el posible trabajo futuro.

7.1. Conclusiones

7.1.1. *Single Sign-On*

En SHAD proponemos una arquitectura de *Single Sign-On* (SSO) que **por primera vez permite al usuario que trabaja físicamente con múltiples máquinas acceder al entorno de computación proporcionando una única autenticación explícita**. Los sistemas de SSO propuestos en la literatura no lo hacen. Dichos sistemas ofrecen SSO *por máquina*, mientras que nosotros ofrecemos un sistema de SSO real para el espacio inteligente.

Una gran parte de los trabajos dedicados a autenticación en computación ubicua ofrecen autenticación para propósitos específicos; son soluciones a medida para un problema de autenticación concreto. **Nosotros ofrecemos al usuario la autenticación automática tanto para los servicios del espacio inteligente como para los servicios comunes de un sistema abierto e Internet**.

La mayoría de las arquitecturas propuestas en la literatura no permiten autenticaciones en entornos fraccionables. Nosotros sí. El hecho de centrar el servicio de SSO en el humano a través de un dispositivo móvil permite al usuario seguir disfrutando de la autenticación automática en particiones de red aisladas del resto del sistema, ya que siempre lleva consigo su servidor personal de SSO: el agente principal de SHAD.

Las arquitecturas SSO existentes que sí funcionan en entornos fraccionables provocan obstrucción al usuario porque normalmente se basan en dispositivos que deben ser insertados en lectores, por ejemplo Smart Cards. **Nuestra arquitectura no provoca obstrucción al usuario**, a no ser que este la configure explícitamente para ello.

Otra diferencia importante es que **nuestra arquitectura se basa en la**

cooperación entre las distintas máquinas de propósito general del usuario que no requieren de hardware específico de autenticación.

SHAD soporta fallos del servidor personal que el usuario lleva consigo. La cooperación de los agentes SHAD mediante un protocolo *Peer-to-Peer* para compartir sus secretos cuando el servidor personal no esta disponible permite que las aplicaciones del usuario puedan autenticarse automáticamente en su ausencia. Comúnmente, los sistemas que dependen de un servidor central de claves no soportan fallos del mismo.

La arquitectura no requiere administración compleja y centralizada para todos los usuarios, ya que cada usuario mantiene su propio repositorio de secretos y es capaz de configurarlo como desee. Por lo tanto, no es necesario un administrador de sistema para mantener la arquitectura operativa. La mayoría de los esquemas propuestos en la literatura sí lo requieren.

El diseño de la arquitectura que proponemos no depende de middleware debido a su diseño basado en sistemas de ficheros. Este factor hace que **las aplicaciones existentes puedan usar el servicio de SSO sufriendo unas mínimas modificaciones**, independientemente del lenguaje de programación en el que estén implementadas y de que se basen en cualquier tipo de middleware.

El sistema es altamente configurable y flexible. Los usuarios que valoran más la comodidad y no desean que se les interrumpa son capaces de configurar su agentes para compartir los secretos sin apenas requerir confirmaciones. Por el contrario, los usuarios que valoran más la seguridad de sus secretos que la comodidad de uso, pueden configurar el sistema para pedir confirmaciones y restringir el uso de los secretos en base al contexto físico del entorno. En general, los usuarios prefieren una mezcla de confort y seguridad, y nosotros les ofrecemos mecanismos para ajustar sus preferencias y definir sus propias políticas.

Nuestro sistema es trazable y predecible. Las acciones que toma el agente de seguridad se basan el las restricciones que el usuario asigna a sus secretos. **El sistema no se basa en estimaciones de ningún tipo**, y en caso

de no disponer de la información necesaria para evaluar una acción, siempre toma la opción más restrictiva. Estos factores influyen en que la confianza del usuario en el sistema se vea incrementada.

La evaluación del prototipo Netfactotum muestra que **es lo suficientemente rápido como para que el usuario no note la diferencia entre usar un sistema de SSO convencional y SHAD**. El prototipo que hemos implementado para ejecutar en un dispositivo móvil sirve como prueba de concepto, y muestra que la aproximación es cómoda y que el uso de un agente SHAD completo de SSO en un dispositivo de este tipo es perfectamente plausible.

Hasta donde conocemos, estas propiedades hacen de la arquitectura SSO de SHAD única y diferente de las distintas aproximaciones para SSO presentadas en la literatura.

7.1.2. Compartición de dispositivos

En SHAD ofrecemos una forma sencilla de compartición de dispositivos dentro de un espacio inteligente, en el que los humanos tienen un número indeterminado de máquinas que les pertenecen, que a su vez disponen de distintos dispositivos conectados.

La arquitectura de compartición de dispositivos se apoya en la arquitectura de SSO, y por tanto se le pueden atribuir las cualidades descritas anteriormente.

Centramos la seguridad en el humano a través de un dispositivo móvil (UbiTerm) que realmente actúa su representante en el sistema. Plasmar la noción de confianza en otros usuarios relacionando UbiTerms y asignando roles a sus dueños es simple e intuitivo.

El diseño centrado en el humano que proponemos hace que la configuración de los dispositivos sea sencilla y se base sólo en cuatro acciones básicas:

1. Asignar un secreto a una máquina cuando se añade al sistema como una acción más de configuración, de la misma forma que se le asigna por ejemplo un nombre.

2. Emparejar el UbiTerm con los UbiTerms de los usuarios en los que se confía.

3. Asignar roles a los usuarios con los que se ha emparejado el UbiTerm.

4. Asociar el acceso a los dispositivos con los distintos roles creados que se han creado.

Los usuarios pueden compartir dispositivos en particiones de red aisladas de servicios centralizados: el uso de la PAN de los usuarios basta para garantizar la autenticación a la hora de acceder a un recurso a través de sus UbiTerms.

Creemos que el prototipo que hemos implementado para la compartición de terminales y dispositivos muestra que la aproximación es aplicable a un espacio inteligente en el que los mecanismos de acceso a los recursos no depende de entidades centralizadas.

La experiencia de uso del prototipo de la arquitectura de SSO indica que la arquitectura de compartición de dispositivos a través de los UbiTerms es posible y ejecutable.

