UNIVERSIDAD DE MURCIA

FACULTAD DE INFORMÁTICA

Providing Federated Access and SSO to Internet Services by means of Kerberos and AAA Infrastructures

Proporcionando Acceso Federado y SSO a Servicios de Internet mediante Kerberos e Infraestructuras AAA

D. Alejandro Pérez Méndez

2014

Universidad de Murcia

Facultad de Informática

PROPORCIONANDO ACCESO FEDERADO Y SSO A SERVICIOS DE INTERNET MEDIANTE KERBEROS E INFRAESTRUCTURAS AAA

Tesis Doctoral

Presentada por:
Alejandro Pérez Méndez

Supervisada por:
Dr. Gabriel López Millán
Dr. Rafael Marín López

Murcia, Septiembre de 2014

D. Juan Antonio Sánchez Laguna, Profesor Titular de Universidad del Área de Ciencia de la Computación e Inteligencia Artificial y presidente de la Comisión Académica del Programa de doctorado en Informática,

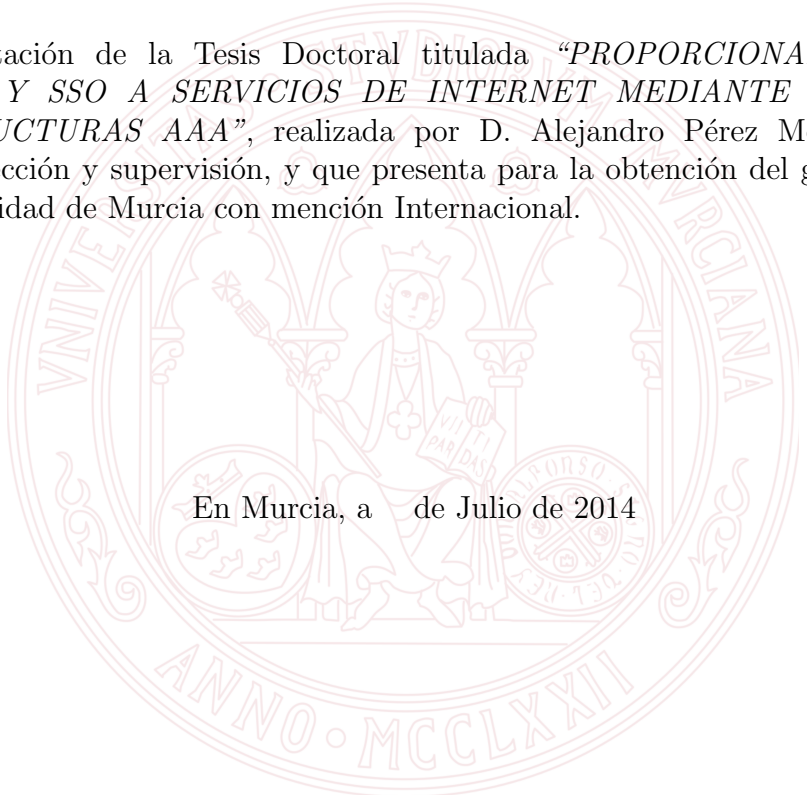
INFORMA:

Que vista la solicitud de autorización de presentación de tesis doctoral de D. Alejandro Pérez Méndez, titulada *"PROPORCIONANDO ACCESO FEDERADO Y SSO A SERVICIOS DE INTERNET MEDIANTE KERBEROS E INFRAESTRUCTURAS AAA"*, realizada bajo la inmediata dirección y supervisión de D. Gabriel López Millán y D. Rafael Marín López, y evaluado el expediente completo, la Comisión Académica del Programa de Doctorado, de conformidad con lo establecido en el artículo 21 del "Reglamento por el que se regulan las enseñanzas oficiales de doctorado de la Universidad de Murcia", resolvió la autorización de presentación de la tesis doctoral.

En Murcia, a de Septiembre de 2014

D. Gabriel López Millán, Profesor Titular del Área de Ingeniería Telemática en el Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia, AUTORIZA:

La presentación de la Tesis Doctoral titulada *"PROPORCIONANDO ACCESO FEDERADO Y SSO A SERVICIOS DE INTERNET MEDIANTE KERBEROS E INFRAESTRUCTURAS AAA"*, realizada por D. Alejandro Pérez Méndez, bajo mi inmediata dirección y supervisión, y que presenta para la obtención del grado de Doctor por la Universidad de Murcia con mención Internacional.

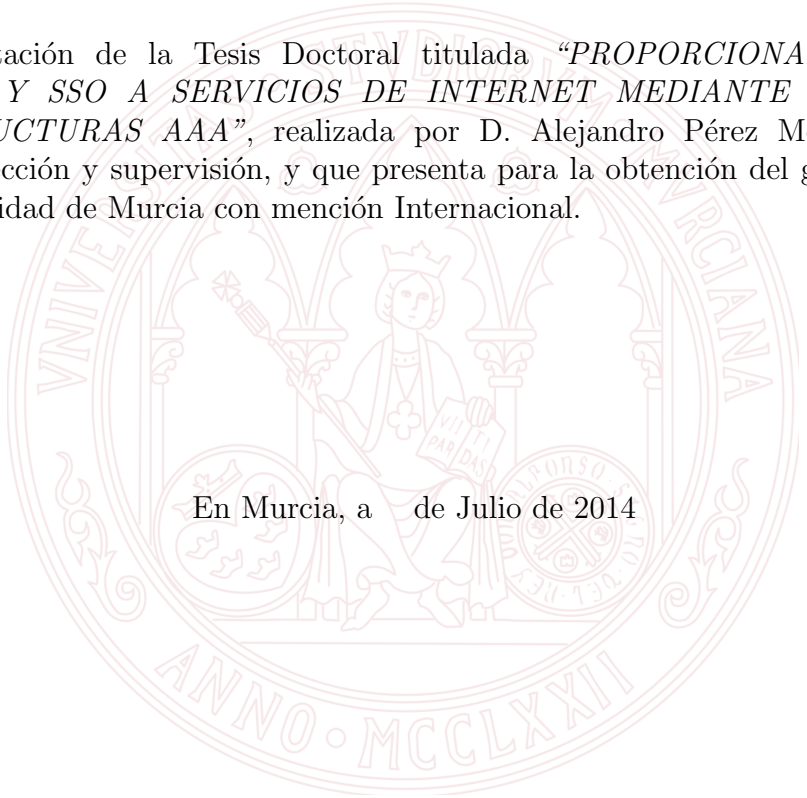


En Murcia, a de Julio de 2014

D. Gabriel López Millán

D. Rafael Marín López, Profesor Contratado Doctor del Área de Ingeniería Telemática en el Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia, AUTORIZA:

La presentación de la Tesis Doctoral titulada *"PROPORCIONANDO ACCESO FEDERADO Y SSO A SERVICIOS DE INTERNET MEDIANTE KERBEROS E INFRAESTRUCTURAS AAA"*, realizada por D. Alejandro Pérez Méndez, bajo mi inmediata dirección y supervisión, y que presenta para la obtención del grado de Doctor por la Universidad de Murcia con mención Internacional.



En Murcia, a de Julio de 2014

D. Rafael Marín López

Resumen

Desde los orígenes de Internet, el control de acceso a los servicios de aplicación (compartición de ficheros, correo electrónico, etc.) ha sido una de las principales preocupaciones para los proveedores de servicio. Generalmente, éstos quieren permitir el acceso sólo a aquellos usuarios que se hayan registrado previamente (p.ej. banca online), o a aquellos que cumplan un conjunto de condiciones específico (p.ej. ser parte del personal de una universidad para acceder a un repositorio de publicaciones de investigación). También quieren mantener un registro de quién accedió y cuándo (p.ej. por motivos de facturación).

El esquema de control de acceso más desplegado desde el comienzo de Internet hasta ahora se da cuando el propio servicio implementa toda la funcionalidad requerida. Sin embargo, este escenario presenta varios inconvenientes: en primer lugar, los usuarios tienen unas credenciales y atributos diferentes para cada servicio de aplicación; en segundo lugar, los servicios tienen que implementar toda la lógica de control de acceso por sí mismos. Para suavizar estos problemas, los proveedores de servicio suelen centralizar el almacenamiento de credenciales e información de identidad en una base de datos compartida. Un ejemplo típico se da en un campus universitario, donde las mismas credenciales y atributos se comparten entre los diferentes servicios.

El siguiente paso en este camino hacia la simplificación consiste en la delegación de (parte de) la funcionalidad de control de acceso en una entidad centralizada, generalmente llamada *Proveedor de Identidad* (en inglés *Identity Provider* - IdP). De esta forma, los usuarios son en realidad autenticados por el IdP, mientras que los servicios de aplicación reciben el resultado de ese proceso, junto con información de identidad (p.ej. atributos de usuario) para tomar una decisión de autorización. Otra ventaja de desplegar un IdP es la posibilidad de proporcionar *Single Sign-On* (SSO), es decir, entregar al usuario algún tipo de información temporal (p.ej. token, cookie, etc.) que se pueda usar para simplificar los siguientes procesos de autenticación para el acceso a otros servicios del proveedor. Algunos ejemplos de tecnologías que permiten el uso de un IdP son SAML, Kerberos, OpenID u OAuth. *Google Accounts* constituye un claro ejemplo de despliegue de un IdP.

Además, el auge de las telecomunicaciones, y las nuevas necesidades que han surgido con

él, ha propiciado el establecimiento de acuerdos de negocios entre diferentes proveedores de servicio: las llamadas *federaciones de identidad*. Estas federaciones definen y regulan cómo se relacionan los servicios de aplicación y los IdPs de diferentes organizaciones para permitir a los usuarios de un proveedor de servicios acceder a cualquiera de los servicios proporcionados por otro proveedor perteneciente a la federación. Cada una de estas federaciones define qué protocolos y tecnologías se usan para cumplir con este propósito, así como los tipos de servicios de aplicación que se soportan en la misma. En concreto, hay dos tipos de federaciones de identidad completamente asentadas: *federaciones de identidad basadas en Web* y *federaciones de identidad basadas en infraestructuras AAA* (*Authentication, Authorization, and Accounting*). Las primeras se centran en proporcionar un acceso federado a aplicaciones Web, usando tecnologías como OAuth, OpenID o SAML. Las segundas se utilizan hoy en día principalmente para controlar el servicio de acceso a la red, usando tecnologías como RADIUS o Diameter.

Esto ha dado lugar a escenarios donde las organizaciones pueden tener desplegadas dos federaciones diferentes: una basada en AAA y otra basada en Web. Pero esto genera una pregunta: *¿qué ocurre con otros tipos de servicio?* En los últimos años ha obtenido un interés y una notoriedad significativa un tipo de escenario que intenta dar una solución de federación a servicios que no disponen de un soporte adecuado para ninguno de los tipos de federaciones de identidad descritos anteriormente, como la transferencia de ficheros, el acceso al terminal remoto o la gestión del Cloud. Este escenario se ha denominado *federación de identidad más allá del Web*.

Desafortunadamente se han definido pocas soluciones para este tipo de federación. Concretamente, hasta donde el autor ha podido conocer, sólo dos de ellas pueden considerarse como tales: *Kerberos* y *Moonshot*. Esta tesis analiza estas dos soluciones y las evalúa en función de una serie de requisitos que hemos definido como deseables para este tipo de federación. Sin embargo, ninguna de estas soluciones cumple todos estos requisitos. Por un lado, el soporte de federación de Kerberos (llamado *cross-realm*) presenta diversos problemas documentados, como la escalabidad del modelo de confianza. Además, requiere establecer una infraestructura de federación adicional a las ya existentes (es decir, basadas en Web y en AAA). Más aún, Kerberos carece de una gestión de autorización que pueda considerarse completa. Por otro lado, Moonshot no proporciona soporte de SSO completo y requiere la adaptación de todos los servicios de aplicación para su funcionamiento.

Esta tesis pretende diseñar soluciones para crear *federaciones de identidad más allá del Web*, evitando los problemas que presentan las anteriores propuestas. En concreto, este documento se centra en el análisis, diseño e implementación de soluciones para la interconexión de Kerberos con federaciones AAA. Kerberos posee grandes cualidades

para el control de acceso dentro de una misma organización (p.ej. es seguro, ligero computacionalmente, soporta SSO, etc.). Además, Kerberos está ampliamente desplegado y está soportado por la mayoría de servicios de aplicación actuales. Sin embargo, como se ha mencionado anteriormente, presenta algunos puntos débiles en cuanto a soporte de federación o de autorización. Por otro lado, las federaciones AAA se han usado durante años para proporcionar acceso a la red, con gran éxito y presencia. La integración de estas tecnologías ofrece una solución que supera al modo de operación *cross-realm* de Kerberos, usando una alternativa de federación con una aceptación mucho mayor. De esta forma, los beneficios de ambas tecnologías se combinan para proporcionar una mejor experiencia de usuario, un despliegue simplificado (no hace falta modificar los servicios de aplicación) y una mejor explotación de recursos, tanto a nivel de cómputo como de uso de la red.

Además de la autenticación federada, esta tesis se centra en el análisis y diseño de mecanismos para incorporar una solución de autorización avanzada en Kerberos. Mientras que Kerberos no define un proceso de autorización preciso, SAML se ha usado con gran éxito y amplio despliegue en federaciones basadas en Web. En particular, entre las ventajas de SAML cabe destacar su gran aceptación por la comunidad, su gran flexibilidad para representar datos de autorización o la disponibilidad de herramientas software para su generación y procesado.

Para lograr la integración de estas tecnologías, el objetivo general de esta tesis puede expresarse como:

> *Analizar, diseñar y validar soluciones que permitan a los usuarios obtener credenciales Kerberos para un proveedor de servicios específico, como resultado de un proceso de autenticación realizado a través de una federación AAA y de un proceso de autorización basado en el intercambio de sentencias SAML con la organización origen.*

Este objetivo se ha abordado de tres formas diferentes, dependiendo del escenario de uso. En primer lugar, se puede extender el mecanismo de autenticación estándar de Kerberos para incorporar soporte para infraestructuras AAA. Una segunda aproximación consiste en realizar la autenticación federada usando un protocolo externo a Kerberos, con soporte nativo para infraestructuras AAA, y generar credenciales Kerberos a partir del mismo. Finalmente, una tercera estrategia consiste en aprovechar el proceso de autenticación federada que se realiza para el acceso a la red (p.ej. eduroam). Como este proceso ha involucrado a la federación AAA, podría usarse para generar credenciales Kerberos en la organización a la que el usuario se ha conectado a la red. Este documento

analiza, diseña y propone una solución para cada una de estas alternativas, a las que se ha denominado *FedKERB*, *PanaKERB* y *EduKERB* respectivamente.

FedKERB proporciona una arquitectura que integra las infraestructuras Kerberos y AAA mediante la definición un nuevo mecanismo de pre-autenticación para Kerberos basado en tecnologías como GSS-API y EAP. De esta forma, Kerberos hace uso de la infraestructura AAA para realizar una autenticación federada, generando nuevas credenciales para el usuario en base al material criptográfico derivado de la autenticación EAP. Además, la arquitectura incluye la posibilidad de realizar una autorización avanzada tras el proceso de autenticación, mediante el transporte de una sentencia SAML con información de identidad del usuario, desde el IdP de la organización origen hasta la infraestructura Kerberos del proveedor de servicios. FedKERB proporciona una solución genérica que satisface todos los requisitos deseados para una *federación de identidad más allá del Web*, incluyendo soporte directo de SSO dentro de los límites del proveedor de servicios (es decir, intra-dominio). Todo esto hace que FedKERB sea adaptable a la mayoría de los escenarios *más allá del Web* previstos. Sin embargo, aunque FedKERB no requiere modificar los servicios de aplicación, sí requiere la modificación del proceso de pre-autenticación Kerberos. Aunque esto sólo implica actualizar una única entidad (el KDC), este requisito podría no ser aceptable para algunos proveedores de servicio o en ciertos escenarios, donde la infraestructura Kerberos desplegada deba permanecer intacta.

PanaKERB define una alternativa a FedKERB que no requiere la modificación del mecanismo de pre-autenticación de Kerberos. Para conseguir este objetivo, esta contribución define una arquitectura que usa un protocolo externo a Kerberos (*out-of-band*) con soporte nativo para infraestructuras AAA (en este caso, PANA) para realizar la autenticación federada del usuario. Este proceso *out-of-band* resulta en la obtención de nuevas credenciales Kerberos, generadas del material criptográfico derivado de la autenticación EAP. La fortaleza de PanaKERB reside en su simplicidad, ya que no requiere la modificación de ningún componente ya desplegado del proveedor. Sin embargo, tiene la desventaja de introducir un protocolo adicional (PANA) y una nueva entidad (*PANA Authentication Agent* - PAA) en la arquitectura. No obstante, esperamos que esta simplicidad haga que su adopción y despliegue sea más fácil que en el caso de FedKERB, resultando una solución más adecuada para aquellas organizaciones que ya tienen una infraestructura Kerberos desplegada y que son reticentes a modificarla para incorporar los cambios requeridos por FedKERB. Además, PanaKERB también permite el uso del modelo de autorización avanzada definido para FedKERB.