7.2. Contribuciones

La presente Tesis ha aportado las siguientes contribuciones al estado del arte en lo que respecta a las arquitecturas de seguridad aplicadas a sistemas ubicuos:

1. **Una arquitectura *Peer-to-Peer* de *Single Sign-On* real para entornos ubicuos en los que los usuarios trabajan con distintos dispositivos concurrentemente.**

2. **El diseño y la implementación de un prototipo de la arquitectura de *Single Sign-On* real.**

3. **El diseño de una arquitectura de seguridad para la compartición de recursos centrada en el humano que funciona en entornos ubicuos dinámicos, heterogéneos y fraccionables.**

4. **Una aproximación para plasmar la noción de confianza entre humanos de una forma simple e intuitiva.**

5. **El diseño y la implementación de un prototipo de dicha arquitectura de compartición de dispositivos.**

6. **La experiencia de uso (y medidas relacionadas) de las arquitecturas anteriormente citadas durante el periodo de un año.**

7.3. Limitaciones del modelo

7.3.1. Dispositivos móviles de autenticación

El problema de usar dispositivos móviles para autenticar al usuario ya ha sido tratado en capítulos anteriores. Hay que tener presente el riesgo de perder el dispositivo que controla el acceso a los recursos y la posibilidad de que caiga en manos de una tercera parte.

Pero también hay que tener en consideración las ventajas de que el usuario lleve su agente personal de seguridad con él constantemente. En resumen, el usuario debe evaluar el compromiso entre el riesgo de perder su dispositivo y la comodidad de acceder a los recursos sin obstrucción en cualquier localización y de controlar el acceso a sus recursos de una forma simple e intuitiva.

Para evaluar correctamente esta limitación, hay que tener en cuenta las limitaciones de los sistemas actuales, en los que no hay *Single Sing-On*. La ausencia de una autenticación automática provoca grandes riesgos de seguridad. Por poner un ejemplo, en estos sistemas los usuarios tienden a escoger contraseñas fáciles de recordar, e incluso a apuntarlas en notas de papel o libretas. Otro ejemplo es que, en un sistema en el que el usuario tiene que introducir contraseñas constantemente, un adversario puede engañarle para que introduzca una clave en un terminal *falso* (un terminal no confiable). Esta

situación no se puede dar en nuestra arquitectura, ya que a la hora de usar un terminal cedido, éste se autentica a través su agente principal.

7.3.2. Secretos almacenados en el hardware

Almacenar la clave de un terminal en su propio hardware (en el caso de los prototipos que se han implementado, en la NVRAM del equipo) supone un riesgo si un adversario tiene acceso físico a dicho hardware.

Por lo general, se asume que cuando el adversario tiene acceso físico al hardware, intentar asegurarlo es inútil. Este principio se conoce como *The Big Stick Principle* [179], y otros esquemas de seguridad propuestos en la literatura para asegurar entornos inteligentes [160] y entornos distribuidos [41] se basan en él.

Por otra parte, el dispositivo elegido para ser el UbiTerm no necesita almacenar ningún secreto en claro en su hardware, ya que el usuario se autentica explícitamente ante él, y por tanto es capaz de descifrar su clave secreta.

Se considera el uso futuro de algún tipo de hardware resistente a alteraciones (*Tamper Resistant*) que se base en métodos de retos (*Challenge-Response*) para la autenticación de los terminales, por ejemplo algún tipo de tarjetas Smart Card. En ese caso, aún teniendo acceso físico al hardware, el atacante no podría robar el secreto para poder autenticar una máquina diferente como un terminal del usuario y así poder suplantarle para adquirir sus secretos. Nótese que en este caso el atacante sí podría robar el hardware de autenticación para usarlo en otra máquina (mediante la que podría acceder a los secretos del usuario), o simplemente utilizar la máquina actual para acceder a los servicios mediante los secretos del usuario disponibles para dicha máquina.

7.3.3. Elección de restricciones, atributos y roles

El acceso a los secretos desde las distintas máquinas del usuario se basa en los atributos que este asigne a cada uno de sus secretos. De esta forma, la seguridad

del sistema depende de las elecciones que tome el usuario a la hora de configurar sus secretos.

El usuario puede realizar una elección equivocada a la hora de asignar las restricciones de sus secretos y hacer el sistema inseguro. Hay que tener en cuenta que la seguridad de la mayoría de los sistemas están sujetos a este problema. Por ejemplo, en un sistema basado en contraseñas se pueden elegir contraseñas inseguras, tales como fechas o nombres propios. Los sistemas que obligan a los usuarios a elegir contraseñas seguras o fuerzan una contraseña aleatoria, están sujetos a que el usuario la anote en un papel o en una pizarra, haciendo el sistema más vulnerable si cabe. Otro ejemplo son los usuarios de Smart Cards, que puede olvidar sus tarjetas en el lector. Los ejemplos son numerosos.

En general, la seguridad se basa en la buena práctica de los usuarios del sistema.

7.4. Trabajo futuro

La linea de trabajo futuro principal es el diseño de un sistema de anuncios sobre vulnerabilidades y de revocación de claves del usuario, tanto para las claves relacionadas con la compartición de dispositivos como para las claves relacionadas con el acceso a los recursos comunes en el entorno.

Como ya hemos comentado en la sección anterior, se considera una linea de trabajo futura para integrar hardware resistente a alteraciones que ofrezca algún método de autenticación que no requiera almacenar claves privadas en claro en ningún dispositivo que se pueda leer.

Por último, otra linea de trabajo podría trata de añadir más mecanismos a SHAD. Podríamos añadir nuevos servicios del espacio inteligente a las restricciones para ceder secretos en la arquitectura, tales como sistemas de visión para la detección de intrusiones [136]. También podríamos integrar otros tipos de hardware de para confirmar las operaciones del agente de SHAD, como dispositivos RFID de contacto o sensores biométricos.

Por último, se podría explorar el uso de tecnología de transferencia de información a través del cuerpo humano [133] para asignar los secretos a las máquinas en tiempo de arranque con solo tocarlas, con el fin de no almacenar secretos de ningún tipo en ellas.

Apéndice A

Herramientas criptográficas

A.1. Introducción

En este apéndice presentamos las herramientas criptográficas utilizadas en SHAD, así como justificamos la forma de usar dichas herramientas para asegurar los distintos mensajes de los protocolos.

A.2. El algoritmo AES

Todos los protocolos descritos en SHAD hacen uso del actual estándar de cifrado simétrico AES (*Advanced Encryption Standard*) [121], que se basa en el algoritmo de cifrado Rijndael [43].

AES reemplazó al algoritmo DES como estándar de cifrado simétrico en enero de 1997 después de competir contra otros algoritmos en un concurso realizado por el NIST (*National Institute of Standards and Technology*) de los EE.UU.

AES es un algoritmo de cifrado en bloque. Un algoritmo de cifrado en bloque se puede definir como una función parametrizada por una clave que convierte un bloque de n bits de texto en claro en un bloque de n bits de texto cifrado [106].

Los protocolos de SHAD utilizan AES en modo CBC (*Cipher Block Chaining*). Este modo consiste en cifrar los bloques aplicándoles antes una operación XOR con el texto cifrado del bloque anterior. Antes de cifrar el primer bloque, se aplica un XOR con un vector de inicialización. Ese vector se suele generar de forma aleatoria.

Para descifrar un bloque, se aplica la función inversa y se aplica un XOR con el bloque cifrado anterior. El tamaño de los datos cifrados debe ser múltiplo de la longitud de bloque.

Las ventajas más notables de este modo son [164, Capítulo 9]:

- Los patrones que pueden existir en el texto en claro quedan ocultos por las operaciones XOR con el texto cifrado anterior.
- Se pueden cifrar múltiples mensajes con una misma clave.

- El texto en claro es relativamente difícil de modificar.
- La velocidad del cifrado depende sólo de la velocidad del algoritmo de cifrado en bloque.

Las desventajas son:

- No se puede paralelizar el cifrado (el descifrado sí). No es posible cifrar un bloque n sin cifrar el bloque $n-1$.
- Un bit erróneo en el texto cifrado supone un bloque entero más un bit erróneos en el texto claro. Si se modifica un i bit del bloque n en el texto cifrado, el bloque n del texto claro es erróneo. También lo es el bit i del bloque $n+1$.
- No se puede recuperar de un error de sincronización. Si se pierde o se duplica un bit en la transmisión del texto cifrado, no se puede conseguir el texto en claro a partir de dicho error.

Las necesidades de SHAD se ajustan a las propiedades citadas.

AES es seguro, y no se contemplan ataques de fuerza bruta con tecnología actual (ni de un futuro cercano), ni siquiera para texto cifrado con la longitud de clave mínima (128 bits) [181, Capítulo 5]. Para los protocolos de SHAD se ha elegido una longitud de clave de 256 bits.

Hasta el momento, el único ataque efectivo que se ha realizado contra AES es relativo a su implementación (*Side Channel Attack*) [131].

El algoritmo es eficiente y fue ideado para ejecutar en dispositivos con poca capacidad de proceso, por ejemplo en CPUs de 8 bits. En [43] se aportan medidas de rendimiento que corroboran este hecho. Una implementación de AES para un procesador Motorola 6805 de 8 bits con una frecuencia de reloj de 4 Mhz es capaz de cifrar a 54 Kbit/s. Un procesador Pentium III de 32 bits con una frecuencia de reloj de 800 Mhz es capaz de alcanzar los 426 Mbit/s.

A.3. El algoritmo SHA-1

Los protocolos de SHAD utilizan un algoritmo *hash* seguro para obtener resúmenes del contenido de los mensajes, y así poder comprobar que los datos útiles del mismo no han sido modificados.

Una función *hash* consiste en resumir unos datos, que se denominan pre-imagen, en una secuencia fija de bits.

La función *hash* segura que se ha elegido para los protocolos es el algoritmo SHA-1[120]. El algoritmo SHA-1 genera un resumen de 160 bits de longitud.

SHA-1 asegura las tres propiedades básicas que debe ofrecer una función *hash* segura[106]:

- Primera resistencia pre-imagen: para cualquiera de los posibles resúmenes que se pueden obtener de la función, no es computacionalmente factible encontrar la pre-imagen que le corresponde.

- Segunda resistencia pre-imagen: no es computacionalmente factible encontrar una pre-imagen que genere el mismo resumen que otra pre-imagen dada.

- Resistencia a la colisión: no es computacionalmente factible encontrar dos pre-imágenes distintas que generen el mismo resumen. La diferencia entre esta propiedad y la anterior es que las dos pre-imágenes son libres (en la anterior, una venía dada).

SHA-1 se considera una función *hash* segura hoy en día, aunque Xiaoyun et al. [192] consiguieron romper la tercera propiedad en 2^{69} cálculos de resúmenes, cuando el ataque de fuerza bruta requiere 2^{80} cálculos. Actualmente el ataque ha mejorado hasta conseguir romper la propiedad de resistencia a colisiones en 2^{63} cálculos [163]. Nótese que nuestros protocolos utilizan SHA-1 junto AES, y que la propiedad objetivo del ataque es la de resistencia a la colisión.

A.4. Uso de los algoritmos

Los mensajes correspondientes a los protocolos de SHAD están formados por una cabecera en texto claro, el vector de inicialización utilizado para el cifrado, y una parte cifrada:

CABECERA	IV	DATOS CIFRADOS

La parte cifrada contiene la carga útil del mensaje junto con otros datos. Esta parte sigue este esquema:

96 *bits*	160 *bits*	*variable*
DATOS ALEATORIOS	RESUMEN (CARGA ÚTIL)	CARGA ÚTIL

La **cabecera** contiene los datos necesarios para suponer el remitente, el destinatario y el tipo de mensaje del que se trata. La cabecera se envía en texto claro, por lo tanto su contenido no es confiable: no está autenticado (puede provenir de cualquier parte), no se puede comprobar su integridad y no es confidencial (cualquier entidad que tenga acceso a la red puede ver su contenido). La función de la cabecera es simplemente permitir a los nodos poder discriminar los mensajes que reciben sin tener que gastar tiempo de procesamiento intentando descifrar su contenido. Una vez descifrado el contenido de la carga útil, se deben comparar con los datos de la cabecera. Si no concuerdan, se debe desechar el mensaje.

El **vector de inicialización (IV)** consiste en un vector de la longitud de un bloque AES que se genera aleatoriamente en el origen del mensaje. La longitud de bloque es de 128 bits. Ese vector se utiliza para producir el primer bloque de texto cifrado cuando se utiliza un algoritmo en modo CBC. Es necesario que el receptor del texto cifrado conozca el vector de inicialización que utilizó el remitente para cifrarlo.

Schneier mantiene que el vector de inicialización puede ir en texto claro en el mensaje, no tiene porqué ser secreto [164, Capítulo 9]. Sin embargo, Stallings mantiene que el vector de inicialización debe ser secreto (sólo lo deben conocer el

remitente y el destinatario) [181, Capítulo 3], ya que un atacante puede provocar cambios en los bits del primer bloque cifrado si es capaz de modificar el vector de inicialización adjunto en el mensaje. Schneier mantiene que si el mensaje consta de n bloques y mantenemos el vector de inicialización en secreto, todavía quedarán $n-1$ vectores de inicialización al descubierto, debido a que en modo CBC antes de descifrar el bloque i se debe realizar un XOR con el bloque cifrado $i-1$. En todo caso, el problema reside en que cambiando los bits del vector de inicialización, se pueden conseguir cambios en el primer bloque de texto claro.

Si un adversario modifica un bit en el bloque de texto cifrado i, entonces consigue modificar por completo el bloque de texto claro i y a la vez modificar un único bit en el bloque de texto cifrado $i+1$ [164, Capítulo 9].

Los datos cifrados comienzan con una cantidad de **datos aleatorios**, cuyo contenido ignora el destinatario. El tamaño de estos datos aleatorios es de 96 bits y hacen que la carga útil quede alineada con el tercer bloque de los datos cifrados.