Finalmente, EduKERB proporciona una arquitectura *cross-layer* que integra Kerberos con la autenticación basada en AAA realizada durante el acceso a la red. Esto se consigue

definiendo un nuevo mecanismo de pre-autenticación para Kerberos que utiliza el resultado de una ejecución EAP exitosa. Dado que estos dos procesos de autenticación (acceso a la red y Kerberos) ocurren en diferentes capas del modelo OSI (capa de enlace y capa de aplicación), recibe el nombre de SSO cross-layer. EduKERB reduce el número de autenticaciones federadas basadas en AAA que son necesarias para acceder a los servicios de aplicación, mejorando la eficiencia y optimizando la utilización de recursos. Otro de los aspectos relevantes de EduKERB, cuando se compara con FedKERB y PanaKERB, es su ámbito de aplicación. Mientras que éstas últimas tienen una aplicabilidad genérica, EduKERB se ha diseñado para aprovechar el proceso de acceso a la red específicamente diseñado para la infraestructura RADIUS de eduroam. Sin embargo, puede implantarse en otros escenarios basados en RADIUS si se desea. EduKERB requiere un mayor número de modificaciones en las infraestructuras existentes que las anteriores propuestas. Sin embargo, estas modificaciones permiten reducir el número de autenticaciones EAP y la sobrecarga asociada en términos de mensajes de red y tiempo de ejecución. EduKERB también permite el uso del modelo de autorización avanzada definido para FedKERB y PanaKERB.

Además de la definición de estas contribuciones, esta tesis también proporciona un análisis funcional y de rendimiento de las mismas. El análisis consiste en la definición de un modelo de rendimiento (describiendo las diferentes operaciones con un impacto en el tiempo de ejecución de cada propuesta), la implementación de tres prototipos para demostrar su viabilidad y la medición de su rendimiento real, basado en la ejecución de estos prototipos. De este análisis se han extraído dos conclusiones principales. En primer lugar, se demuestra que las tres propuestas descritas en esta tesis presentan tiempos de ejecución similares a los que se dan durante el acceso a la red (que sirve como referencia dado que es el uso más extendido de las infraestructuras AAA). Además, dado que han sido implementadas y ejecutadas en un entorno real, su viabilidad queda demostrada más allá del punto de vista teórico. Finalmente, se confirma que el rendimiento de las tres propuestas es muy similar, por lo que la decisión de usar una u otra dependerá de los requisitos de cada escenario de despliegue.

Para finalizar, esta tesis discute las líneas de investigación para trabajo futuro más interesantes que han surgido durante su realización, y que pueden proporcionar avances en el área de la misma. Estas líneas se pueden catalogar en función de su plazo estimado de realización. En concreto, el trabajo futuro a corto plazo incluye trabajar activamente en el *despliegue de tecnologías de federación de identidad más allá del Web en entornos reales*; y el análisis, diseño e implementación de una solución que permita el *uso de HTTP como un protocolo out-of-band para Kerberos*, en lugar de PANA. Para el medio plazo, las líneas

de investigación propuestas son el análisis y diseño de soluciones para extender los límites de aplicabilidad del SSO, desde el modelo actual intra-organizacional hacia un modelo de *SSO inter-organizacional*; y el análisis y diseño de *mecanismos para el establecimiento y la gestión dinámica de federaciones AAA*, donde las relaciones de confianza se establezcan de una forma automatizada entre los miembros de la federación. A largo plazo, esta tesis propone el análisis, diseño e implementación de estrategias para integrar el concepto de *IDaaS (Identity as a Service)* con las tecnologías de federación de identidad más allá del Web discutidas en la misma, como una manera de simplificar su despliegue y extender su uso.

Agradecimientos

Me gustaría dedicar esta página para agradecer a toda la gente que, de una u otra forma, ha ayudado y contribuido a que esta tesis sea una realidad.

En primer lugar, a mi mujer Isabel. Gracias por todo tu cariño, apoyo, comprensión y paciencia. Gracias por hacerme feliz todos los días. Sin ti, nada de esto tendría sentido.

A mis padres, que desde niño me han proporcionado todo lo necesario para recibir una buena educación y formación.

A Gabi y Rafa, directores de esta tesis, pero también compañeros y amigos, que con sus sabios consejos e incansable trabajo han sabido guiar a buen término este proyecto que comenzó hace cuatro años. Gracias por invertir tantas horas en mí, investigando, proponiendo ideas, revisando artículos y corrigiendo capítulos. Pero también gracias por las conversaciones de pasillo, las bromas y risas, por las charlas de fútbol... Si una tesis es el resultado de un trabajo de equipo, yo he tenido la ventaja de jugar con los mejores. Y no me olvido de Fernando, miembro de este grupo durante muchos años y amigo y compañero incansable en artículos, reuniones e implementaciones. Gracias por tu compañerismo y tu ayuda desinteresada.

También quiero dar las gracias a Antonio, que me introdujo en el mundo de la investigación, abriéndome las puertas a uno de los grupos de investigación más importantes de la Universidad de Murcia. Gracias por haber hecho lo posible (y a veces lo imposible) por proporcionarme a mí y a otros tantos un puesto laboral donde poder crecer como investigadores, pero también como personas.

A todos los que han sido durante todos estos años mis compañeros de trabajo en la Facultad de Informática. Los que llevan conmigo desde que empezamos la carrera y con los que he compartido tantas cosas, Pedro J., Juan Antonio, y Pedro M. Los que han trabajado conmigo en los diversos proyectos en los que he estado, Manolo Bernal, Fernando Bernal y Elena. A todos los profesores que se han convertido en compañeros, en especial a Gregorio y Diego. A mis compañeros de *Dibulibu* y de la *T3*, por crear un entorno de trabajo inmejorable, donde se trabaja y se disfruta a partes iguales. Y a Manolo Sánchez, que ya no está con nosotros, pero del que aprendí mucho, tanto a nivel personal como profesional.

Finalmente, a Rhys Smith, por haberme acogido durante mi estancia en la Universidad de Cardiff, y haberme guiado y apoyado en el trabajo que realicé allí.

University of Murcia

Faculty of Computer Science

PROVIDING FEDERATED ACCESS AND SSO TO INTERNET SERVICES BY MEANS OF KERBEROS AND AAA INFRASTRUCTURES

PhD Thesis

Author:
Alejandro Pérez Méndez

Thesis Advisors:
Dr. Gabriel López Millán
Dr. Rafael Marín López

Murcia, September 2014

Abstract

From the beginning of the Internet, access control to application services (file sharing, electronic mail, etc.) has been one of the main concerns for service providers. Typically, they might want to make an application service available only to users that where enrolled in a previous process (e.g. online banking), or to those who fulfil a specific set of conditions (e.g access to research publications for university staff). They also might want to keep a record of who accessed to them, and when (e.g. for billing purposes).

The most deployed scheme for access control during the early days of the Internet, and still nowadays, takes place when each individual application service implements all the required functionality. However, this has a number of drawbacks: end users have a different set of credentials and attributes for each application service, and application services need to implement the whole access control logic. To alleviate these problems, service providers typically centralise the storage of credentials and authorization information into a shared database. A typical example would be a campus where the same credentials and attributes are shared amongst the different application services.

A step forward in this simplification path is the outsourcing of (part of) the access control functionality, from the application services to a centralised entity, typically called *Identity Provider* (IdP). In this way, end users are actually authenticated by the same IdP, while application services receive the outcome of that process, along with identity information (e.g. end users attributes) to make an authorization decision. Another advantage of deploying an IdP is the possibility of providing *Single Sign-On* (SSO). That is, providing some sort of temporary information (e.g. token) that can be used to simplify subsequent authentication processes for accessing to other provider's services. Examples of technologies that allow the use of an IdP include SAML, Kerberos, OpenID, or OAuth. *Google Accounts* constitutes a clear example of IdP.

Furthermore, the growth of telecommunications, and the new needs that have arose with it, have promoted the establishment of business agreements between different service providers, the so-called *identity federations*. They define and regulate how application services and IdPs interact to allow end users from one service provider to access any of

the application services provided by another service provider within the federation. Each federation defines which protocols and technologies are used to fulfil these requirements, and which kind of application services are supported. In particular, there are two types of well-established identity federations: *Web-based identity federation* and *Authentication Authorization and Accounting (AAA)-based identity federation*. The former focuses on providing federated access to web-based application services, using technologies such as OAuth, OpenID, or SAML; whereas the latter is nowadays mostly used to control the network access service, using technologies such as RADIUS or Diameter.

This has resulted into scenarios where organizations may have deployed two different federations: one based on AAA and the other to provide access to web-based services. But this raises a question: *What about other types of services?* In recent years, a kind of scenario that aims to provide a federation solution for application services that lack of proper support for any of the types of federations described above, such as file transfer, terminal access, or Cloud, has attracted substantial notoriety and interest. This scenario has been called *Identity federation beyond the web*.

Unfortunately, only a few solutions for this kind of federation have been defined. In particular, to the best of author's knowledge, only two of them can be considered: *Kerberos* and *Moonshot*. This thesis analyses these two solutions, and assesses them against a number of requirements that we have defined as desirable for this kind of federation. However, none of these solutions addresses all the requirements. On the one hand, Kerberos federation support (called *cross-realm*) has several documented issues, such as the scalability of the trust model. Besides, it requires the establishment of an additional federation infrastructure, in addition to those that already exist (i.e. AAA-based and web-based). Besides, it lacks of a complete authorization management. On the other hand, Moonshot lacks of support for complete SSO, and requires the adaptation of all the application services to work.

This thesis aims to design solutions for the *Identity federation beyond the web* problem that address these requirements, overcoming the aforementioned proposals' issues. In particular, this document focuses on the analysis, design and implementation of ways to interconnect Kerberos with AAA-based federations. Kerberos possesses great features for intra-domain access control (e.g. secure, lightweight, integrated SSO, etc.). Besides, Kerberos is widely deployed and supported by most of current application services. However, as already commented, it has some weak points when it comes to federation or authorization support. On the other side, AAA-based federations have been used for decades to provide federated access to the network access service, with great success and presence. The integration of these two technologies offers an approach that supersedes

the *Kerberos cross-realm* operation with a more widely adopted federation technology. In this way, the benefits from both technologies are summed up, resulting into a better user experience, simplified deployment (no need to modify application services), and a better exploitation of resources (either, computational and network usage).

Besides federated authentication, this thesis focuses on the analysis and design of means to incorporate advanced authorization processing capabilities into Kerberos. While Kerberos does not define a way to perform fine-grained authorization, SAML has been used with great success and wide deployment for web-based identity federations. In particular, SAML advantages include its great acceptance by the community, its great flexibility to represent authorization data, or the availability of multiple tools for its generation and processing.

To achieve the integration of these technologies, the general objective of this thesis can be expressed as:

> *To analyse, design, and validate solutions that enable end users to bootstrap Kerberos credentials for a specific service provider, as the result of an authentication process conducted through the AAA-based federation, and an authorization process based on the exchange of SAML statements with the home organization.*

This objective has been approached in three different ways, depending on the target scenario. First, the standard Kerberos authentication mechanism can be extended in order to incorporate support for the use of AAA infrastructures. A second approach would be performing the federated authentication using a protocol, independent from Kerberos, with native support for AAA infrastructures, and bootstrap the Kerberos credentials from its result. Finally, a third alternative takes advantage of the federated authentication process performed to provide access to the network service (e.g. eduroam). As this process already involves the AAA federation, it could be used to bootstrap Kerberos credentials on the same service provider where the end user is attached to the network. This dissertation analyses, designs, and proposes a solution for each one of these bootstrapping approaches, named *FedKERB*, *PanaKERB*, and *EduKERB*, respectively.

FedKERB provides an architecture that integrates the Kerberos and AAA infrastructures by defining a new pre-authentication mechanism for Kerberos based on technologies such as GSS-API and EAP. In this way, Kerberos makes use of the AAA infrastructure to perform a federated authentication, and bootstraps new credentials for the end user based on the cryptographic material derived from the EAP authentication. Besides, the architecture includes the possibility of performing

advanced authorization after the authentication process, based on the transport of a SAML assertion containing identity information from the home organization's IdP to the service provider's Kerberos infrastructure. FedKERB provides a generic solution that accommodates all the requirements desired for an *identity federation beyond web* solution, including straightforward support for SSO within the boundaries of the service provider (i.e. intra-organization). All of this makes FedKERB adaptable to most of the foreseen *identity federation beyond web* scenarios. However, although FedKERB does not require application services to be modified, it does require the modification of the Kerberos pre-authentication process. While this would only imply updating a single entity (i.e. the KDC), this requirement might not be acceptable for some service providers and scenarios, where the deployed Kerberos infrastructure must be left intact.

PanaKERB defines an alternative to FedKERB that does not require any modification to the existing Kerberos pre-authentication mechanism. To achieve this objective, this contribution defines an architecture that uses an *out-of-band* protocol with native support for AAA infrastructures (in this case, PANA), to perform the federated authentication of the end user. This *out-of-band* process results into the bootstrapping of new Kerberos credentials, generated from the cryptographic material derived from the EAP authentication. The strength of PanaKERB resides on its simplicity, since it does not require the modification of any of the components already deployed on the service provider. However, it comes at the expense of introducing an additional protocol (i.e. PANA) and an entity (i.e. *PANA Authentication Agent* - PAA) into the architecture. We expect that this simplicity makes its adoption and deployment easier than for FedKERB, making of it a more suitable solution for those organizations that already have a Kerberos infrastructure deployed, and that are reticent to modify it to incorporate any of the changes required by FedKERB. Additionally, PanaKERB also allows the use of the advanced authorization model defined for FedKERB.

Finally, EduKERB provides a cross-layer architecture that integrates Kerberos with the AAA-based authentication performed during the access to the network service. This is achieved by defining a new Kerberos pre-authentication mechanism based on the results of a successful EAP execution. As these two authentication processes happen at different layers of the *Open Systems Interconnection* (OSI) model (i.e. link and application layers), this is called *cross-layer SSO*. EduKERB reduces the number of federated AAA-based authentication processes required to access application services, improving efficiency and optimizing the resource utilisation. Another of the relevant aspects of EduKERB, when compared to FedKERB and PanaKERB, is its applicability scope. Whereas the latter have a generic applicability, EduKERB has been designed to take advantage of the

specific network access authentication process defined for the GÉANT AAA federation, that is, the *eduroam's* RADIUS infrastructure. Nevertheless, it can be deployed in other scenarios based on RADIUS if desired. EduKERB requires more modifications to the existing infrastructures than FedKERB and PanaKERB. However, these modifications allow reducing the number of EAP authentications and the related overload in terms of network messages and computational time. EduKERB also allows the use of the advanced authorization model defined for FedKERB and PanaKERB.

In addition to defining these contributions, this thesis also provides a performance and functional analysis of them. This analysis consists of the definition of a performance model (describing the different time-consuming operations performed on each proposal) the implementation of three prototypes to demonstrate their functional viability and feasibility, and the measurement of their actual performance, based on the execution of these prototypes. Two major conclusions are extracted from the results of this analysis. On the one hand, it is demonstrated that the three proposals described in this thesis present similar execution times to the ones required for the network access service (which is used as a reference since it is the most extended use of AAA infrastructures). Moreover, as they have been implemented and executed in a real environment, their feasibility is also demonstrated beyond a theoretical stand point. Finally, it is confirmed that the three proposals perform very similarly, so the decision of using one or another will be based on the requirements of the deployment scenario.

Finally, this thesis discusses some interesting research topics that have been found worth being explored as future work during its realization, and that can provide further improvements in this thesis' area. These topics can be catalogued according to their expected addressing term, from those to be done in a short-term, to those envisioned in the long-term. Specifically, the proposed short-term topics include actively working on the *deployment of identity federation beyond the web technologies in real environments*; and the analysis, design and implementation of a solution enabling the *use of HTTP as out-of-band protocol for Kerberos*, instead of PANA. In the medium-term, the proposed topics are the analysis and design of solutions to extend the boundaries of the applicability of SSO from the current intra-organization SSO model to an *inter-organization SSO* one; and the analysis and design of *dynamic establishment and management mechanisms for AAA federations*, where trust relationships can be established in an automated fashion between members. In the long-term, this thesis proposes the analysis, design, and implementation of approaches for integrating the *IDaaS* concept with the identity federations beyond the web technologies that it discusses, as a way to simplify their deployment and spread their usage.

Acknowledgements

I would like to spend this page to thank those people that, one way or another, have helped and contributed to make this thesis a reality.