A continuación insertamos el **resumen** SHA-1 de la carga útil (160 bits), que ocupa el resto del primer bloque mas el segundo bloque entero. A partir del tercer bloque comienza la carga útil.

Mediante el resumen se puede comprobar la integridad de los datos contenidos en la carga útil. Cuando el destinatario recibe el mensaje y descifra los datos cifrados, extrae la carga útil y su resumen. Si no es capaz de reproducir el mismo resumen mediante la carga útil recibida, entonces se entiende que se ha violado la integridad de los datos y se descarta el mensaje.

Si un atacante modifica el vector de inicialización, dicha variación no tiene efectos en el contenido útil del mensaje. Pero sí puede afectar al resumen, lo que provocaría la invalidación de la carga útil. En ese caso, el destinatario es capaz de detectar la manipulación del mensaje y descartarlo.

La **carga útil** contiene los datos del protocolo, que incluyen *nonces* y marcas de tiempo que aseguran que el mensaje es fresco y no es fruto de un ataque de repetición (*replay attack*). El uso del contenido de la carga útil se especifica en

la descripción de cada protocolo de SHAD.

No utilizamos MACs (*Message Authentication Code*) adicionales en los mensajes debido a que el propio cifrado de la carga útil proporciona tanto autenticación (debido al contenido de la carga útil: ids de usuario, *nonces*, marcas de tiempo) como confidencialidad al mensaje. Eso es así siempre que la clave utilizada sólo la conozcan los dos extremos de la comunicación. El protocolo no cumple ninguna de las situaciones descritas en [181, Capítulo 11] en las que se recomienda usar MACs de los mensajes en lugar de confidencialidad.

Bibliografía

[1] International Standard ISO/IEC 8802-11. ANSI/IEEE Standard 802.11, 1999.

[2] International Standard ISO/IEC 8802-3. ANSI/IEEE Standard 802.3, 1997.

[3] *Plan 9 Programmer's Manual, Section 5: Plan 9 File Protocol*. AT & T Bell Laboratories, Murray Hill, NJ, fourth edition, 2002. http://www.cs.bell-labs.com/magic/man2html/5/0intro.

[4] A. Abdul-Rahman and S. Hailes. Supporting Trust in Virtual Communities. En *Proceedings of the 33rd International Conference on System Sciences*, páginas 1–9, 2000.

[5] K. Aberer and Z. Despotovic. Managing Trust in a Peer-to-Peer Information System. En *Proceedings of the 9th International Conference on Information and Knowledge Management (CIKM)*, páginas 310–317, 2001.

[6] J. Al-Muhtadi, M. Anand, N. D. Mickunas, and R. Campbell. Secure Smart Homes Using Jini and Sesame. En *Proceedings of the 16th Annual Computer Security Applications Conference*, páginas 77–85, 2000.

[7] J. Al-Muhtadi, D. Mickunas, and R. Campbell. Wearable Security Services. En *Proceedings of the 21st International Conference on Distributed Computing Systems*, páginas 226–232, 2001.

[8] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and N. D. Mickunas. A Flexible, Privacy-preserving Authentication Framework for Ubiquitous Computing Environments. En *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops (IWSAEC 2002)*, páginas 771–776, 2002.

[9] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and N. D. Mickunas. Cerberus: A Context-Aware Security Scheme for Smart Spaces. En *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, páginas 489–496, 2003.

[10] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[11] P. Ashley, M. Vandenwauver, and B. Broom. A Uniform Approach to Securing Unix Applications Using SESAME. En *Proceedings of the 3rd Australasian Conference on Information Security and Privacy, LNCS 1438*, páginas 24–35, Springer-Verlag, 1998.

[12] *Plan 9 Programmer's Manual, Section 5: Attach*. AT & T Bell Laboratories, Murray Hill, NJ, fourth edition, 2002. http://www.cs.bell-labs.com/magic/man2html/5/attach.

[13] *Plan 9 Programmer's Manual, Section 6: Authsrv*. AT & T Bell Laboratories, Murray Hill, NJ, fourth edition, 2002. http://www.cs.bell-labs.com/magic/man2html/6/authsrv.

[14] F. J. Ballesteros, E. M. Castro, G. Guardiola, K. Leal, and P. de las Heras. /net. A Network Abstraction for Mobile and Ubiquitous Computing Environments in the Plan B Operating System. En *Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications*, páginas 112–121, 2004.

[15] F. J. Ballesteros, G. Guardiola, K. Leal, E. Soriano, P. Heras, E. M. Castro, and S. Arévalo. Plan B: Boxes for Networked Resources. *Journal of the Brazilian Computer Society, special issue on Adaptive Software Systems*, 10(1):31-42, 2004.

[16] F. J. Ballesteros, G. Guardiola, E. Soriano, and K. Leal Algara. Omero: Ubiquitous User Interfaces for Ubiquitous Programmers. *Enviado para su publicación. Disponible en http://lsub.org/papers/.*

[17] F. J. Ballesteros, G. Guardiola, E. Soriano, and K. Leal. Traditional Systems Can Work Well for Pervasive Applications. A Case Study: Plan 9 From Bell Labs Becomes Ubiquitous. En *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*, páginas 295–299, 2005.

[18] F. J. Ballesteros, K. Leal, E. Soriano, and G. Guardiola. Omero: Ubiquitous User Interfaces in the Plan B Operating System. En *Proceedings of the 4th IEEE International Conference on Pervasive Services*, páginas 77–83, 2006.

[19] F. J. Ballesteros, E. Soriano, and G. Guardiola. The LS Smart Space for Programmers. *Enviado para su publicación. Disponible en http://lsub.org/papers/.*

[20] F. J. Ballesteros, E. Soriano, G. Guardiola, and K. Leal. Plan B: A Pervasive Computing Environment Without Middleware Designed for Programmers. *IEEE Pervasive Computing.* Futura publicación.

[21] F. J. Ballesteros, E. Soriano, K. Leal, and G. Guardiola. Plan B: An Operating System for Ubiquitous Computing Environments. En *Proceedings of the 4th IEEE International Conference on Pervasive Services*, páginas 126–135, 2006.

[22] J. E. Bardram, R. E. Kjaer, and M. O. Pedersen. Context-aware User Authentication – Supporting Proximity-Based Login in Pervasive Computing. *UbiComp 2003, LNCS 2864*, páginas 107–123, Springer-Verlag, 2003.

[23] P. Barker and S. Kille.*The Cosine and Internet x.500 Schema.* Internet Engineering Task Force: RFC 1274, 1991.
http://www.ietf.org/rfc/rfc1274.txt.

[24] D. J. Barrett, R. Silverman, and R. G. Byrnes. *SSH, The Secure Shell: The Definitive Guide.* O'Reilly, second edition, 2005.

[25] A. Beaufour and P. Bonnet. Personal Servers as Digital Keys. En *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications*, páginas 319–328, 2004.

[26] S. M. Bellovin and M. Merritt. Limitations of the Kerberos Authentication System. *ACM SIGCOMM*, 20(5):119–132, 1990.

[27] Computer Systems Research Group UC Berkeley. *4.4BSD Programmer's Reference Manual.* O'Reilly, AT & T Bell Laboratories, Murray Hill, NJ, first edition, 1994.

[28] *Plan 9 Programmer's Manual, Section 1: BIND.* AT & T Bell Laboratories, Murray Hill, NJ, fourth edition, 2002. http://www.cs.bell-labs.com/magic/man2html/1/bind.

[29] M. Blaze, J. Feigenbaum, and A. D. Keromytis. The Role of Trust Management in Distributed Systems Security. En *Secure Internet Programming: Security Issues for Mobile and Distributed Objects, LNCS 1603*, páginas 185–210, Springer-Verlag, 1999.

[30] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. En *1996 IEEE Symposium on Security and Privacy*, 1996.

[31] The Official Bluetooth® Wireless Info Site. http://www.bluetooth.com/.

[32] B. W. Boehm. A Spiral Model of Software Development and Enhancement. *Computer*, 21(5):61–72, 1988.

[33] L. Bussard and Y. Roudier. Authentication In Ubiquitous Computing. En *Proceedings of Ubicomp 2002, Workshop on Security in ubiquitous computing*, 2002.

[34] M. Cagalj and J. P. Hubaux. Key Agreement in Peer-to-Peer Wireless Networks. *Proceedings of the IEEE*, 94(2):467–478, 2006.

[35] C. K. Hess and R. H. Campbell. A Context File System for Ubiquitous Computing Environments. *Technical Report No. UIUCDCS-R-2002-2285 UILU-ENG-2002-1729*, 2002.

[36] R. Campbell, J. Al-Muhtadi, P. Naldurg, G. Sampemane, and M. Mickunas. Towards Security and Privacy for Pervasive Computing. En *Proceedings of the International Symposium on Software Security, LNCS 2603*, páginas 1–15, Springer-Verlag, 2002.

[37] S. K. Card, T. P. Moran, and A. Newell. *Handbook of Perception and Human Performance*. John Willey and Sons, 1986.

[38] *Common Object Request Broker Architecture: Core Specification*. The Object Management Group, Inc., third edition, 2004. http://www.omg.org/cgi-bin/apps/doc?formal/04-03-01.pdf.

[39] M. Corner and B. Noble. Protecting Applications with Transient Authentication. En *Proceedings of the First ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys'03)*, páginas 57–70, 2003.

[40] Netscape Communications Corp. *The SSL (Secure Sockets Layer) 3.0 Protocol*. Mountain View, CA, USA, 1995.

[41] R. Cox, E. Grosse, R. Pike, D. Presotto, and S. Quinlan. Security in Plan 9. En *Proceedings of the 11th USENIX Security Symposium*, páginas 3–16, 2002.

[42] S. Creese, M Goldsmith, B. Roscoe, and I. Zakiuddin. Research Directions for Trust and Security in Human-Centric Computing. En *1st Workshop on Security and Privacy at the Conference of Pervasive Computing*, 2004.

[43] J. Daemen and V. Rijndael. *The Design of Rijndael*. Springer, 2002.

[44] T. DeMarco. *Structured Analysis and Systems Specification*. Prentice-Hall, New Jersey, USA, 1979.

[45] Y. Desmedt. Threshold Cryptosystems. En *Proceedings of Advances in Cryptology - Auscrypt'92, LNCS 768*, páginas 3–14, Springer-Verlag, 1993.

[46] A. Detsch, P. Gaspary, M. P. Barcellos, and G. G. H. Cavalheiro. Towards a Flexible Security Framework for Peer-to-Peer Based Grid Computing. En *Proceedings of the 2nd workshop on Middleware for grid computing*, páginas 52 – 56, 2004.

[47] R. Dhamija and A. Perring. Déjà vu: A User Study Using Images for Authentication. En *Proceedings of the 9th USENIX Security Symposium*, 2000.

[48] T. Dierks and C. Allen. The TLS Protocol Version 1.0. Internet Engineering Task Force: RFC 2246, 1999.
http://www.faqs.org/rfcs/rfc2246.html.

[49] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[50] E. Dodono, J. Quaini Sousa, and H. Crestana Guardia. Gridbox: Securing Hosts from Malicious and Greedy Applications. En *Proceedings of the 2nd workshop on Middleware for grid computing*, páginas 17–22, 2004.

[51] Tom Duff. Rc: The Plan 9 Shell. En *Proceedings for the Winter 1994 USENIX Conference*, páginas 223–234, 1994.

[52] C. M. Ellison, B. Franz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. SPKI Certificate Theory, Simple Public Key Certificate, SPK1 Example. *Internet draft, IETF SPKI Working Group*, 1997.

[53] C. English, P. Nixon, and S. Terzis. Dynamic Trust Models for Ubiquitous Computing Environemnts. En *Proceedings of the Ubicomp 2002, Workshop on Security in ubiquitous computing*, 2002.

[54] P. Eronen and P. Nikander. Decentralized JINI Security. En *Proceedings of the Network and Distributed System Security Symposium 2001*, páginas 161–172, 2001.

[55] *Plan 9 Programmer's Manual, Section 4: File Servers.* AT & T Bell Laboratories, Murray Hill, NJ, fourth edition, 2002. http://www.cs.bell-labs.com/magic/man2html/4/factotum.

[56] Fortezza™ Cryptocard. http://www.mykotronx.com/products/fortezza/crypto.asp.

[57] Gaia: Demos. http://choices.cs.uiuc.edu/gaia/demos.html.

[58] Gaia: Active Spaces for Ubiquitous Computing. http://choices.cs.uiuc.edu/gaia/index.html.

[59] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project AURA: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive computing*, 1(2):22–31, 2002.

[60] The GLOBUS Alliance. http://www-unix.globus.org.

[61] The GLOBUS Alliance: GSI Documentation. http://www-unix.globus.org/toolkit/docs/3.2/gsi/index.html.

[62] The GLOBUS Alliance: About the GLOBUS Toolkit. http://www-unix.globus.org/toolkit/about.html.

[63] L. Gong. Java Security: Present and Near Future. *IEEE Micro*, 17(3):14–19, 1997.

[64] L. Gong. JXTA: a Network Programming Environment. *IEEE Internet Computing*, 5(3):88–95, 2001.