On the first place, to my wife Isabel. Thanks for all your love, support, sympathy, and patience. Thanks for making me so happy everyday. Without you, nothing of this would have made any sense.

To my parents that, since I was a child, have provided me with everything I needed to receive a good education and academic training.

To Gabi and Rafa, thesis advisors, but also colleges and friends, that with their wise advises and restless work have taken this project in the right direction. Thanks for investing so many hours in me, researching, proposing ideas, and reviewing papers and chapters. But also thanks for all the chats in the halls, the jokes and laughters, the football conversations... If a thesis is the result of a team work, I have had the advantage of playing with the best players. And I do not forget Fernando, member of this group for many years, and an indefatigable partner and friend in papers, meetings, and implementations. Thank you for your comradeship and generous help.

I would also like to thank Antonio, who introduced me into the researching world, opening the doors of one of the most prominent research groups of the University of Murcia. Thanks for having done the possible (and sometimes the impossible) for providing me and others with a job where we could grow as researchers, but also as persons.

To all those from the Faculty of Computer Science who have been my colleagues for all these years. Those who have been to my side since we started the degree, and that have shared so many things with me: Pedro J. Juan Antonio, and Pedro M. Those who have worked with me in the different projects I have been involved in: Manolo Bernal, Fernando Bernal and Elena. All the professors that have become colleagues, specially Gregorio and Diego. To my workmates from the *Dibulibu* and *T3* rooms, for creating such a remarkable workplace, where work and joy are mixed in equal proportions. And to Manolo Sánchez, who is no longer amongst us, but who taught me so much, both personally and professionally.

Finally, to Rhys Smith, for having hosted me during my stay at the University of Cardiff, and having guided and supported me in the work I performed there.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This first chapter gives a brief contextualization of the history and evolution of the access control systems in the area of application services, outlining the gaps, drawbacks and shortcomings that have motivated the work of this thesis. After that, it describes the specific problem statement of this thesis, including a general scenario where current access control systems cannot provide a satisfactory solution. Then, the main objectives of the thesis, as well as its main contributions, are described. Finally, the chapter details the structure of this document, and lists the publications that have resulted from the performed research.

1.1 Contextualization

From the beginning of the Internet, access control to application services (e.g. file sharing [1], electronic mail [2], etc.) has been one of the main concerns for system administrators. In general, when an organization (i.e. service provider) offers a specific set of services to a group of end users, one of its main concerns is assuring that the service will be provided only to those who are entitled to. For instance, there are some application services that are public, free of charge, and that can be used in an anonymous way (e.g. access to a weather web page). However, there are other application services requiring some degree of control over who is granted to access them, and under what conditions (e.g. access to a medical record database).

Service providers may have different motivations to perform access control to their services. For instance, an application service may only be available to those end users that have been previously enrolled and have a valid account. This is probably the most common reason to enforce access control. The enrolment of the end user is typically the consequence

of the establishment of a contractual relationship between that end user and the service provider. For example, this would be the case of on-line banking, social networks, and email providers.

Another motivation to desire access control is allowing the access to the service to only those end users who fulfil a specific set of conditions. A good example of this requirement would be the access to criminal history records, which should be granted only to law enforcement officers. Or the access to certain research publications, which is granted only to staff of some universities and research centres.

Furthermore, service providers may want to keep a record of who accessed the application service, and when. The motivations for having such a registry are several, ranging from the mere generation of accounting and statistical data, to being able to take countermeasures in case of misuse of the service, or to bill the end user for the use of the service. An example would be a *Voice over IP* (VoIP) service [3], where the end user is billed for the amount of time she is connected.

1.1.1 Access control functionality

Typically, access control to network application services consists of three different processes that match the aforementioned motivations: *authentication*, *authorization*, and *accounting* [4].

Authentication is the process by which a service provider verifies the identity of the end user who is trying to access the service. Typically, the service provider requires the end user to provide an identifier (e.g. a username), and challenges her to demonstrate the knowledge or the possession of one or more pieces of information associated to that specific identifier. In particular, these pieces of information (also called *factors*) can be a combination of the following types: *something the user knows* (e.g. a password or a pass-phrase), *something the end user has* (e.g. a smart card or mobile phone), or *something the end user is* (e.g. a fingerprint or an iris pattern).

Authorization is the process by which a service provider determines whether the end user should be granted to access the requested service. Typically, this process is split into two different sub-processes: the gathering of authorization information (e.g. end user attributes, environment variables, etc.), and the taking of the authorization decision (e.g. based on access control policies [5]).

Accounting is the process by which a service provider monitors the consumption of the resource performed by the end user along the lifespan of an authorized session. It is mainly used for billing purposes (e.g. duration of a video call, video streaming, etc.).

1.1.2 Per-application access control

The most deployed scheme for access control during the early days of the Internet, and still nowadays, takes place when each individual application service implements the whole access control functionality. That is, the application service directly verifies the factors provided by the end user, and makes the authorization decision based on the information stored in some internal database (e.g. files, SQL database, etc.). However, this scenario has a number of drawbacks for both, end users and service providers. On the one hand, end users need to maintain different set of credentials and attributes for each application service they access. This makes really difficult to remember which password needs to be introduced for a particular application service, or to maintain the coherence of the identity information across different application services whenever there is a relevant change. For instance, whenever the end user wants to change her postal address, it may require to manually update several dozens of accounts. On the other hand, application services need to implement the whole access control logic, and to maintain a database with the set of credentials and attributes for each possible user they have.

To alleviate these problems, service providers typically centralise the storage of credentials and authorization information into a shared database. In this way, end users only need to have an account per service provider, instead of per-application service. For instance, this happens in a typical campus scenario (e.g. the University of Murcia), where the same credentials and attributes are shared amongst all the application services (e.g. email, e-learning, Intranet...).

1.1.3 Centralising access control functionality: The IdP

A step forward in this simplification path is the outsourcing of (part of) the access control functionality from the individual application services to a centralised entity, typically called *Identity Provider* (IdP) [6]. This entity, deployed on the service provider, shares trust relationships with all the application services deployed on its domain. In this way, end users are actually authenticated by the same IdP, while application services receive the outcome of this authentication process, along with enough identity information (e.g. end users attributes) to allow them to make an authorization decision. Moreover, in some deployments the authorization decision sub-process can be performed by an independent entity called *Policy Decision Point* (PDP) [5], which manages the set of access control policies. Examples of protocols and technologies that allow the centralisation of the access control functionality are SAML [6], Kerberos [7], OpenID [8], OAuth [9], RADIUS [10], or

Diameter [11].

Another advantage of deploying an IdP is the possibility of providing *Single Sign-On* (SSO) [12]. As the authentication process for any of the application services deployed on the service provider's domain is always performed with the same IdP, the end user can be provided with some sort of temporary information (e.g. token, short-term credential, etc.) to simplify subsequent authentication processes to any of these application services. In this way, the end user needs to introduce her credentials (e.g. username and password) only during the first access. However, IdP technologies do not always support SSO. For example, SAML implements this feature based on HTTP cookies [13], while Kerberos defines the concept of tickets [7]. On the contrary, RADIUS and Diameter lack support of SSO.

Google Accounts [14] constitutes a clear example of IdP, where end users have a unique account to be used for all of the Google's services, and the access control logic is centralised on a single server (i.e *http://accounts.google.com/*).

At this point, service providers have simplified their identity management, but they are still forming identity silos. That is, identity information is both stored and used exclusively in a per service provider basis. Although to a lesser degree, such deployments still have the problems described above: end users need to manage several accounts for different service providers, and every service provider needs to deploy an IdP.

1.1.4 Interconnecting organizations: Identity federations

The impressive growth of telecommunications has promoted the establishment of business agreements between different service providers, the so-called *identity federations* (hereafter federations) [15], in order to increase the revenues of the deployed services. Indeed, such federations allow a service provider's end user to access to the services offered by any of the affiliated service providers within the federation. An identity federation defines and regulates the relationships between the different service providers, establishing how their application services and IdPs can interact to allow end users within the federation to access any of the provided application services.

To better understand how this federated access is possible, let us consider the following example, depicted in Figure 1.1. Let us assume the student *Alice* is enrolled in the IdP of an organization: *University A* (*Univ-A*). As a consequence of a enrolment process, that IdP securely provides her with an identifier (e.g. *alice@univ-a.edu*) and an associated credential (e.g. a password). From *Alice's* point of view, *Univ-A* is her *home organization*.

Besides, *Univ-A* belongs to an identity federation, named *edufed*. This federation consists of a group of organizations offering a set of application services to authenticated

and authorized end users within the federation. From *Alice*'s point of view, these organizations act as service providers. Then, let us assume that she wishes to access to an application service deployed at *University B* (*Univ-B*), also belonging to *edufed*. Typically, the process (see Figure 1.1) begins with *Alice* requesting access to the application service (1). *Univ-B* is not able to authenticate *Alice* by itself, as it does not have any registered information about her. However, by means of *Alice's* identifier, *Univ-B* knows that she belongs to *Univ-A* (2). Therefore, it redirects the authentication process towards *Univ-A*'s IdP (3), that verifies Alice's password. Assuming *Alice* is authenticated (4), *Univ-A* may provide additional authorization information to *Univ-B* (5), so that the latter can perform the authorization process, and customize the access to the application service (6), based on some policy rules. If the authentication and authorization processes conclude with success, the application service fulfils the *Alice*'s request (7).

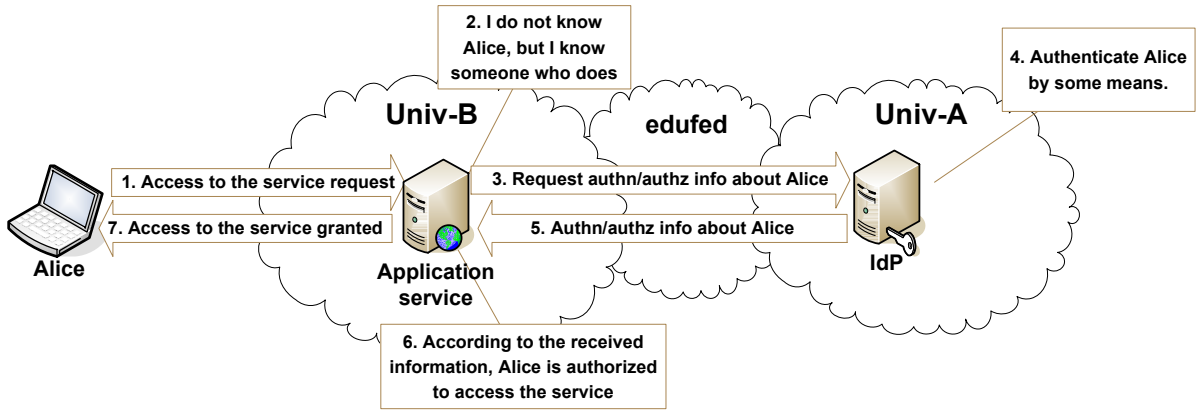


Figure 1.1: Generic identity federation.

This brief example summarizes how general identity federations work. However, each federation defines which protocols and technologies are used to fulfil these requirements. In particular, each federation needs to specify, among other things, how the service provider discovers the end user's home organization (e.g. based on the end user identifier), how the authentication information is redirected to the home organization's IdP (e.g. via web redirections), which authentication mechanism is used between the end user and the IdP (e.g. username and password), and how the authorization information is represented and transported between organizations (e.g. SAML).

Another important aspect of each federation is the kind of application services that are supported. In particular, two types of well-established identity federation follow this general model: *Web-based identity federation* and *Authentication Authorization and*

Accounting (AAA)-based identity federation. The former, as suggested by the name, focuses on providing federated access to web-based application services. For instance, several companies (Google [16], Amazon [17], Flickr [18], Microsoft [19], etc.) already support web-based federations by deploying technologies such as OAuth, OpenID, or SAML. On the other side, AAA is a generic access control framework which was conceived to support any type of application service in federated environments [20]. However, nowadays it is mostly used to control the network access service, that is, the connection to a communication network (e.g. 802.11 [21]). To support this type of federation, each organization deploys the so-called AAA server, interconnected with the AAA servers of the rest of member organizations, forming an AAA infrastructure. Authentication is typically performed by the use of the *Extensible Authentication Protocol* (EAP) [22], which allows the use of a extensible set of authentication mechanisms. AAA infrastructures are used by many organizations to provide federated access to the network service. For instance, roaming in cellular networks is managed through the use of such infrastructures [23]. Another example is eduroam (*edu*cation *roam*ing) [24], an AAA infrastructure providing Wi-Fi roaming to the members of the federation, which is composed of hundreds of international research and education organizations. Moreover, there are companies doing roaming federations based on AAA, such as IPass [25], Mach [26], and Syniverse [27]. These business federations integrate most of the world-wide WiFi, resulting into hundreds of millions of potential end users.

1.2 Motivation and problem statement: Identity federations beyond the web

The aforementioned situation has resulted into scenarios where organizations may have deployed two different federations, one based on AAA to provide federated access to the network access service; and the other one, based on web technologies, to provide access to web-based services. An example of such deployments can be found at many universities around the world, where the access to the network is managed by means of *eduroam* [24], while the access to the web applications is managed by other federations, such as the *Servicio de Identidad de RedIRIS* (SIR) [28]. But this raises a question: *What about other types of service?* In recent years several *beyond the web* scenarios, and the federated access to them, have attracted substantial notoriety and interest. Without doubt *Cloud* [29] and *Grid* [30] services are revolutionizing the way that organizations operate their *Information and Communications Technologies* (ICT) infrastructures, out-sourcing

to external service providers in an attempt to minimize management and operation costs, or to obtain computational powers or storage capacities that are otherwise unattainable. Other commonly used *beyond the web* services include electronic mail, remote file access, terminal access, or instant messaging. Besides, although some specific implementations of these services provide some level of support for AAA-based authentication, they are not standardized and, more importantly, lack of proper federation support that allow the secure exchange of credentials between the end user and the IdP at the home organization. That is, they are more intended for local deployments within a particular service provider. For these reasons, they cannot be included in any of the types of federations described before. Some additional example scenarios are mentioned in [31].

Therefore, this raises the necessity of another type of federation, which have been called *Identity federation beyond the web* [32], and that provides support to these application services. Note that the term *beyond the web* includes both, non web-based and web-based application services.

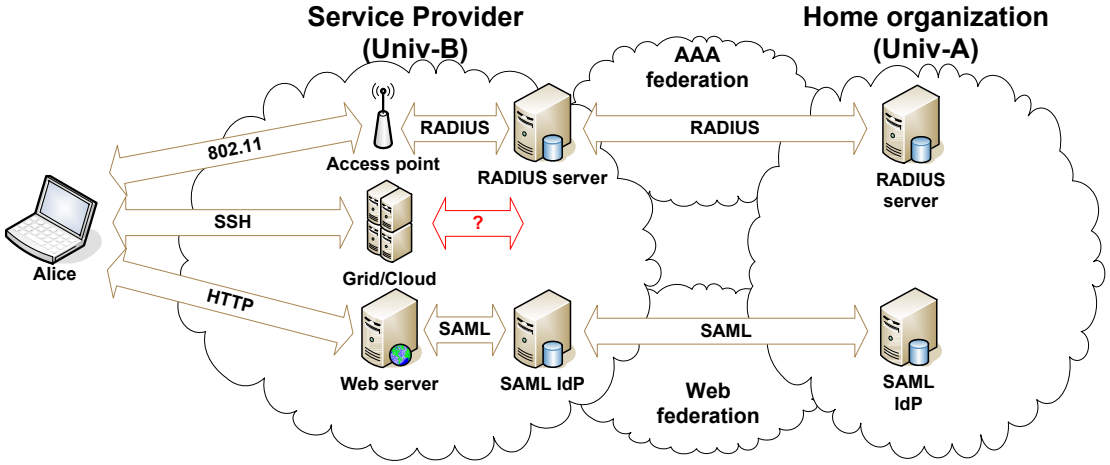


Figure 1.2: Example scenario for the *identity federation beyond the web* problem.

Figure 1.2 depicts a realistic scenario of this problematic. It is based on the example described in Figure 1.1. Let us imagine that *Univ-B* has three services available: a network access service, a Grid/Cloud service, and a web application. *SP-B* and *Univ-A* are members of an AAA-based federation, currently used to provide access to the network. In particular, the federation uses RADIUS as the AAA protocol. To take part of the federation, each organization has deployed a RADIUS server. Besides, both providers also belong to a web-based federation, used to provide access to web-based services. This federation is based on the SAML protocol. To take part of this federation, each one of them has deployed a

SAML IdP, and adapted their web applications to support this protocol. Hence, they have two types of federations deployed, one for the network access service, and the other one for web-based services. In fact, as said before, this scenario is common amongst universities and research centres around the world [24, 33].

Given this scenario, *SP-B* decides to enable Alice to access the Grid/Cloud service, by means of the SSH [34] protocol. However, as the SSH protocol does not support integration with any of the deployed federations, *SP-B* would need to create a (temporary) account for Alice, manually checking whether she accomplishes with the required authorization requirements. For instance, *SP-B* would need to create a new account for Alice (e.g. *alice-univ-a@sp-b.org*), and to communicate the password to Alice by some means (e.g. email). This is usually a tedious procedure for both, end users and administrators, that makes roaming of users and accessibility to the application services difficult, as it requires manual configuration. Besides, this breaks the notion of federation, since these processes should be performed in an automatic and seamless way.

Therefore, in this scenario, it would be extremely beneficial if end users could perform a federated access to the Grid/Cloud service or, in general, to any application service deployed in *Univ-B*.

In particular, taking into account the functional and security features already supported by the web-based and AAA-based technologies, a solution for *identity federations beyond the web* should ideally provide a combination of the best from them, considering fulfilling the following requirements:

(R1) *Authentication at the home organization.* Regardless of the type of application service, an end user's initial authentication must be performed with the IdP from her home organization. This implies that the end user never reveals long-term authentication credentials to service providers.

(R2) *Fine-grained authorization at the service provider.* After a successful authentication, the end user's IdP provides the application service with identity information (e.g. role, group, entitlement, language, age, etc.) about the end user. An application service may take an access control decision based on that information. A generic solution must support different kinds of authorization data to support a potentially diverse range of applications services.

(R3) *Data transport security.* The exchange of sensitive information such as credentials or identity information must be protected to assure its authenticity, integrity, and

confidentiality. That is, they must only be accessible by the intended recipient, but no one else.

(R4) *Single Sign-On.* SSO avoids the recurrent execution of potentially lengthy authentication procedures with the IdP each time the end user accesses an application service within the federation. The initial authentication is performed with the IdP, which may be located in a different organization than the service provider (see R1). This process yields further tokens that can be used to request access to other application services without necessarily requiring another authentication with the IdP in the home organization.

(R5) *Reuse of existing infrastructures and standards.* Re-using infrastructures and standards eases the deployment of a solution, since most of the components would have been already implemented and tested. Examples of standards typically re-used by federation solutions are AAA, PKI [35], TLS [36], or SAML. Besides, the fewer elements a solution requires to modify, the easier its deployment will be.

(R6) *Identity privacy.* Sometimes, end users want to access an application service without revealing their real identity. This can be achieved by the use of anonymous or pseudonymous identifiers. A generic solution should allow the use of these kind of identifiers, depending on the required level of privacy requested by the end user and offered by the service provider.

Unfortunately, only a few solutions for *identity federations beyond the web* have been defined. In this context, to the best of author's knowledge, one can consider *Kerberos* [7] and Moonshot [37].

Kerberos is a well-known standard protocol which is becoming one of the most widely deployed technologies for authentication and key distribution in application services. It provides a secure three-party protocol based on shared secret cryptography. Kerberos provides some features that makes of it a perfect candidate for services *beyond the web*. In particular, it stands out for the wide range of supported application services; its efficiency (based on symmetric cryptography); and its support for SSO (based on the distribution of tickets). Furthermore, Kerberos p a federated access operation mode called *cross-realm* [38]. This operation mode allows a client from an organization to obtain a ticket to access to an application service deployed on a different organization. However, whereas many service providers do use Kerberos to manage their own subscribers, they do not widely deploy Kerberos cross-realm infrastructures. In addition to some recognized issues

for the deployment of Kerberos cross-realm infrastructures [38], the main reason is that it would require the deployment of a parallel federation infrastructure (besides those already established for the access to web application services and the network access service). Maintaining different federation infrastructures requires a significantly high administrative effort (e.g. duplicate end user database, modification of firewall rules to allow Kerberos from outside, redundant establishment of trust relationships), as well as new failure points. This strongly goes against requirement *R5*. As such, the author has no knowledge of any currently deployed *cross-realm* infrastructure. Besides, Kerberos lacks of a complete authorization management that allows performing fine-grained access control in federated environments, precluding the addressing of *R2*. Due to these reasons, Kerberos cross-realm cannot be considered as a realistic solution for the stated problem.

On the other hand, Janet [39], the GÉANT community, and the TERENA EMC2 task-force[1] recently promoted the creation of a project called *Moonshot* [37], with the aim to specify a federated authentication and authorization architecture that, using the already existing AAA infrastructures for the network access service (i.e. eduroam), enables access control to most Internet application services. The ultimate objective is to enable end users to access application services using a AAA-based federation. To solve this problem, Moonshot defines a new mechanism for the *Generic Security Service Application Program Interface* (GSS-API) [40], which allows application services to authenticate the end user through an AAA infrastructure, and to retrieve authorization information from the home organization. As a consequence of the work in Moonshot, a new working group has been formed in the *Internet Engineering Task Force* (IETF) standardization organism [41], with the purpose of developing and standardizing the technologies required for implementing the designed identity federation architecture. This working group is called *Application Bridging for Federated Access Beyond Web* (ABFAB) [32]. Although Moonshot offers a valid solution for the stated problem, it does not provide an optimal solution. In fact, the weak points of Moonshot are the lack of support for SSO, not addressing requirement *R4*; and the requirement to adapt all the individual servers to include support for their newly defined GSS-API mechanism (against requirement *R5*). Section 2.4.3 provides further technical details about Moonshot.

[1] http://www.terena.org/activities/tf-emc2/

1.3 Objective of this thesis

Although current state of the art approaches provide valid solutions for the *identity federations beyond the web* problem, they present some gaps (especially related to R2, R4, and R5) that leave an open door for further research and improvements. Therefore, the main aim of this thesis is to design solutions for this problem that do not suffer from the issues of the aforementioned proposals, addressing all the requirements stated before.

As already commented, Kerberos provides great features for intra-domain access control (e.g. secure, lightweight, integrated SSO, etc.). Besides, Kerberos is supported by most of current application services. But it has some weak points related to federation or authorization support. On the other side, AAA-based federations have been used for decades to provide federated access to the network access service, with great success and presence. The integration of these two technologies offers an approach that supersedes the *Kerberos cross-realm* operation with a more widely adopted federation technology, allowing end users within the federation to authenticate without being previously enrolled on the Kerberos user database. In this way, the benefits from both technologies are summed up, resulting into a better user experience (e.g. SSO), simplified deployment (no need to modify application services), and a better exploitation of resources (either, computational and network usage). Therefore, this thesis focuses on the analysis, design and implementation of solutions to interconnect Kerberos with AAA-based federations.

Besides, this thesis also analyses and designs means of incorporating advanced authorization processing capabilities into Kerberos. While it does not define a way to perform fine-grained authorization, SAML has been used with great success for web-based identity federations. In particular, SAML advantages include its great acceptance by the community, its great flexibility to represent authorization data, the availability of multiple libraries and tools for its generation and processing.

To achieve the integration of these technologies, and fulfil the stated requirements, the general objective of this thesis can be expressed as:

> *To analyse, design, and validate solutions that enable end users to bootstrap Kerberos credentials for a specific service provider, as the result of an authentication process conducted through the AAA-based federation, and an authorization process based on the exchange of SAML statements with the home organization.*

This objective has been approached in three different ways, depending on the specific characteristics of the target scenario and the involved organizations. A

straightforward approach applicable for most cases would be extending the standard Kerberos authentication mechanism, in order to incorporate support for the use of AAA infrastructures. However, some organizations might have Kerberos infrastructures already deployed, and might be reticent to modify them. For those cases, another approach would try to maximize the addressing of R5 (minimizing the impact on the existing infrastructures), by performing the federated authentication using a protocol, independent from Kerberos, with native support for using AAA infrastructures. At the end of the authentication, new dynamically credentials would be enforced in the end user and the KDC. A third alternative takes advantage of the federated authentication process already performed in certain deployments to provide access to the network service (e.g. eduroam). As this process already involves the AAA federation, it could be used to bootstrap Kerberos credentials on the same service provider where the end user is attached to the network. Each one of these alternatives has resulted into the analysis and design of a solution during the realisation of this PhD.

Although the scope of this thesis is to provide solutions that are valid for any current or future AAA-based federation, it is worth noting that the motivational context for what the work of this thesis was originally conceived is the eduroam RADIUS infrastructure [24], and the access to the application services deployed within the GÉANT community [42]. This might be important to understand some of the design decisions and protocol choices made throughout this document. Section 6.5 provides some considerations about the usage of the eduroam's infrastructure for this purpose.

1.3.1 Specific objectives

According to the different approaches to perform the bootstrapping of Kerberos credentials enumerated above, the general objective of this thesis can be split into five more specific objectives:

(O1) To design a solution allowing the bootstrapping of Kerberos credentials on the service provider by enabling the Kerberos infrastructure to make a direct use of the AAA infrastructure to perform a federated authentication.

(O2) To design a solution enabling the bootstrapping of Kerberos credentials on the service provider as a result of a federated AAA-based authentication process performed by means of an independent protocol (*out-of-band*) with native support for AAA infrastructures, avoiding the modification of neither the elements of the Kerberos infrastructure, nor the application services.

(O3) To design a solution enabling the bootstrapping of Kerberos credentials as a result of the federated AAA-based authentication performed during the access to the network service, providing a *cross-layer* federated authentication for Kerberos that integrates the access to the network service with the access to upper-layer application services.

(O4) To design an authorization model enabling the Kerberos infrastructure on the service provider to be fed with end user identity information coming from the home organization after authentication, and to use it to make fine-grained access control decisions.

(O5) To validate the designed solutions by means of analytical models and prototype implementations, evaluating their functionality, security, feasibility and performance.

1.4 Contributions

In order to accomplish the objectives described in Section 1.3, this thesis provides three different contributions, named *FedKERB*, *PanaKERB*, and *EduKERB*.

- **FedKERB**. This contribution, described in Chapter 3, defines a solution integrating the Kerberos and AAA infrastructures by defining a new pre-authentication mechanism based on EAP for Kerberos. In this way, Kerberos makes use of the AAA infrastructure to perform a federated authentication, and bootstraps new credentials for the end user based on the cryptographic material derived from the EAP authentication. This addresses objective O1.

  Besides, the architecture includes the possibility of performing advanced authorization after the authentication process, based on the transport of identity information from the home organization's IdP to the Kerberos infrastructure in the service provider. When the end user tries to access to a particular application service, the Kerberos infrastructure can use this information to make an authorization decision. This addresses objective O4.

- **PanaKERB**. This contribution, described in Chapter 4, arises from the motivation of defining an optimized alternative to FedKERB, in order to avoid the modification of the existing Kerberos authentication mechanisms. For that, this contribution defines an architecture that uses an independent protocol (i.e. *out-of-band*) with native support for AAA infrastructures, to perform the federated authentication of the end user. This *out-of-band* process results into the bootstrapping of new

Kerberos credentials, generated from the cryptographic material derived from the EAP authentication. This addresses objective O2 of this thesis. This contribution also justifies the selection of *Protocol for Carrying Authentication for Network Access* (PANA) [43] as *out-of-band* protocol, over other alternatives.

Additionally, PanaKERB also allows the use of the advanced authorization model defined for FedKERB.

- **EduKERB**. This contribution, described in Chapter 5, defines a solution enabling a *cross-layer* authentication for Kerberos, based on the federated authentication process already performed to access the network service. In particular, EduKERB takes advantage of the network access authentication process defined for *eduroam* to bootstrap Kerberos credentials. This reduces the number of federated AAA-based authentication processes required to access application services, improving efficiency and optimizing the resource utilisation. This addresses objective O3 of this thesis.

  EduKERB also allows the use of the advanced authorization model defined for FedKERB and PanaKERB.

Besides, each contribution has been validated in terms of security, functionality and performance, addressing O5. The security validation can be found on each contribution's chapter, while the functionality and performance validation is described in Chapter 6.

Each one of these contributions has its own requirements, objectives, advantages and disadvantages, making them more appropriate for a set of specific scenarios than the others. Chapter 3, 4, and 5 will analyse these aspects with great detail.

1.5 Thesis structure

The rest of this document is organized as follows:

Chapter 2 provides a brief description of the most relevant technologies related to access control for different kind of services, and how their use can be applied in federated environments. Specifically, this chapter focuses on the access control to the network, web applications, and generic application services. Finally, this chapter introduces related state of the art proposals that deal with federated authentication and authorization solutions for application services.

Chapter 3 describes in detail the first of the contributions of this thesis: *FedKERB*. The chapter starts motivating the problem and describing the proposed architecture. After that, it enumerates the design possibilities to define an EAP-based Kerberos pre-authentication

mechanism, and details the interaction between the elements of the architecture to accomplish the required functionality. Finally, it discusses some aspects of the proposal that require attention, and provides a security analysis of the contribution.

Chapter 4 specifies *PanaKERB*, the second of the contributions of this thesis. This chapter starts with the motivation of the problem, and continues with the definition of the architecture, describing the involved elements, and the operational workflow. Finally, the chapter discusses some security considerations that have special relevance.

Chapter 5 details *EduKERB*, the third contribution of this thesis. Similarly to the previous chapters, it starts with the motivation of the problem. Then, it describes the architecture and the general operation of the proposal in detail. Finally it provides a thorough security analysis.

Chapter 6 provides a performance evaluation and functional validation of the contributions described in the previous chapters, in order to demonstrate their feasibility beyond their theoretical standpoint. This chapter provides a high level performance model describing the main time-consuming tasks that are required to complete the execution of each one of the proposals, describes the prototypes developed for this thesis, and provides an empirical performance analysis based on the results obtained from their execution.

Finally, Chapter 7 presents the conclusion of this work, and discusses the future work.

1.6 Related publications

The research work carried out in this thesis has led to the publication of different works at conferences, research journals, and IETF drafts. The most relevant contributions are presented, in chronological order.

Indexed Journals (JCR)

- R. Marín-López, F. Pereñíguez, G. López, and A. Pérez-Méndez. **Providing EAP-based Kerberos pre-authentication and advanced authorization for network federations**. *Elsevier Computer Standard & Interfaces*, 33(5):494-504, 2011. Listed in Q2 (28/104) of the JCR 2011, area of Computer Science, Software Engineering.

  This paper presents an analysis of the motivation for integrating Kerberos and AAA infrastructures, and provides a fist approach to the architecture of the FedKERB proposal.

- A. Pérez-Méndez, F. Pereñíguez-García, R. Marín-López, and G. López-Millán. **A cross-layer SSO solution for federating access to kerberized services in the eduroam/DAMe network**. *Springer International Journal of Information Security*, 11(6):365–388, 2012. Listed in Q4 (105/132) of the JCR 2012, area of Computer Science, Information Systems.

  This paper proposes a cross-layer SSO solution for kerberized services deployed in the eduroam federation, providing an extensive security analysis. It constitutes the main contribution to the EduKERB proposal.

- A. Pérez-Méndez, F. Pereñíguez-García, R. Marín-López, and G. López-Millán. **Out-of-band federated authentication for Kerberos based on PANA**. *Elsevier Computer Communications*, 36(14):1527 – 1538, 2013. Listed in Q2 (50/135) of the JCR 2013, area of Computer Science, Information Systems.

  This paper provides a solution for federated authentication for Kerberos based on PANA. It constitutes the main contribution to PanaKERB.

- A. Pérez-Méndez, F. Pereñíguez-García, R. Marín-López, G. López-Millán, and J. Howlett. **Identity Federations Beyond the Web: A survey**, *Communications Surveys & Tutorials*, IEEE, vol.PP, no.99, pp.1,1. doi: 10.1109/COMST.2014.2323430, 2014. Listed in the Q1 (3/135) of the JCR 2013, area of Computer Science, Information Systems.

  This paper provides a survey of identity federations beyond the web. In particular, it focuses on Moonshot and FedKERB, as they are the only two that are being discussed on the standardization bodies.

Conferences

- A. Pérez, F. Pereñíguez, R. Marín-López, G. López. **Federando Autenticación y Autorización en Servicios Kerberos mediante GSS-API y EAP**. *In JITEL 2011.*

  This conference provides an evolved version of the architecture of the FedKERB proposal.

- A. Pérez Méndez, F. Pereñíguez García, R. Marín López, G. López Millán. **Federación de servicios kerberizados en eduroam**. *In RECSI 2012.*

  This conference discusses the problem of federating the access to application services in the eduroam network, and provides a summary of the EduKERB proposal.

- A. Pérez, Fernando Pereñíguez, R. Marín-López, G. López y D.R. López. **Mejora del Protocolo RADIUS para Soportar la Fragmentación de Datos de Autorización**. *In JITEL 2013*.

  This conference paper provides a general description of the extension to the RADIUS protocol to support the fragmentation of packets with a length over 4096 octets.

Standardization groups

- A. Pérez-Méndez, R. Marin-Lopez, F. Pereñíguez-Garcia, and G. Lopez-Millán. **GSS-API pre-authentication for Kerberos**. *IETF Internet Draft*, draft-perez-krb-wg-gss-preauth-02, Sep 2012.