[65] S. D. Gribble, M. Welsh, R. Behren, E. A. Brewer, D. E. Culler, N. Borisov, S. E. Czerwinski, R. Gummadi, J. R. Hill, A. D. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Y. Zhao. The NINJA Architecture for Robust Internetscale Systems and Services. *Computer Networks. Special issue on Pervasive Computing*, 35(4):473–497, 2000.

[66] R. Grimm and B. Bershad. Future Directions: System Support for Pervasive Applications. En *Proceedings of International Workshop on Future Directions in Distributed Computing (FuDiCo 2002), LNCS 2584*, páginas 212–217, Springer-Verlag, 2002.

[67] GSMWorld - What is GPRS? http://www.gsmworld.com/technology/gprs/intro.shtml.

[68] V. Gupta, M. Millard, S. Fung, Yu Zhu, N. Gura, H. Eberle, and S. Chang Shantz. Sizzle: A Standards-Based End-to-End Security Architecture for the Embedded Internet. En *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*, páginas 247–256, 2005.

[69] J. Haartsen, M. Naghshineh, J. Inouye, O. Joeressen, and W. Allen. Bluetooth: Vision, Goals, and Architecture. *Mobile Computing and Communications Review*, 2(4):38–45, 1998.

[70] R. N. Haber How We Remember What We See. *Cientific American*, 250(5):104–112, 1970.

[71] Hexamite, Hexamite, Engadine, Australia. *CHX5 The Next Generation Ultrasonic Positioning*, 2005. http://www.hexamite.com/hx5c.pdf.

[72] R. Hill, J. Al-Muhtadi, R. Campbell, A. Kapadia, P. Naldurg, and A. Ranganathan. A Middleware Architecture for Securing Ubiquitous Computing Cyber Infraestructures. *IEEE Distributed Systems Online*, 5(9):1–14, 2004.

[73] L. Hong and A. K. Jain. Integrating Faces and Fingerprints for Personal Identification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1295–1307, 1998.

[74] R. Housley, W. Polk, W. Ford, and D. Solo. Internet x.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Internet Engineering Task Force: RFC 3280, 2002. http://www.ietf.org/rfc/rfc3280.txt.

[75] Hp Cambirge Research Lab: Compaq Personal Server Project. http://www.crl.hpl.hp.com/projects/personalserver/.

[76] Ibutton. http://www.ibutton.com.

[77] The Internet Engineering Task Force. http://www.ietf.org.

[78] Sun Microsystems Inc. *Sun White Paper: Jini Architectural Overview.* Santa Clara, CA, USA, 1999.
http://www.sun.com/jini/whitepapers/architecture.pdf.

[79] Infrared Data Association. http://www.irda.org.

[80] S. Izuka, K. Uwazumi, K. Nakahama, S. Nakajima, and K. Ogawa. Secure PC Environment Roaming Technology for Ubiquitous Office. En *Workshop on Security in Ubiquitous Computing, Ubicomp 2003*, 2003.

[81] Inside the Java Ring Event. http://java.sun.com/features/1998/07/ring-project.html.

[82] B. Johanson, A. Fox, and T. Winograd. The Interactive Project: Experience with Ubiquitous Computing Rooms. *IEEE Pervasive Computing*, 1(2):71–78, 2002.

[83] P. Johansson and M. Kazantzidis. Bluetooth: An Enabler for Personal Area Networking. *IEEE Network*, 15(5):28–37, 2001.

[84] A. Josang. The Right Type of Trust for Distributed Systems. En *Proceedings of the 1996 New Security Paradigms Workshop*, páginas 119–131, 1996.

[85] W. Josephson, E. G. Sirer, and F. B. Schneider. Peer-to-peer Authentication with a Distributed Single Sign-On service. En *Proceedings of the International Workshop on Peer-to-Peer Systems, LNCS 3279*, páginas 250–258, Springer-Verlag, 2004.

[86] P. Kaijser, T. Parker, and D. Pinkas. SESAME: The Solution to Security for Open Distributed Systems. *Computer Communications*, 17(7):501–518, 1994.

[87] P. Kamp and R. N. M. Watson. Jails: Confining the Omnipotent Root. En *Proceedings of the 2nd International System Administration and Networking Conference (SANE2000)*, 2000.

[88] H. Kato, Y. Sato, T. Fukumoto, T. Sasaki, and M. Funabashi. Trust Network-Based Filtering to Retrieve Trustworthy Word-of-Mouth Information. *Workshop on Security in Ubiquitous Computing, Ubicomp 2003*, 2003.

[89] B. W. Kernighan and R. Pike. *The UNIX Programming Environment.* Prentice Hall, 1984.

[90] T. J. Killian. Processes As Files. En *Proceedings of the Summer 1984 USENIX Conference*, páginas 203–207, 1984.

[91] N. Klobitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[92] H. Kopp, U. Lucke, and D. Tavangarian. Security Architecture For Service-Based Mobile Environments. En *Proceedings of the 2nd International Workshop on Middleware Support for Pervasive Computing, IEEE PerCom 2005*, páginas 199–203, 2005.

[93] D. P. Kormann and A. D. Rubin. Risks of the Passport Single Sing-On Protocol. *Computing Networks, Elsevier Science Press*, 33:51–58, 2000.

[94] R. Kravets, K. Schwan, and K. Calvert. Power-aware Communication for Mobile Computers, 1999.

[95] X. Lai and J. L. Massey. A Proposal for a New Block Encryption Standard. *Advances in Cryptology - EUROCRYPT, LNCS 473*, páginas 389–404, Springer-Verlag, 1990.

[96] B. Landon, M. Miller, and C. de Herrera. *Windows Mobile Solutions*. Visual, 2004.

[97] M. Langheinrich. When Trust Does Not Compute - the Role of Trust in Ubiquitous Computing. *Workshop on Privacy at the 5th International Conference on Ubiquitous Computing, Ubicomp 2003*, 2003.

[98] K. Leal, F. J. Ballesteros, G. Guardiola, and E. Soriano. Ubiterm: A Hand-Held Control-Center for User's Activity Mobility. En *Proceedings of the IEEE International Conference on Pervasive Services 2005*, páginas 127–136, 2005.

[99] H. Lee and K. Kim. An Adaptive Authentication Protocol Based on Reputation for Peer-to-Peer Systems. En *Proceedings of the Symposium on Cryptography and Information Security (SCIS) 2003*, páginas 661–666, 2003.

[100] J. Lopez, R. Oppliger, and G. Pernul. Authentication and Authorization Infrastructures (AAIS): A Comparative Survey. *Computing And Security, Elsevier Science Press*, 23:578–590, 2004.

[101] Ls: Papers and Documentation. http://lsub.org/#docs.

[102] Ls: Demos http://lsub.org/ls/demos.

[103] Laboratorio de Sistemas. http://lsub.org.

[104] A. S. Tanenbaum M. Steen, P. Homburg. Globe: A Wide-Area Distributed System. *IEEE Concurrency*, 1999.

[105] D. Mazieres, M. Kaminsky, M. Frans Kaashoek, and E. Witchel. Separating Key Management from File System Security. En *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, páginas 124–139, 1999.

[106] A. Menezes. *Handbook of Applied Cryptography.* CRC, 1996.

[107] D. Mery. *Symbian OSversion 7.0 functional description. White paper.* Symbian Ltd., 2003. http://www.symbian.com/technology/whitepapers.html.

[108] V. S. Miller. Use of Elliptic Curves in Cryptography. En *Proceedings of Advances in Cryptography, CRYPTO'85, LNCS 228*, páginas 417–426, Springer-Verlag, 1986.

[109] Mimio by Virtual Ink. http://www.mimimo.com/meet/mimioboard/.

[110] K. Mitnick and W. L. Simon. *The Art of Deception.* Wiley, 2002.

[111] G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, 1965.

[112] Mozilla Suite - The All-in-One Internet Application Suite. http://www.mozilla.org.

[113] Microsoft .Net Passport: Balanced Authentication Solutions. 2002.

[114] Microsoft .Net Passport Review Guide. 2003.

[115] Microsoft Windows Family Home Page.
http://www.microsoft.com/windows.

[116] Mypasswordsafe Manual.
http://www.semanticgap.com/myps/.

[117] J. Gutknecht N. Wirth. The Oberon System. *Software Practice and Experience*, 19(9):857–893, 1989.

[118] National Institute of Standards and Technology. *FIPS PUB 46-2: Data Encryption Standard*. National Institute for Standards and Technology, Gaithersburg, MD, USA, 1993.

[119] National Institute of Standards and Technology. *FIPS PUB 186: Digital Signature Standard (DSS)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, 1994.

[120] National Institute of Standards and Technology. *FIPS PUB 180-1: Secure Hash Standard*. National Institute for Standards and Technology, Gaithersburg, MD, USA, 1995.

[121] National Institute of Standards and Technology. *FIPS PUB 197: Advanced Encryption Standard*. National Institute for Standards and Technology, Gaithersburg, MD, USA, 2001.

[122] *Plan 9 Programmer's Manual, Section 6: NDB*. AT & T Bell Laboratories, Murray Hill, NJ, fourth edition, 2002. http://www.cs.bell-labs.com/magic/man2html/6/ndb.

[123] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM* 21(12):993–999, 1978.

[124] Netscape Browser. http://browser.netscape.com.

[125] *The Network Information Service*. Sun Microsystems, 1990. in System & Network Administration, SunOS 4.1 manual set.

[126] National Security Agency and Central Security Service. www.nsa.gov.

[127] B. O'Hara and A. Petrick. *IEEE 802.11 Handbook: A Designer's Companion*. IEEE Standards Press, second edition, 2005.

[128] Okiok. *Focal Point Evaluator's Guide*, 2004. http://www.okiok.com/index.jsp?page=Focal+Point.

[129] On-hand PC Homepage. http://www.matsucomusa.com.

[130] R. Oppliger. Microsoft .Net: A Security Analysis. *IEEE Computer*, 36:29–35, 2003.

[131] D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: the Case of AES. *Cryptology ePrint Archive, Report 2005/271*. 2005.

[132] Palmsource: PalmOS. http://www.palmsource.com/palmos/.

[133] D. G. Park, J. K. Kim, S. J Bong, J. H. Hwang, C. H. Hyung, and S. W. Kang. Context Aware Service Using Intra-Body Communication. En *Proceedings of the 4th IEEE International Conference on Pervasive Computing and Communications*, páginas 84–91, 2006.

[134] R. Perlman and C. Kaufman. Secure Password-Based Protocol for Downloading a Private Key. En *Proceedings of the The Internet Society Network and Distributed System Security Symposium 1999*, 1999.

[135] A. Perring and D. Song. Hash Visualization: A New Technique to Improve Real-World Security. En *Proceedings of International Workshop on Cryptographic Techniques and E-Commerce CrypTEC 99*, 1999.

[136] A. Pineda. *Aplicación de seguridad basada en visión.* PFC. Universidad Rey Juan Carlos. 2006. http://gsyc.escet.urjc.es/∼apineda/pfc.

[137] R. Pike, D. Presotto, K. Thompson, and H. Trickey. Plan 9 from Bell Labs. En *Proceedings of the Summer 1990 UKUUG Conference*, páginas 1–9, 1990.

[138] R. Pike. Acme: A User Interface for Programmers. En *Proceedings for the Winter 1994 USENIX Conference*, páginas 223–234, 1994.

[139] R. Pike. Plumbing and Other Utilities. En *Proceedings of the USENIX Annual Technical Conference 2000*, páginas 159–170, 2000.

[140] Plan 9 From Bell Labs Fourth Edition. http://www.cs.bell-labs.com/plan9dist/.

[141] J. Postel and J. Reynolds. *Telnet Option Specification.* Internet Engineering Task Force: RFC 855, 1983. http://www.ietf.org/rfc/rfc855.txt.

[142] J. Postel and J. Reynolds.*Telnet Protocol Specification.* Internet Engineering Task Force: RFC 854, 1983. http://www.ietf.org/rfc/rfc854.txt.

[143] R. S. Pressman. *Software Engineering: A Practitioner's Approach.* Mc Graw Hill, 1997.

[144] Protocom, Protocom Development Systems. *SecureLogin Single Sign-On White Paper*, 2003. http://www.protocom.com/html/whitepapers/.

[145] Pwsafe Password Database. http://nsd.dyndns.org/pwsafe/.

[146] Real academia de la lengua española: el diccionario en cifras. http://buscon.rae.es/diccionario/Drae_v2/cifras.htm.

[147] Random Art. http://www.random-art.org.

[148] W. Rankl and W. Effing. *Smart Card Handbook*. John Wiley and Sons, second edition, 2000.

[149] E. Rescorla. HTTP over TLS. Internet Engineering Task Force: RFC 2818, 2000. http://www.ietf.org/rfc/rfc2818.txt.

[150] Epcglobal Inc. http://www.epcglobalinc.org.

[151] N. Rhodes and J. McKeehan. *Palm OS Programming: The Developer's Guide*. O'Reilly, second edition, 2001.

[152] R. Rivest. The MD5 Message-Digest Algorithm. Internet Engineering Task Force: RFC 1321, 1992. http://www.ietf.org/rfc/rfc1321.txt.

[153] R. Rivest. A Description of the RC2 Encryption Algorithm. Internet Engineering Task Force: RFC 2268, 1998. http://www.ietf.org/rfc/rfc2268.txt.