  The GSS-API pre-authentication mechanism required by the FedKERB proposal was detailed in this draft and presented to the IETF Kerberos WG.

- A. Pérez-Méndez, R. Marín-López, F. Pereñíguez-García, and G. López-Millán. **GSS-EAP pre-authentication for Kerberos**. *IETF Internet Draft*, draft-perez-abfab-eap-gss-preauth-01, Mar 2012.

  The FedKERB operation was described in this draft and presented to the IETF ABFAB WG as an alternative to the direct authentication with the application service using GSS-EAP.

- A. Pérez, J. Howlett, **Options for ABFAB-based Kerberos pre-authentication**. *IETF Internet Draft*, draft-perez-abfab-kerberos-preauth-options-01, March 2012.

  This draft, presented to the IETF Kerberos WG, discusses two design alternatives to provide EAP-based Kerberos pre-authentication: one based on the direct transport of EAP packets, and the other based on GSS-EAP.

- A. Pérez-Méndez, R. Marín-López, F. Pereñíguez-García, G. López-Millán, A. DeKok, and D. López. **Support of fragmentation of RADIUS packets**. *IETF Internet Draft (in WGLC)*, draft-perez-radext-radius-fragmentation-07, July 2014.

  This draft describes in detail a proposal for supporting the fragmentation of RADIUS packets, allowing the transport of large amounts of authorization data (over 4096 octets). This document was presented to the IETF RADEXT WG, and accepted as working item. Moreover, it is currently in *Last Call* status, and will likely be promoted as experimental RFC within the next months.

Chapter 2

Background and State of the Art

The research work described in this thesis document is built upon standard technologies and protocols. Therefore, knowing them and understanding how they work is of paramount importance to understand the following chapters of this document. This chapter provides a brief description of the most relevant technologies related to the access control problem for different kinds of services, and how their use can be applied to federated environments.

Specifically, Section 2.1 focuses on the access control to the network service, while Section 2.2 provides background on access control to web applications, and Section 2.3 describes other technologies that are able to provide these security services to generic application services. Finally, Section 2.4 introduces related state of the art proposals that deal with federated authentication and authorization to application services.

2.1 Access control to the network service

This first section describes the technologies that typically take part of the access control to the network service. Namely, these technologies are AAA infrastructures, used to allow the exchange of authentication, authorization and accounting information between the *home organization* and the *service provider*; EAP [22], which is used to perform authentication between the end user and the home organization; and *lower layer* protocols, which transport EAP packets between the end user and the service provider.

2.1.1 AAA protocols

The protection of the network service has been under great consideration by the standardization bodies. Indeed, several working groups (WG), such as the NASREQ WG[1] and the AAA WG[2], were created within the IETF with the purpose of analysing the security of the systems that control access to this and other services. These systems are typically known as *AAA infrastructures*, where AAA is the acronym for the three pillar processes that take part on the access control functionality: *Authentication*, *Authorization*, and *Accounting*:

- *Authentication.* This process verifies whether the end user is who she claims to be, usually after the validation of a credential (e.g. password, digital certificate or biometric information).

- *Authorization.* This process determines whether the end user has permission to access the requested resource (in this case, the network), based on the available information about her, like the role in the organization, date and time to perform the access, kind of available subscription, etc.

- *Accounting.* This process monitors the consumption of the resource performed by the end user along the lifespan of an authorized session. It is mainly used for billing purposes.

Although AAA infrastructures are widely deployed for network service, they were conceived to support any type of application service in federated environments [20].

The IETF defined the *generic AAA architecture* in [44], describing the components that must take part of the AAA infrastructure, as well as the possible ways of interactions between them to carry out the access control functionality under different scenarios.

This generic architecture is composed by the following entities:

- *End user.* The person or network equipment willing to access the network resource.

- *Home organization.* Organization where the end user has a subscription. It deploys the *home AAA server*, which is able to verify end user's credentials, hence asserting end user's identity.

[1] http://datatracker.ietf.org/wg/nasreq/
[2] http://datatracker.ietf.org/wg/aaa/

- *Service provider*. Organization where the network resource is actually being offered. It deploys the *service provider's AAA server* and the *specific equipment* for the services it offers. The *service provider's AAA server* communicates with the *home AAA server* to exchange AAA information.

- *Specific equipment*. It communicates with the *service provider's AAA server* in order to authenticate the end user and to obtain an authorization decision when access to a resource is requested. An example of specific equipment would be a Wi-Fi (802.11 [21]) *Access Point* (AP).

If the service provider and the home organization are not collocated, it is required that both belong to the same AAA-based federation, or that they have some kind of *Service Level Agreement* (SLA) [45] established, in order to allow the secure exchange of authentication and authorization information. They may also exchange accounting information to bill the end user for the use of the resource.

AAA-based federations are usually built following a hierarchical pattern, in such a way that every member of the federation does not need to know the IP address of the rest of members, but only those from its immediate father and children in the hierarchical tree (also known as *AAA proxies*). This scheme simplifies deployment and establishment of security associations, improving its scalability. Figure 2.1 shows an example of hierarchical organization. The thicker lines represent the path followed by the AAA messages exchanged between AAA server$_{112}$ and AAA server$_{132}$.

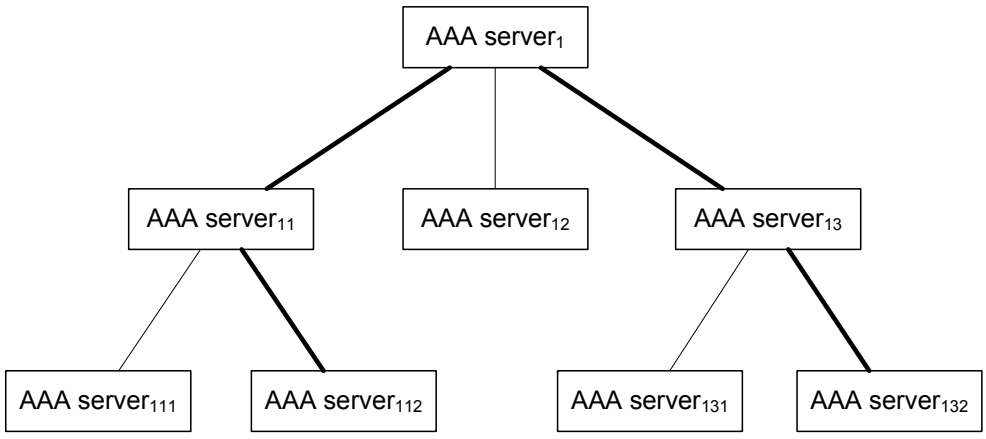


Figure 2.1: Hierarchical AAA scheme.

Having this in mind, a typical AAA protocol exchange would be as follows: an end user tries to access the network service using a Wi-Fi AP (specific equipment), providing

some information about herself (e.g. user name identifier). As the AP has no means to verify the identity of the end user, it sends a new AAA request to its local AAA server (service provider's AAA server) to ask for an authentication and authorization decision. In turn, if the end user does not belong to the service provider, the AAA request is forwarded to the corresponding home AAA server, which generates a AAA response. Several request/response exchanges may be required to successfully authenticate and authorize the end user. After that, if the process was successfully completed, the end user is granted with access to the network, and the Wi-Fi AP may generate accounting records that are sent to the home organization for billing purposes.

The AAA specification does not impose any particular authentication mechanism to be used, it just provides a transport for the information. Therefore, AAA can potentially be adapted to use any authentication mechanism. However, for each one of them, it would be required to define how authentication data is exchanged between the AAA servers. For this reason, EAP has become into the most popular authentication mechanism for AAA protocols, as it provides a middle layer between the authentication method itself and the AAA protocol. This allows using a wide range of authentication methods without requiring the modification of the AAA protocol. Besides, it adapts perfectly to the federated nature of AAA infrastructures.

Following subsections describe the two most relevant AAA protocols nowadays: RADIUS and Diameter. The former is the most broadly deployed and used, due to its longer existence and simplicity. The later appeared to solve the issues that RADIUS had, such as lack of confidentiality, datagram oriented transport and limited attribute and message size. However, the natural opposition to change already deployed and working systems, and its higher complexity compared to RADIUS, have limited its success for several scenarios. Both protocols support performing authentication by means of EAP.

RADIUS

RADIUS [10] is a protocol for carrying authentication, authorization, and accounting information between a *Network Access Server* (NAS), which controls access to network resources, and an *Authentication Server* (AS) which has information about the end user. The NAS and AS functionalities can be mapped to the *specific equipment* and *home AAA server* from the generic AAA architecture described above, respectively.

RADIUS is a client-server protocol, where the NAS plays the role of the client, requesting the authentication and authorization of end users trying to connect, and the AS acts as the server processing those requests. The authentication and authorization

processes are performed simultaneously, while accounting is optionally carried out in a later stage.

The protocol consists of the exchange of the so-called RADIUS packets, where information is conveyed in form of RADIUS attributes. There are four types of packets used for authentication/authorization:

- *Access-Request*. It is generated by the NAS, and typically contains information about the end user trying to access the service. Upon reception, the AS generates a reply using one among the following three packet types.
- *Access-Accept*. It is used to provide the NAS with a successful result of an authentication/authorization process. It may provide additional configuration information to be enforced in the NAS in order to adjust the service parameters.
- *Access-Reject*. It is used to provide the NAS with a failed authentication/authorization process. It may contain information about the reason of the failure, in order to allow the NAS to inform the end user.
- *Access-Challenge*. It is used to request additional information from the NAS, in case the one received so far is not enough to complete the authentication/authorization process. The *Identifier* field of the RADIUS header and the *State* attribute are used to tie together all the *Access-Request*/*Access-Challenge* packets belonging to the same conversation.

Besides, [46] defines other two packet types used to transport accounting information: the *Accounting-Request* packet, sent from the NAS to a RADIUS accounting server; and the *Accounting-Response*, sent from the RADIUS accounting server to the NAS.

The transport of RADIUS packets is performed over UDP (port 1812 for authentication/authorization and 1813 for accounting). The NAS may retransmit an *Access-Request* packet if no proper reply arrives within a reasonable time frame, as the packet is considered lost.

The NAS can directly send *Access-Request* packets to the AS if the IP address of the latter is known. However, if it is unknown, the packets can be sent instead to an intermediary server (called RADIUS proxy) that either delivers the packet itself or knows how to route them to its destination via other proxy. The use of proxies allows to create hierarchical structures of RADIUS servers, as described above in Figure 2.1.

RADIUS security is based on pre-established shared secrets between RADIUS servers. These secrets are used to encrypt the transmitted user passwords, and to provide integrity

protection and authentication of packets [10,47]. However, this security scheme has several drawbacks and vulnerabilities:

- Shared secrets between RADIUS servers are statically configured, making the addition of new members to the federation more difficult, thus limiting its scalability.
- When RADIUS proxies are used, there is not a shared secret between the NAS and the AS server. Instead, security is applied hop-by-hop, meaning that every proxy must be trusted as they have access to private information (e.g. user passwords or network keys).
- As RADIUS does not provide confidentiality (except for user passwords), an eavesdropper could examine the content of other RADIUS attributes that may contain sensitive information (e.g. delivery of EAP keying material [47]).

An additional problem of RADIUS is the limitation imposed to the length of attributes and packets. While attributes are restricted to a maximum of 256 bytes, packets can not exceed 4096 bytes. Both limits suppose a drawback that has been tried to be solved from the IETF in several occasions [47–49], although they are proposals for specific use cases and they do not provide a generic solution. Currently, the RADEXT WG [50] is working on a packet fragmentation proposal [51] to allow the exchange of large amounts of authorization information between RADIUS entities.

To overcome the enumerated security flaws, TLS has been proposed as an optional transport for the RADIUS protocol [52], in order to provide stronger security protection to packets, establishing a direct security association between the NAS and the AS. Nevertheless, a completely new AAA protocol called Diameter has been designed within the IETF, to solve these security issues and to improve the protocol design to adapt it to new requirements.

Diameter

Diameter [11] arises to overcome the problems and deficiencies that RADIUS has. This issues make RADIUS hardly usable for some of the new scenarios that have been appearing with the pass of the time, and that have higher requirements in terms of transport capabilities and security. Diameter uses a connection-oriented transport protocol (TCP or SCTP [53] on port 3868), since the reduced overhead of using connectionless protocols is no longer an important aspect with nowadays equipments. It also allows longer packets and attributes than RADIUS, providing natural support for the exchange of large amounts

of data. Finally, it also incorporates stronger security mechanisms, such as TLS [36], DTLS/SCTP [54], or IPsec [55] by default. While TLS and DTLS support is mandatory, supporting IPsec is optional. Nevertheless, using one of them for securing the exchange of messages is mandatory.

The protocol consists of the exchange of messages between the Diameter client and the Diameter server, which are used for carrying AAA information between them. The Diameter client and server functionalities can be mapped to the *specific equipment* and the *home AAA server* from the generic AAA architecture described above, respectively.

Instead of defining different message types, Diameter uses the concept of *command* to specify the expected behaviour upon reception. Each command consists of a request and its corresponding answer. Table 2.1 provides a brief summary of the main Diameter commands defined in the base protocol specification.

Table 2.1: Common Diameter Commands.

Command	**Abbreviation**	**Description**
Capabilities-Exchange-Request /Answer	CER/CEA	Discovery of a peer's identity and its capabilities.
Disconnect-Peer-Request /Answer	DPR/DPA	To inform the sender's intention to shut down the connection.
Re-Auth-Request /Answer	RAR/RAA	Sent to an access device (NAS) to solicit user re-authentication.
Session-Termination-Request /Answer	STR/STA	Sent to the server to inform the provision of a service to a user
Accounting-Request /Answer	ACR/ACA	To exchange accounting information between Diameter client and server.

The Diameter base protocol specification defines a minimal set of elements to allow the exchange of AAA information between the parties. Specifically, it provides the definition of the format for messages and *Attribute Value Pairs* (AVPs) transporting the information. It also defines a basic collection of messages, commands and attribute types, which allow performing accounting tasks. However, it does not provide authentication or authorization support for any particular application service. This support has to be provided by means of the so-called *Diameter applications*. Diameter applications define the commands and AVPs that are used for a particular application or protocol. New commands and AVPs can be defined within the context of an application, although re-utilisation of the existing ones when possible is highly recommended.

There exist several Diameter applications already defined for the most typical kind

of services, like the *Network Access Server Requirements Application* (NASREQ [56]), which satisfies typical network access services requirements and provides a mean for interacting with legacy (RADIUS) systems, or the Diameter Extensible Authentication Protocol Application (Diameter-EAP [57]), which defines how to transport EAP packets from a NAS to a EAP server. A Diameter client or server is referred to as *Diameter X Client/Server* when fully they support application "X" (e.g. Diameter EAP Client).

2.1.2 EAP

As mentioned in Chapter 1 and in previous subsections, the *Extensible Authentication Protocol* (EAP) [22] is a protocol designed to perform authentication by means of a extensible set of mechanisms and technologies, through the so-called *EAP methods*. As depicted in Figure 2.2, three different entities might be involved in an EAP authentication process: the *EAP peer*, the *EAP authenticator* and the *EAP server*. The *EAP peer* is the subject of the authentication, the *EAP authenticator* is the entity interested in the outcome of the authentication process, and the *EAP server* is the entity that knows how to authenticate the *EAP peer*. The EAP server can be either co-located with the EAP authenticator (*standalone configuration*), or with a backend AAA server (*pass-through configuration*) [22]. In the latter case, the *EAP authenticator* does not take part on the authentication process, and simply acts as a mere intermediary between *EAP peer* and *EAP server*.

Since EAP is the most used authentication mechanism in AAA protocols, its entities can be mapped in a one-to-one relation with those from the AAA architecture. In particular, the *EAP peer* functionality usually lies on the *end user*, while the *EAP authenticator* one is performed by the *specific equipment* (pass-through configuration), and the *EAP server* is implemented by the *home AAA server*.

An EAP conversation consists of several request/response packets exchanged between the EAP peer and the EAP server, through the EAP authenticator. In order to convey packets between the EAP peer and the EAP authenticator, an *EAP lower-layer* protocol is required. Section 2.1.3 provides more details about lower layer protocols. In the pass-through configuration, the help of an auxiliary AAA protocol is required to transport packets between the EAP authenticator and the EAP server. The specifications that define how EAP is transported in RADIUS and Diameter can be found in [47] and [57] respectively. Section 2.1.1 describes AAA infrastructures and protocols.

The EAP authenticator usually starts the authentication process by requesting the EAP peer's identity through an *EAP Request/Id* packet. The EAP peer answers with an

EAP Response/Id containing its identity. With this information, the EAP authenticator determines the EAP server that will handle the authentication and forwards the packet to it through the AAA infrastructure. Upon the reception of this packet, the EAP server selects the authentication method to be performed. The method execution typically involves several *EAP Request/Response* exchanges between the peer and server. Finally, the outcome of the process is indicated with either an *EAP-Success* or an *EAP-Failure* packet.

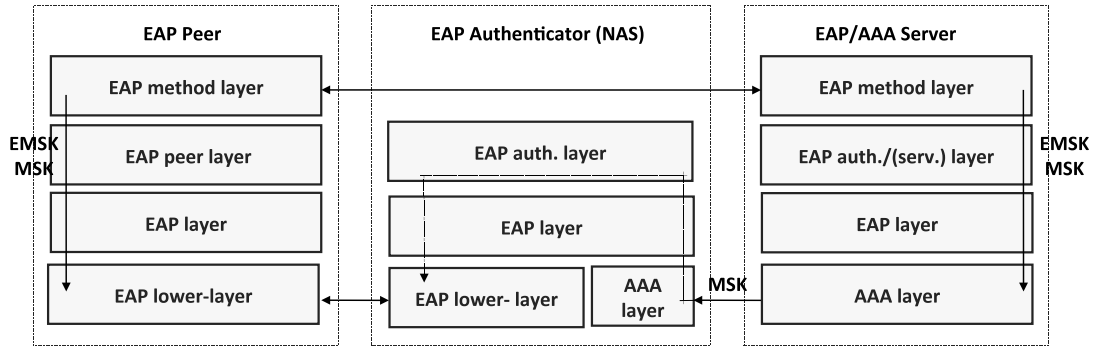


Figure 2.2: EAP Framework Model

There are a wide variety of standard EAP methods, each one showing different security properties. One of the most simple ones is the EAP-MD5 [22] method, where the EAP peer must return an MD5 digest computed over a challenge received from the EAP server and its own password. This method is susceptible to man-in-the-middle and dictionary attacks [58], and does not provide mutual authentication. A more secure approach is provided by the EAP-TLS [59] method, where a TLS security association is established between EAP peer and EAP server. This method provides mutual authentication, and avoids the man-in-the-middle situation. However, the EAP peer must be configured with an X.509 [35] certificate for authentication, something that is usually a hassle. Finally, the tunnelled methods such as EAP-TTLS [60] or PEAP [61], establish a TLS channel between the EAP peer and EAP server. This channel is then used to encapsulate a simpler authentication method such as CHAP, PAP or MD5 digests, where the end user's identifier and password are transmitted. Other relevant EAP methods are EAP-AKA [62], EAP-SIM [63], and EAP-IKEv2 [64]

After a successful authentication, some EAP methods are able to generate keying material. According to the *EAP Key Management Framework* [65] two keys are exported: the *Master Session Key* (MSK) and the *Extended Master Session Key* (EMSK). While the MSK is typically provided to the authenticator to establish a security association with the peer (e.g. layer2 security in Wi-Fi networks), the EMSK must be kept in secret between

the EAP peer and EAP server. In particular, the EMSK has been proposed [66] to derive further keys, called *Usage-Specific Root Keys* (USRK), which can be used for different purposes (e.g. to provide security to specific applications).

As nothing precludes the EAP server and EAP authenticator to belong to different organizations, EAP is intrinsically prepared to operate in federated environments. In this way, an end user belonging to a particular organization (*home organization*) can use EAP to be authenticated to access the network in a different organization (*service provider*). For this operation it is required that both organizations establish a trust relationship which allows the usage of a AAA protocol to transport the EAP packets. This is further described in section 2.1.1. A successful example of network federation based on EAP is *eduroam* [24].

Besides the access control to the network service, the usage of EAP to perform authentication for other services and scenarios is under consideration in the IETF [67].

2.1.3 Lower layer protocols

As explained in Section 2.1.2, the EAP peer and authenticator require of a protocol to transport the EAP packets between them, in order to complete a successful authentication (e.g. wireless links). These protocols are denoted as *lower layers*, and the requirements they have to fulfil, as well as their expected behaviour, are described in section 3 of [22].

There exist a variety of lower layer protocols, as several network access technologies have adopted EAP as their authentication protocol. Besides, some network layer protocols have opted to include EAP support in order to provide extensible authentication methods. This subsection focuses on those technologies and protocols that have a bigger relevance, either because they are widely used, or due to their importance in the context of this work.

802.1X

The 802.1X specification [68] is an access control model for IEEE 802 *Local Area Networks* (LAN). It is based on the use of EAP to allow different authentication methods. It defines three components: the *supplicant* willing to access the network, the *authenticator* providing the service, and the *authentication server*, which authenticates the supplicant. These entities play the role of EAP peer, authenticator and server respectively.

The access control mechanism is performed in terms of *ports*. A port represents the association between the supplicant and the authenticator. When it is *open*, the supplicant can only communicate with the authenticator to perform the EAP authentication, nothing else. Once the authentication process has been completed, the port state is changed to *closed* meaning the access to the network is granted. The main problem of this access

control model is that the cryptographic material derived from the EAP authentication is not used to establish a security association between the supplicant and the authenticator. This is something that forthcoming protocols, such as 802.11 [21] and its 4-way handshake, imposed for improved security.

In addition to this access control model, the 802.1X specification also defines the *EAP over LAN* (EAPOL) protocol. This lower layer protocol determines how EAP packets are encapsulated into IEEE 802.11 frames, and how they are exchanged between the supplicant and the authenticator.

802.11

The 802.11 specification [21] defines a set of standards for IEEE Wireless LAN. One of the aspects covered by this specification is the access control to the network, for which originally two models were proposed. The *Open System Authentication* model imposes no authentication. Using this model, any station able to associate with the AP is automatically granted to access the network. On the contrary, the model based on the *Wired Equivalent Privacy* (WEP) protocol requires the station and the AP to know a pre-shared secret. However, WEP encryption has been proven to be insecure [69], motivating the creation of the 802.11i standard [70], which tries to overcome the aforementioned security issues.

As a result of the work on 802.11i, the *Wi-Fi Protected Access* (WPA) was released in 2003. WPA is an interim specification on a preliminary version of the 802.11i draft. In 2004, the final version of 802.11i (denoted as WPA2) was released. While WPA uses the RC4 [71] encryption algorithm, WPA2 uses the stronger AES [72].

802.11i defines an authentication process divided into three phases:

1. *IEEE 802.11 association phase.* During this phase the station discovers the capabilities offered by the AP and associates with it.

2. *IEEE 802.11 authentication phase.* This phase allows to make use of two different methods, one based on pre-shared keys and the other based on 802.1X specification. If the latter is used, EAP is executed between the station (acting as EAP peer) and the AP (acting as EAP authenticator). The EAP server can be either collocated with the AP or be deployed as a separated AAA server (i.e. RADIUS or Diameter). A *Pairwise Master Key* (PMK) is derived from either the pre-shared key or the MSK exported by the EAP method.

3. *IEEE 802.11 security association phase.* This phase consists of the execution of the *4-way handshake* protocol, using the PMK derived from the previous phase. As a

result, a *Pairwise Transient Key* (PTK) and a *Group Transient Key* (GTK) are derived to protect traffic between the station and the AP. Besides, the port in the AP is set to *closed* state.

802.16

The 802.16 specification [73] is a set of standards authored by the IEEE focused on wireless broadband provisioning in metropolitan areas. It is also known as *WiMAX*, a commercial name given by the *WiMAX Forum industry alliance*.

On its first version, 802.16 was intended for fixed scenarios where a *Subscriber Station* (SS) does not move from one *Base Station* (BS) to another. The 802.16e [74] specification arises to overcome this limitation and to improve the access control mechanism for either, fixed and mobile scenarios. In this sense, the PKMv2 (Privacy and Key Management) protocol was defined to provide mutual authentication and secure key distribution between the SS and BS. Authentication can be performed by means of public key encryption (i.e. RSA) or by means of EAP. In the latter case, the SS plays the role of EAP peer while the BS acts as the EAP authenticator. Depending on the specific configuration, the EAP server can be located either in the BS or in a dedicated AAA server.

Similarly to 802.11, after a successful authentication a PMK is derived from the MSK by the SS and the BS. Afterwards, they perform a 3-way handshake process to generate a TEK (Traffic Encryption Key) used to protect network traffic in the wireless link.

802.21a

The IEEE 802.21 [75] standard specification aims to enable a seamless movement between heterogeneous networks using different technologies (also known as *Media Independent Handover - MIH*). The network service is provided by each provider through the so-called *Point of Attachment* (PoAs). Hence, access control to the network would be determined by the underlying technology of each one of these PoAs (e.g. 802.11, 802.16...). Besides, network providers deploy some *Point of Service (PoS)* elements, which support remote events and command services to aid in the handover process.

To allow this media independent handover (MIH) each network element implements the following services, which provide information relevant to the handover: *Media Independent Event Service* (MIES), that provides information about changes in link characteristics or quality; *Media Independent Command Service* (MICS), that allows upper-layers to control and manage link-layer behaviour; and *Media Independent Information Service* (MIIS), that

manages information about the network topology, discovering available neighbour access networks and PoAs.

There are several task groups within the IEEE aiming to define extensions to 802.21. Specifically, IEEE 802.21a is defining mechanisms to protect the MIH message exchanges performed between the MN and the PoSs. In particular, the solution described in [76] proposes the use of EAP to perform the MN authentication prior accessing the PoS services, and to make use of the exported keying material to derive further keys to protect the subsequent MIH messages.

IKEv2

The *Internet Key Exchange* (IKE) protocol was designed in order to automate the IPsec [55] security association establishment. Its first version was defined in three IETF RFCs (2407, 2408, and 2409). However, this version suffered of some limitations and complexity, motivating the proposal of a second version. The result was the IKEv2 [77] protocol.

The IKEv2 protocol is executed between two parties: the *initiator* and the *responder*. The initiator starts the IKEv2 protocol, whereas the responder acts as server during the negotiation. The protocol is composed of a well defined set of four main exchanges (*request-response*), namely: IKE_SA_INIT, IKE_AUTH, CREATE_CHILD_SA and INFORMATIONAL. Though the IKEv2 protocol uses a non reliable transport protocol (UDP), the concept of exchange allows to ensure reliability, as there is an expected and well defined response for each request.

The IKE_SA_INIT exchange establishes a security association (SA) at IKE level, named the IKE SA (IKE_SA), between the participant entities. This IKE_SA will protect all the following IKE exchanges. Once the IKE_SA is established, an IKE_AUTH exchange is performed in order to authenticate the parties and create the first IPsec SA (CHILD_SA) between them.

In addition to the authentication methods already supported by IKE (i.e. pre-shared key and public key cryptography), IKEv2 adds support for using EAP. The transport of the EAP packets is performed through several IKE_AUTH exchanges between the initiator (playing the EAP peer role) and the responder (acting as EAP authenticator). The keying material exported by the EAP method (if any) is used in the key derivation process defined by IKEv2 to generate the IPsec security association keys.

Differently from the previously described network access technologies, which operate at link layer (L2), IKEv2 operates at application layer (L5). This aspect allows the EAP peer and the EAP authenticator to be deployed in different physical networks, even using

different network access technologies.

PANA

The *Protocol for carrying Authentication for Network Access* (PANA) [43] is a lower layer protocol designed to transport EAP packets between an EAP peer and an EAP authenticator over IP networks. Specifically, it operates on top of UDP, making PANA independent from the underlying network access technology.

The PANA network access control model considers a *PANA Client* (PaC) which requests access to the network service offered by an *Enforcement Point* (EP), such as an AP or a router. The EP is controlled by a *PANA Authentication Agent* (PAA), responsible for authenticating and authorizing PaCs which are requesting access to the network service. Through AAA protocols, the PAA communicates with the *Authentication Server* (AS), an entity which is in charge of verifying the credentials provided by a PaC. The AS functionality is typically implemented by a AAA server. If the AS correctly verifies the credentials, it sends authorization parameters (cryptographic material, network access lifetime, etc.) to the PAA which, in turn, transfers some configuration information to the EP by using either an Application Program Interface (API) or a Configuration Network Protocol (CNP) such as SNMP [78]. According to this operation, the PaC, PAA and AS implement the EAP peer, EAP authenticator and EAP server functionalities, respectively.

The PANA operation is developed along four different phases. In first place, the *authentication and authorization phase* is initiated by the PaC through a *PANA-Client-Initiation* message sent to the PAA. In this phase, the PaC and the PAA exchange several *PANA-Auth-Request/Answer* (PAR/PAN) messages in order to negotiate some parameters such as the integrity algorithms used to protect PANA messages. They also exchange PANA messages transporting EAP, which are forwarded to the AAA server to perform the authentication. At the end of this phase two security associations are established: on the one hand, a *PANA security association* (PANA SA) is established between the PaC and PAA in order to integrity protect PANA messages; on the other, a *PaC-EP SA* is generated by performing a security association protocol between the PaC and an EP to protect data traffic.

2.2 Access control to web applications

Nowadays, the World Wide Web (WWW) [79] represents a special case of application service, due to its growth and wide extended use. Hence, it is worth of being discussed in

a separated section from other application services.

Along the years, the WWW has became widely used, due to its versatility and user-friendly operation. It serves as a platform where the so-called web applications offer end users a wide variety of services, ranging from the basic newspaper publication to the most complex social network.

The following subsections describe different access control mechanisms used to perform authentication and authorization in web applications, from the basic HTTP authentication to more advanced federated authentication and authorization mechanisms.

2.2.1 HTTP authentication

Originally, the HTTP protocol [80] provided the *Basic Access Authentication* scheme, intended to provide a basic access control to web applications. This scheme defines a means by which a web server can request the end user's user agent (e.g. web browser) to provide valid user credentials, before granting access to a protected resource. The web server checks whether the provided credentials match with the ones stored in some database (e.g. local file), and authorizes access to the requested resource in such a case. However, this scheme is not considered secure since the credentials are provided in clear-text, requiring the use of a secure transport (e.g. TLS channel).

To overcome the more serious flaws of the *Basic Access Authentication*, the IETF defined the *Digest Access Authentication* [81]. In this scheme the credentials are never sent in clear-text over the network. Instead, the web server provides a challenge (i.e. a nonce value) to the user agent, which replies with the result of a hash function (typically MD5) computed over the username, the password, the given nonce, the HTTP method, and the requested URI. Although the password is not sent in clear-text, this mechanism does not provide a great security as it does not allow mutual authentication and it is vulnerable to man-in-the-middle attacks. Therefore, a secure transport such as TLS would be required as well.

Using any of these mechanisms, the web server is responsible of the authentication procedure (i.e. assure end user identity), though any relevant identity information about the end user must be managed by each particular web application associated to the end user's identifier.

2.2.2 Web forms based authentication

In opposition to the HTTP authentication, where access control is performed at the HTTP protocol level, a different alternative consists in leveraging authentication to each individual web application. For doing that, the web application prompts unauthenticated users with a web form where they must provide their credentials. The web application processes the received credentials and matches them with some application-specific database. If the authentication succeeds, the user is provided with a session identifier, which must be presented by the user agent on every subsequent request made to the web application. The lifetime of this session is limited, hence after a configured amount of time, the end user will be required to authenticate again. The session identifier can be stored by the user agent into a *persistent cookie*. This allows re-using the session identifier in future accesses to the same web application within the session lifetime.

This authentication mechanism provides the web application with a finer control over the authentication/authorization process, as it simplifies other processes such as logout, or password update, which are now controlled by the web application and not by the web server.

It is recommended to combine this method with the use of TLS in order to protect the transmission of the html form, and to avoid man-in-the-middle attacks.

2.2.3 Federated operation

Web applications may decide to take part of a federation (Section 1.1) in order to decouple access to the resource from the end user authentication process. In this case, the latter is delegated into a third party entity called *Identity Provider* (IdP). This IdP is in charge of managing end user's credentials and identity information. This federated access allows: a) web applications to not worry about how the authentication is performed (e.g. plain login/password, digest-based authentication, public key cryptography, smart cards or biometry), and focus on the content delivery; b) end users to use a single identity through multiple web applications; c) end users to be able to access to those federated applications after performing a single authentication process, benefiting from SSO, and considerably improving the user experience.

Such a federated scenario requires that both, web applications and IdPs, agree on common technologies to represent and exchange authentication and authorization information. There are several standards aiming to provide a solution for some of these federation aspects. Although they are intended to solve different scenarios, and they use

different mechanisms, they all can be summarized under the following generic operation scheme:

1. The end user accesses a specific web application.
2. At some moment, the web application requires additional identity information about the end user (e.g. authentication token or a specific set of identity attributes) that must be provided by the end user's IdP.
3. The web application redirects the end user agent (e.g. using HTTP redirections [80] or HTTP GET/POST) to the IdP. This HTTP request include which web application is making the request, and what information is being requested.
4. The IdP authenticates the end user making use of one of the mechanisms described above, and prompts her for consent to disclose the requested information to the web application.
5. If the end user agrees, the IdP redirects the user agent back to the web application, including one or more statements with the information required by the web application.