[154] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[155] R. L. Rivest and B. Lampson. SDSI: A Simple Distributed Security Infrastructure. En *CRYPTO'96 Rumpsession*, 1996.

[156] M. Roman, C. K. Hess, R. Cerqueira, A. Ranganat, R. H. Campbell, and K. Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing* 1(4):74–82, 2002.

[157] RSA Security. http://www.rsasecurity.com.

[158] A. D. Rubin. *White-Hat Security Arsenal*. Addison wesley, 2001.

[159] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-End Arguments in System Design. *ACM transactions on Computer Systems*, 2(4):277–288, 1984.

[160] G. Sampemane, P. Naldurg, and R. Campbell. Access Control for Active Spaces. En *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC2002)*, páginas 343–352, 2002.

[161] R. S. Sandhu and P. Samarati. Access Control: Principles and Practice. *IEEE Communications Magazine*, 32(9):40–48, 1996.

[162] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[163] B. Schneier. *Schneier On Security: New Cryptanalytic Results Against SHA-1.* http://www.schneier.com/blog/archives/2005/08/new_cryptanalyt.html.

[164] B. Schneier. *Applied Cryptography.* John Wiley and Sons, 1996.

[165] B. Schneier. *Beyond Fear: Thinking Sensibly about Security in an Uncertain World.* Copernicus Books, New York, NY, 2003.

[166] B. Schneier. Password Safe. http://www.counterpane.com/passsafe.html.

[167] D. Schoder and K. Fischbach. Peer-to-Peer Prospects. *Communications of the ACM*, 46(2):27–29, 2003.

[168] SD Card Association. http://www.sdcard.org.

[169] *Plan 9 Programmer's Manual, Section 1: Secstore.* AT & T Bell Laboratories, Murray Hill, NJ, fourth edition, 2002. http://www.cs.bell-labs.com/magic/man2html/1/secstore.

[170] SECUDE Single Sign-On. http://www.secude.de/products/sso/.

[171] SESAME: A Secure European System for Applications in a Multi-Vendor Environment. https://www.cosic.esat.kuleuven.ac.be/sesame/.

[172] N. Shankar and W. Arbaugh. On Trust for Ubiquitous Computing. En *Proceedings of Ubicomp 2002, Workshop on Security in Ubiquitous Computing*, 2002.

[173] N. Shankar and D. Balfanz. Enabling Secure Ad-Hoc Communication Using Context Aware Security Services. En *Proceedings of Ubicomp 2002, Workshop on Security in Ubiquitous Computing*, 2002.

[174] Smartcard Alliance. http://www.smartcardalliance.org.

[175] E. Soriano. Shad: A Human Centered Security Architecture for Partitionable, Dynamic and Heterogeneous Distributed Systems. En *Proceedings of the 5th ACM/IFIP/USENIX International Middleware Workshops (1st International Middleware Doctoral Symposium)*, páginas 294–298, 2004.

[176] E. Soriano, F. J. Ballesteros, and G. Guardiola. Peer-to-Peer Single Sign-On Security Scheme for Today's Smart Spaces. En *Actas de las XIV Jornadas de Concurrencia y Sistemas Distribuidos*, páginas 309–324, 2006.

[177] F. Stajano. The Resurrection of Duckling - What next? En *Proceedings of Security Protocols 2000, LNCS 2133*, páginas 204–214, Springer-Verlag, 2000.

[178] F. Stajano. Security for whom? The Shifting Security Assumptions of Pervasive Computing. En *Proceedings of the International Security Symposium 2002, LNCS 2609*, páginas 16–27, Springer-Verlag, 2002.

[179] F. Stajano. *Security for Ubiquitous Computing.* John Wiley and Sons, 2002.

[180] F. Stajano and J. Crowcroft. The Butt of the Iceberg: Hidden Security Problems of Ubiquitous Systems. En *Ambient Intelligence: Impact on Embedded System Design.* Basten et al., 2003.

[181] W. Stalligns. *Cryptography and Network Security.* Prentice Hall, 2002.

[182] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An Authentication Service for Open Network Systems. En *Proceedings of the Winter 1988 USENIX Conference*, páginas 191–202, 1988.

[183] W. R. Stevens. *Advanced Programming in the UNIX Environment.* Addison-Wesley, 1992.

[184] W. R. Stevens. *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols.* Addison-Wesley, 1996.

[185] Symbian: The Mobile Operating System. http://www.symbian.com/.

[186] A. Tanenbaum. *Distributed Operating Systems.* Prentice-Hall, 1995.

[187] J. Tisal. *The GSM Network: GPRS Evolution: One Step Towards UMTS.* John Willey and Sons, second edition, 2001.

[188] W. M. Ulrich. *Legacy Systems: Transformation Strategies.* Prentice Hall PTR, 2002.

[189] I. Vajda and L. Buttyan. Lightweight Authentication Protocols. En *Workshop on Security in Ubiquitous Computing, Ubicomp 2003*, 2003.

[190] P. Viswanathan, B. Gill, and R. Campbell. Security Architecture in GAIA. *Technical Report UIUCDCS-R-2001-2215 UILU-ENG-2001-1720, Universiy of Illinois at Urbana-Champaign*, 2001.

[191] VMWare, Inc. http://www.vmware.com.

[192] X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. En *Proceedings of the Crypto 2005 25th Annual International Cryptology Conference. LNCS 3621*, páginas 17–36, Springer-Verlag, 2005.

[193] M. Weiser. "The Computer for the 21st Century". *Scientific American*, 265(3):94–104, 1991.

[194] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for Grid Services. En *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, páginas 48–57, 2003.

[195] Wi-Fi Alliance. http://www.wi-fi.org.

[196] X-10 Home Page. http://www.x10.com.

[197] K. W. Yeh and L. Wang. *IrDA Implementation, Instant Solution and Certification*. ACTiSYS Corp. http://www.actisys.com/IrDAOverview-Word.pdf.

[198] W. Yeong, S. Kille, and T. Howes. Lightweight directory access protocol. Internet Engineering Task Force: RFC 1777, 1995. http://www.ietf.org/rfc/rfc1777.txt.

[199] T. Ylonen. SSH - Secure Login Connections Over the Internet. En *Proceedings of the 6th USENIX Security Symposium*, páginas 37–42, 1996.

[200] F. Zhu, M. W. Mutka, and L. M. Ni. The Master Key: A Private Authentication Approach for Pervasive Computing Environments. En *Proceedings of the 4th IEEE International Conference on Pervasive Services*, páginas 212–221, 2006.

[201] P. Zimmermann. *Pretty Good Privacy (PGP), PGP User's Guide*. Massachusetts Institute of Technology, 1994.