The following subsections provide a brief description of the most relevant technologies supporting web federation, describing their target use cases and the functionality provided.

SAML

The *Security Assertion Markup Language* (SAML) [6] is a XML-based security specification defined by the *OASIS Security Services Technical Committe* [82]. It is intended to, on the one hand, provide a means to represent authentication and authorization statements; and on the other hand, to define how this information is generated, exchanged, and processed by the different involved parties.

SAML defines *Assertions* as the security statement format. Each *Assertion* usually includes the issuer's identity (*Issuer*), the subject's identity (*Subject*), and the set of conditions under which the statement is valid (e.g. validity period or recipient identity). Besides, different kind of information is included in the assertion depending on the type(s) of statement to encode. *Authentication statements* assert that the included *Subject* was previously authenticated by an authority in a particular context, and they are mainly used in SAML-based SSO scenarios [12]. *Attribute statements* assert that the *Subject* is associated to the supplied attributes. Attributes can define a role, a group membership

or relevant information for each entity. Finally, *Authorization Decision statements*[3] assert whether an entity can gain access to a specified resource based on some evidence, usually another statement.

The SAML specification also defines the concept of *Protocol*. A *Protocol* describes how SAML elements (e.g. Assertions) are packaged into *SAML request* and *SAML response* messages, and the processing rules associated to their generation and processing. Usually SAML *Protocols* follow a request/response scheme, and are used to retrieve a specific kind of statement. For example, the *Authentication Query*, *Attribute Query*, and *Authorization Decision Query* protocols are used to request their homonyms statements.

In order to define how a SAML protocol message is transported from one party to another, the SAML specification defines the concept of *Binding* [83]. The most relevant *bindings* defined for SAMLv2 are *SAML SOAP*, *Reverse SOAP (PAOS)*, *HTTP Redirect (GET)*, *HTTP POST*, *HTTP Artifact*, and *SAML URI*.

Finally, SAML defines several *profiles* [84] as a set of rules describing how assertions, protocols and bindings are combined to fulfil the requirements of a specific scenario. One of the most important SAML profiles is the *Web Browser SSO Profile*, describing how an end user gets authenticated with a service provider by means of the *Authentication Request Protocol*, the *HTTP GET/POST* bindings, and the use of the *Authentication statement*.

Using the XML signature and the XML encryption specifications [85], a SAML message can be signed or encrypted by its issuer. This provides a means for authentication, integrity, and confidentiality for SAML messages, avoiding eavesdropping and tampering. This security model is based on public key encryption, therefore the assertion consumer requires to have access to the issuers public key to validate the signature. This can be achieved by several means, like manual configuration of trusted public keys on each peer, its automatic distribution along with the signed message, or the use of a Public Key Infrastructure (PKI).

SAML-based federations are built around the concept of *metadata* [86]. Using a specific SAML profile requires that involved parties agree on a set of parameters, like identifiers or supported bindings. Besides, they need to know some details about the other entities, such as their certificates, their public keys, or the endpoints or URIs where users should be directed to perform a specific action. This meta-information is represented in an standardised way using the *SAML Metadata* specification [86]. In a similar way to public keys, each member of the federation needs to have access to other members' metadata in order to be able to communicate with them. This can be achieved by manually establishing the metadata on each entity of the federation. Alternatively, metadata information can be

[3]Since SAMLv2, the authorization decision feature (statement and query) has been frozen; if more functionality is desired, it is recommended that XACML be used.

made available online to the members of the federation through a trusted and well-known server, typically called *Metadata Server* (MDS).

There are several identity federation solutions based on SAML, such as Shibboleth [87], SimpleSAMLphp [88], or Microsoft Info Card [89]. In [90], readers can find an extensive survey of SAML-based identity federation solutions.

XACML

The *eXtensible Access Control Markup Language* (XACML) [5] is the OASIS proposal for a standard access control language and protocol. It is based on XML, and it includes two different specifications: the first one is an access control policy language, which defines the set of subjects that can perform particular actions on a set of resources; the second one is a format to encode access control requests and responses. The standard also describes an extensible architecture that can be adapted to different scenarios and policy types. In this sense, two important elements must be highlighted: the *Policy Enforcement Point* (PEP) which controls the access to the resource and enforces the decision taken; and the *Policy Decision Point* (PDP), which evaluates policies and takes a decision based on the available information.

OpenID

OpenID [8] is an open source standard for SSO authentication and data management in web applications. End users register with an OpenID IdP of their choice, where they establish their credentials (i.e password) and a set of identity attributes. Each end user is identified unequivocally by a URL or XRI [91] assigned by the IdP. The functionality of this identifier is twofold. On the one hand, it allows the OpenID provider to distinguish between the different registered users. On the other, it makes possible for service providers to resolve the location of the IdP.

In this way, end users do not need to remember traditional authentication credentials such as username and password for every service provider. Instead, when they try to access a web application, they are redirected to their IdP to be authenticated. Besides, they are prompted to indicate the data (attributes) they want to share with the web application. After being authenticated, the end user is redirected back to the web application, including the resulting information (i.e. authentication outcome and attributes).

Communication between web applications and OpenID IdPs is based on HTTP GET/POST methods, where HTTP tags are used to represent the information. The use of SSL/TLS is strongly adviced to avoid impersonation attacks.

Unlike SAML, OpenID IdP's are not intended to provide highly reliable and trusted identity information about the end user. There are mere convenient intermediaries that simplify end user's interaction with web applications, avoiding the recurrent introduction of the same information. Therefore, the service provider does not need to have any kind of pre-established trust relationship with the end user's IdP, making its deployment much simpler and scalable than SAML-based federations. Even more, OpenID allows any end user to set up her own IdP. This allows the end user to be in control of her personal data.

WS-*

A *web service* is a special kind of web application intended to support machine-to-machine interaction over the network. A web service exposes a set of operations that can be executed by remote parties, and the interface to these operations is described using the machine-processable WSDL [92] format. The messages to convey requests and their associated responses are encoded using SOAP [93], and transported over HTTP.

The generic term WS-* refers to a collection of standard specifications for web services, part of them released by OASIS. Among other aspects, this collection includes specifications on how authentication, security, trust and federation must be managed by web services. Some of the more relevant specifications are: WS-security [94], an extension to SOAP that aims to provide security to web services, defining how to sign and encrypt SOAP messages, and how to attach security tokens that assert sender's identity (e.g. SAML assertions, Kerberos tickets, X.509 certificates and username/password credentials); WS-Trust [95], which deal with aspects such as key management and trust bootstrapping; WS-Federation [96], which specifies how web services take part on an identity federation; and WS-SecurityPolicy [97], which specifies how to define security policies on these environments.

OAuth

The *OAuth Authorization Framework* [9] is an open standard for authorization that allows end users to share limited access to their protected resources with third-party applications. OAuth is based on the JWT [98] notation, and it is designed to work over HTTP, but other protocols could be possible.

The typical authorization process consists of a set of three request/response exchanges. In the first one, the *Client* (third-party application) requests an *Authorization Grant* from the *Resource Owner* (end user) to access a specific protected resource controlled by a *Resource Server* (service provider). This request can be performed either directly, or

through the *Authorization Server* (identity provider). The second exchange involves the *Client* presenting the requested grant to the *Authorization Server*, in order to obtain a specific *Access Token* for the protected resource. Finally, in the third exchange, the *Client* requests access to the protected resource by presenting the *Access Token*. In this model, the *Authorization Server* and the *Resource Server* can be either collocated or separated in different entities.

The key point in OAuth is to allow Clients to access Resource Owner's data by means of an authentication process carried out between the Resource Owner and the Authorization Server. A typical example scenario is when Alice (Resource Owner) authorizes a Facebook Game (Client) to automatically post her highest score in her Facebook Wall (Resource Server) with her Facebook registered account (Authorization Server).

Though the focus is not the authentication of the end user, the first step of OAuth is similar to the generic federated flow described at the beginning of Section 2.2.3, where the *Client* would be the web application that needs some authorization information (in this case, an Authorization Grant), the *Resource Owner* would be the end user, and the *Authorization Server* would match with the IdP.

OpenID connect

The OAuth standard does not provide a means for the *Client* (web application) to get authentication and authorization information about the *Resource Owner* (end user). *OpenID connect* [99] is a suite of specifications that aims to provide such an identity management layer on the top of OAuth. It provides many of the OpenID features, but with a simpler API.

OpenID connect defines a new type of OAuth token, named *ID Token*, that contains authentication information. Furthermore, this token can also be used later to request additional authorization information from a special end point called *UserInfo*, which manages end user's identity information. Access to the *UserInfo* is performed by following the standard OAuth protocol, where the *Client* uses the *ID Token* as a the authorization grant. Answers provided by the UserInfo end point can include three different types of claims: normal, aggregated (from other sources but provided by the OpenID Provider) and distributed (references are provided in order that the Client can retrieve them).

2.3 Access control to generic applications

Previous sections have described the most relevant technologies used to control the access to the network service and to web applications. However, those technologies are designed to work within their own silos, not being interoperable and not being directly applicable to other kind of applications, such as remote access to terminal (SSH [34]), file transfer (NFS [100]), email (SMTP [101], IMAP [102]), etc.

This section provides an overview of some technologies that enable access control to generic applications, providing them with a unified interface to perform end user authentication and to obtain security services.

2.3.1 Kerberos

Kerberos [7] is a secure three-party protocol for authentication and key management based on shared secret key cryptography. The protocol provides a SSO platform through the so-called *tickets*. A ticket is a piece of encrypted and integrity protected information that allows an end user to be authenticated without requiring her to provide further credentials.

Kerberos messages are exchanged between three types of entities: a *Client* that represents an end user willing to access a specific service, an *Application Server* (AppS) providing the specific application service, and a *Key Distribution Center* (KDC) in charge of authenticating clients and distributing tickets within a specific *realm* (organization). The KDC is integrated by two servers, typically deployed on the same physical entity: the *Authentication Server* (AS) and the *Ticket Granting Server* (TGS). While the former is responsible for authenticating the Client, the latter is in charge of issuing tickets to access Application Servers. Kerberos assumes that both the Client and the AppS have a pre-established trust relationship with the AS and TGS, respectively. In particular, the trust relationship between AS and the Client is defined by a shared secret named *reply key*, which is usually derived from the Client's password.

Figure 2.3 shows a typical Kerberos exchange. The execution starts when the Client, usually at log-on time, requests a *Ticket Granting Ticket* (TGT) from the AS through a *KRB_AS_REQ/REP* exchange *(1)*. The TGT is a special service-independent ticket used for requesting other tickets. The AS generates a *TGS session key* that is included in the TGT to make it available to the TGS. The *KRB_AS_REP* message contains the TGT, other information useful to the user (e.g. ticket lifetime), and a copy of the *TGS session key* protected with the *reply key*. Therefore, only the legitimate Client will be able to decrypt the key. In addition to this, Kerberos implements a feature called *pre-authentication* that

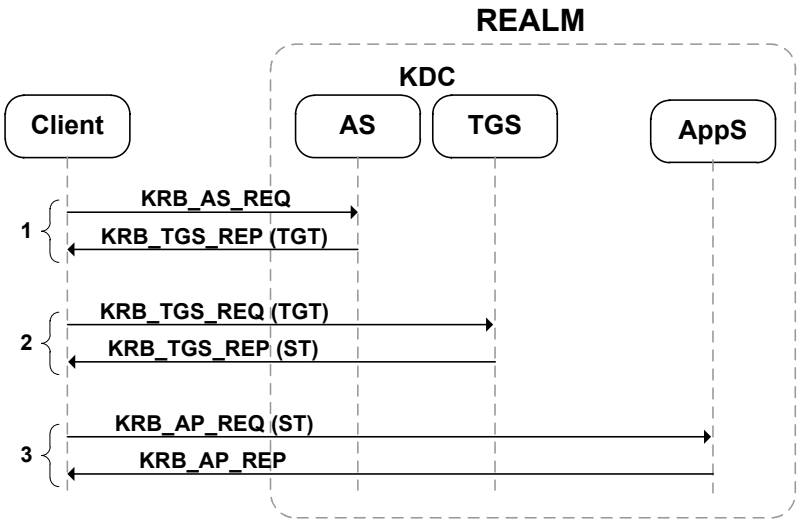


Figure 2.3: Kerberos standard signalling.

allows the KDC to authenticate the Client before providing the TGT. This feature consist in the exchange of pre-authentication data (*padata*) elements between the Client and the KDC. These elements are transported within the *padata* field of the KRB_AS_REQ/REP messages, and within the *e-data* field of the KRB_ERROR message. The specific format and semantic of these *padata* elements depend on the pre-authentication mechanism being used. The IETF has standardized an extensible architecture to facilitate the design of new pre-authentication mechanisms for Kerberos [103].

Once the Client owns the TGT, it can request a *Service Ticket* (ST) from the TGS for accessing the specific AppS *(2)* it desires. With this purpose, the Client sends a *KRB_TGS_REQ* protected with a checksum computed with the *TGS session key*. When the TGS validates the TGT, it generates a *service session key* that is included in both the ST and the *KRB_TGS_REP* message. Finally, similarly to the TGS exchange, the Client authenticates itself to the service *(3)* by sending the ST to the application server in a *KRB_AP_REQ* message. Optionally, a *KRB_AP_REP* message can be used if the Client needs to authenticate the AppS. The acquired ST is valid for further accesses to the same service (within a validity period). If the Client desires to access a different ApPS, a new ST needs to be issued by means of the same TGT obtained at log-on time. Hence, the end user is able to access any AppS during the session period without further re-authentication processes. A more detailed description of this flow can be found in [7].

In addition to this, Kerberos supports a federated operation mode called *cross-realm*. Founded on the establishment of trust relationships between TGS/KDCs of different realms, the Client follows the path from the home to the visited realm by obtaining the

so-called *cross-realm TGTs*. The process finalizes when the Client contacts the KDC in the visited realm and obtains a valid ST for the target service. Figure 2.4 depicts this process, where the Client is originally registered in *REALM1's* KDC. As observed, the cross-realm solution requires all the intermediate organizations to deploy a Kerberos infrastructure, even when they are not interested in providing Kerberos-based services themselves.

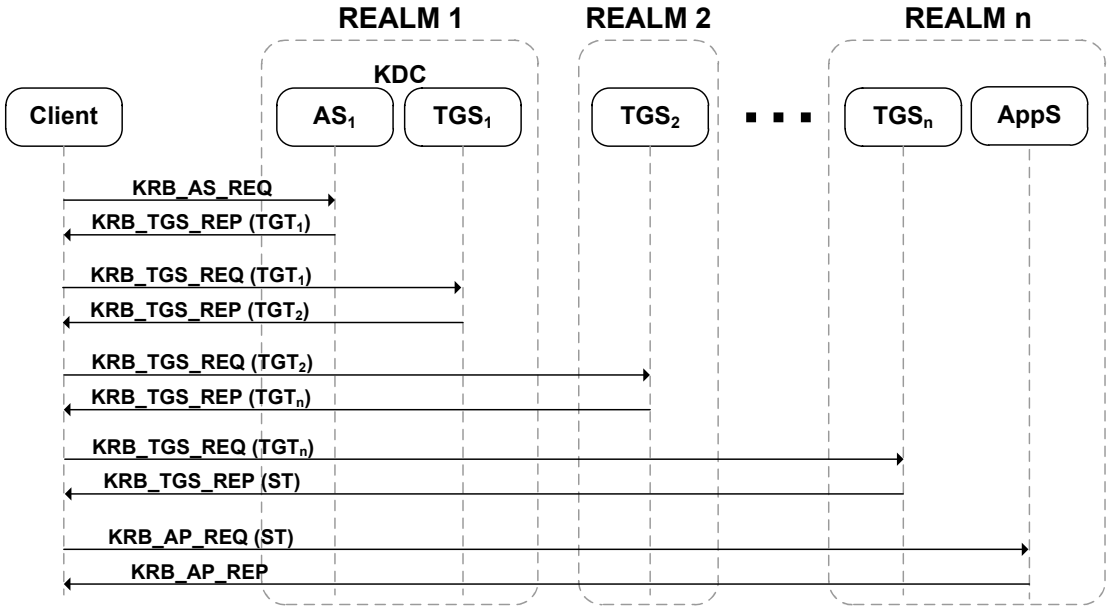


Figure 2.4: Kerberos cross-realm signalling.

2.3.2 GSS-API

The *Generic Security Service Application Program Interface* (GSS-API) [40] is a generic framework defined within the KITTEN WG [104] that provides application with security services such as authentication, integrity and confidentiality. Distributed application services that need to protect their communications can employ the different security services offered by the GSS-API and remain independent from any particular security mechanism. Nowadays a large number of Internet protocols, such as SSH [34], NFS [100], and several non-IETF applications, such as Microsoft SQL server [105], support GSS-API as a means for obtaining security services. The most extended GSS-API mechanism is based on the Kerberos protocol [106], although other mechanisms, such as the one based on EAP [107], have also been proposed.

GSS-API allows a *GSS-API Initiator*, usually the end user, to establish a *security context* with a *GSS-API Acceptor*, usually the application service. The negotiation of

the context starts when the initiator invokes the *GSS_Init_sec_context()* function, which returns a token to be passed to the acceptor (using the application protocol), and indicates a pending status (GSS_S_CONTINUE_NEEDED) to complete the context establishment. The acceptor receives the token and passes it to the *GSS_Accept_sec_context()* function. Assuming a single round-trip authentication mechanism, the function indicates a GSS_S_COMPLETE status and returns a token to be sent back to the initiator. Finally, the initiator invokes again the *GSS_Init_sec_context()* function (passing the received token), which also returns a GSS_S_COMPLETE status, indicating the successful establishment of the security context. If the authentication mechanism requires more than one round-trip to be completed, the acceptor and the initiator obtain a GSS_S_CONTINUE_NEEDED status, and the process is repeated until GSS_S_COMPLETE status is obtained.

Once the security context has been established, the GSS-API offers two different mechanisms to protect application messages sent by either, the initiator or the acceptor: on the one hand, with the *GSS_GetMIC*/*GSS_VerifyMIC* calls, messages are authenticated and integrity protected; on the other, by using the *GSS_Wrap*/*GSS_Unwrap* functions, confidentiality is also provided. The cryptographic material used to protect these messages is derived from the keying material resulting from the underlining mechanism (e.g. Kerberos).

2.3.3 SASL

The *Simple Authentication and Security Layer* (SASL) [108] is a framework defined within the KITTEN WG, intended to provide authentication and data security services in connection-oriented protocols via replaceable mechanisms. The framework incorporates interfaces with both protocols and mechanisms, in such a way that new mechanisms can be used with old protocols, and new protocols can reuse existing mechanisms, as long as these protocols include support for SASL. In this sense, a large number of protocols, such as IMAP, SMTP, XMPP [109], and LDAP [110] support the SASL framework. Regarding SASL mechanisms, there are a number of them based on different authentication technologies, such as DIGEST-MD5 [111] or CRAM-MD5 [112], though the GS2 Mechanism Family [108] is specially relevant for the purpose of this thesis, since it allows the use of GSS-API mechanisms within SASL, hence extending the applicability of GSS-API mechanisms also to those protocols supporting the SASL framework.

SASL defines two main participants: the *Client* and the *Server*. The Client is the one trying to get authenticated to access a service (i.e. the end user), while the Server is the entity that provides the service and, hence, the one that wants to verify Client's

identity. This authentication process is called *Authentication Exchange*, and its outcome is the establishment of a security layer between them, which provides security services such as confidentiality and integrity protection.

A typical SASL *Authentication Exchange* starts when the Client requests authentication via a specific mechanism. After that, a sequence of one or more pairs of server-challenges and client-responses are exchanged. The purpose of these exchanges may include the authentication of the Client to the Server, the authentication of the Server to the Client, the transmission of authorization information or the negotiation of a security layer. At the conclusion of the Authentication Exchange, the Server sends a message indicating the outcome of the exchange to the Client.

2.4 Solutions for AAA-based federated authentication and authorization

This section describes three of the most relevant solutions that aim to provide a federated authentication and authorization based on AAA infrastructures. These solutions are based on the background technologies described in the previous subsections of this chapter. In particular, Section 2.4.1 introduces *eduroam*, a network access federation for the research and education community. Section 2.4.2 describes the DAMe project, which builds upon the eduroam infrastructure to add federated authorization capabilities to the network. Finally, Section 2.4.3 describes the Moonshot project, a proposal that aims to provide an access control interface to applications for a federated authentication and authorization, based on AAA infrastructures and GSS-API.

2.4.1 eduroam

eduroam (*Educational Roaming*) [24] is a proposal of the *Trans-European Research and Education Networking Association* (TERENA) [113], now funded by the GÉANT GN3 project. It provides a secure and inter-institutional roaming network access federation for the international research and education community. Its purpose is to allow staff belonging to participating institutions[4] to obtain Wi-Fi Internet connectivity when visiting other institutions. Authentication is actually carried out at the end user's home institution using the institution's specific authentication method and credentials. It is present in hundreds of institutions of different countries around the world [114]

[4]In eduroam, the term *institution* is used to denote an organization.

The federation substrate in *eduroam* is based on an AAA architecture interconnecting the participating institutions. In particular, *eduroam* is built upon a hierarchical infrastructure of RADIUS servers, where the root is managed by TERENA, and a second level is provided by the different *National Research and Educational Network* (NREN) of each participant country. Each institution willing to participate in *eduroam* needs to connect a RADIUS server to the one located on its corresponding NREN, and be assigned with a sub-realm. Moreover, it is possible to define further levels in this hierarchy, in the cases where an institution decides to include an additional partition of its users. For example, the realm *diic.um.es* indicates the sub-realm *diic* within the organization *University of Murcia (um)*, connected to *RedIris*, the Spanish NREN (*es*). Figure 2.5 illustrates a simplified version of this hierarchy.

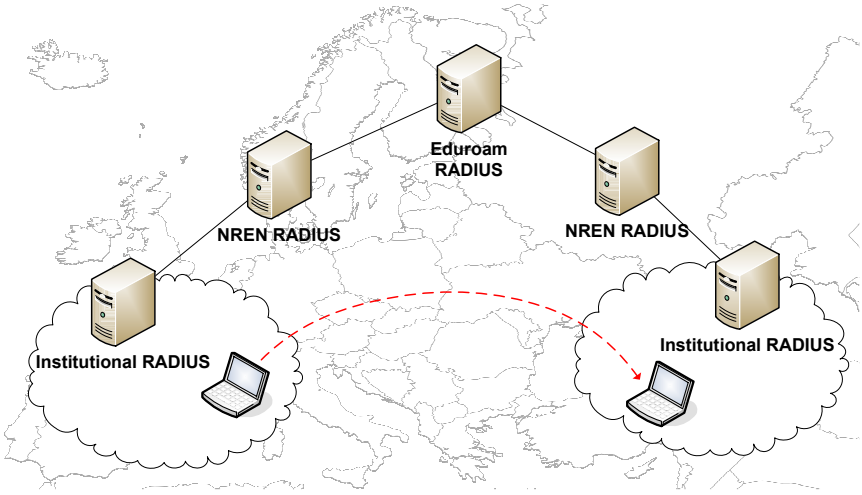


Figure 2.5: *eduroam* infrastructure.

eduroam optionally supports the use of dynamic RADIUS routing, where institutions can announce the location of their RADIUS servers over DNS, in such a way that other institutions can look up and contact them directly without the need of any intermediate RADIUS infrastructure. However, this routing model is only supported over RADIUS/TLS (RadSec).

User authentication is carried out by the EAP protocol. Each organization is free to set up the EAP method of its choice, as long as this method provides effective protection against eavesdropping for critical user data (such as passwords). In particular, the most commonly used authentication mechanisms are tunnelled EAP methods such as EAP-TTLS [60] or PEAP [61], although EAP-TLS [59] can also be used. EAP packets are transported from the EU to the visited institution's access point (AP) using the

802.11 technology, while the RADIUS infrastructure is responsible to deliver them to the corresponding home institution.

2.4.2 DAMe

Since the current *eduroam* infrastructure only supports end user authentication and very basic authorization procedures, the *DAMe* (*Deploying Authorization Mechanisms for federated services in eduroam architecture*) [115] project was initiated with the intention of improving the federated network access scenario of eduroam. In particular, it specifies a mechanism to support a more fine-grained authorization level, based on an *authentication token* (*eduToken*) obtained from the home institution (IdP) during the network access authentication. This *eduToken*, described in [115], can be used by the visited institution (service provider) to request additional end user's attributes to take an authorization decision. DAMe extensions to *eduroam* mainly rely on the SAML and XACML standards to represent statements and policy information, respectively.

Moreover, the *eduToken* was also proposed as a possible enabler to perform authentication and authorization for web applications within the visited institution, although this approach was not much further investigated nor standardized.

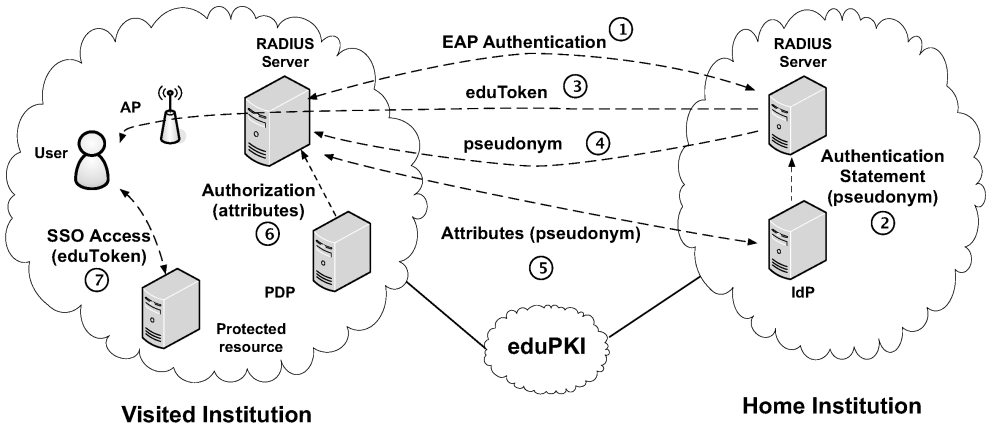


Figure 2.6: DAMe architecture overview.

Figure 2.6 describes the general DAMe architecture. As it can be observed, an end user from a *Home Institution* (HI) is requesting network access through an access point (AP) located in a *Visited Institution* (VI). Initially, according to the basic *eduroam* functionality, the end user engages in an EAP authentication *(1)* with the HI through the VI, where EAP messages are transported between both institutions by using the RADIUS infrastructure

deployed in *eduroam*. Once the end user is successfully authenticated, the home RADIUS server contacts the *Identity Provider* (IdP) in order to get a SAML authentication statement *(2)*. Among other information, this statement contains a transient pseudonym which will serve to refer the authenticated end user in future communications with the IdP when, for example, retrieving her associated attributes. This statement is called *eduToken*, and its specific format and characteristics are described in [115]. The *eduToken* is directly delivered to the end user *(3)* through a secure communication channel, typically implemented by a TLS channel established during the execution of tunnelled EAP methods, such as PEAP or EAP-TTLS. These methods usually require of a Public Key Infrastructure (PKI) to establish trust between the end user and the *EAP server*, as the end user needs to verify the HI's X.509 certificate. In the *eduroam* environment, this is achieved by means of *eduPKI* [116]. Finally, the pseudonym is sent to the visited RADIUS server *(4)* within the final *Access-Accept* RADIUS message.

On the one hand, the pseudonym is used for performing the authorization of the end user during the network access. More precisely, the received pseudonym is used by the visited RADIUS server for obtaining relevant attributes *(5)* from the IdP, in order to decide whether the end user is granted access to the network. The obtained attributes are provided to the *Policy Decision Point* (PDP), located in the visited institution, which checks the local policies and takes a decision *(6)*. On the other hand, the *eduToken* was also intended to be used for implementing a cross-layer SSO mechanism to access web resources available within the federation *(7)*, although as mentioned before, this functionality was not much further investigated nor standardized.

2.4.3 Moonshot/ABFAB

The Moonshot project [37] is a proposal that aims to specify a federated authentication and authorization architecture that enables access control to most application services. The main contribution of this project, and cornerstone of its architecture, is the *GSS-API Mechanism for the Extensible Authentication Protocol* (GSS-EAP) mechanism [107]. This allows providing application services with a common access control layer (GSS-API), while the authentication is based on EAP and the AAA infrastructure, which have intrinsic capabilities for a federated operation. Moreover, Moonshot's architecture also describes how additional identity information can be retrieved from the end user's IdP, and provided to the application service in order to allow further authorization.

Moonshot appears under the umbrella of Janet and the TERENA EMC2 task-force. As an outcome of this project, a new working group has been formed in the *Internet*

Engineering Task Force (IETF) standardization organism, with the purpose of developing and standardizing the technologies required for implementing the identity federation architecture designed in Moonshot. This working group is called *Application Bridging for Federated Access Beyond Web* (ABFAB). The proposed architecture is based on the use of commonly used security mechanisms, though some extensions to them are required to fulfil the desired functionality. Namely, the ABFAB architecture is mainly based on EAP, RADIUS, GSS-API, SASL and SAML specifications.

The main elements that take part in the ABFAB architecture are the *Client* (i.e. end user), the *Relaying Party (RP)* (i.e. application service) and the *Identity Provider (IdP)*. Between them, three interfaces are defined to convey the required interactions to provide the federated access control. Figure 2.7 depicts these entities and relationships.

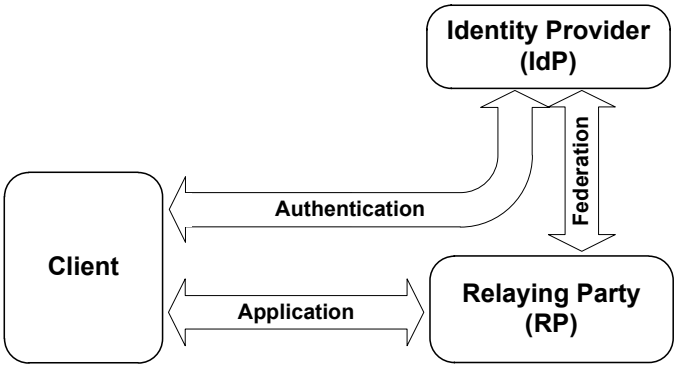


Figure 2.7: Moonshot's architecture.

The *authentication* interface is intended to provide mutual authentication between the Client and the IdP. This interface is implemented using EAP, where the RP acts as the EAP authenticator (pass-though mode), serving as intermediary of the conversation. It also provides the means of confirming RP identity through the EAP channel binding feature [117], which allows the EAP peer (Client) and EAP server (IdP) to verify they are dealing with the same EAP authenticator (RP). This feature prevents man-in-the-middle attacks.

The *federation* interface provides the federation substrate, that is, the means for the establishment of the trust relationship between the RP and the IdP. It also provides a channel allowing the RP to obtain Client's identity information from the IdP, and to convey authentication packets from the Client to the IdP and vice-versa. It is implemented using AAA protocols (e.g. RADIUS). Conversely, SAML is used to represent the authorization information provided by the IdP (e.g. authentication/attribute statements).

The *application* interface provides the following main functionalities: a) executes the application protocol requested by the Client, b) performs mutual authentication between Client and RP, and c) provides the security services required by the application protocol after authentication (e.g. confidentiality or integrity protection). While the first functionality is out of the scope of ABFAB, the other two are accomplished by the use of the GSS-API. The Client acts as the GSS Initiator, while the RP acts as the GSS Acceptor.

As the Client authentication with the IdP is performed by means of EAP, the GSS mechanism used between the Client and the RP must act as an EAP lower layer, conveying the transport of EAP packets from the Client to the RP and vice-versa. Since there were not GSS-API mechanism providing that functionality, ABFAB defined the GSS-EAP mechanism. The GSS-EAP specification describes how EAP can be used to establish a GSS context between the Client and the RP, and how the keying material exported by the EAP method is used to provide confidentiality and integrity protection to the application protocol after authentication. The use of the GSS-EAP mechanism is limited to those EAP methods that satisfy a number of requirements. Namely, they must provide dictionary attack resistance (e.g. tunnelled methods within TLS), they must support key derivation and mutual authentication, and they must provide channel binding. Example of suitable EAP methods are EAP-TTLS or PEAP.

Moreover, the GSS-EAP mechanism allows the RP to obtain authorization attributes from the IdP. In particular, this information is presented to the RP through the GSS-API name attributes feature [118], associated to the Client's name. Specifically, ABFAB has defined a set of name attributes for the GSS-EAP mechanism [119], intended to represent RADIUS attributes and SAML Attributes and Assertions.

The procedure that a Client must perform to gain access to a service provided by a RP is depicted in Figure 2.8. In the first place, a GSS-EAP authentication starts over the application protocol. The Client provides her identifier (in the form of a NAI [120]) to the RP within an EAP response packet, transported within a GSS-API token (1). Based on the *realm* part of the NAI, the RP determines the IdP that will handle the authentication and forwards the EAP response using the AAA infrastructure (2). The RP also includes its identity in the message. The IdP starts a new EAP method with the Client. The EAP packets generated by the execution of the method are transported back and forth to the Client using the RP as EAP pass-through authenticator (3). As part of the EAP protocol, the Client sends channel bindings in order to verify the identity of the RP. After the authentication process, Client and IdP have mutually authenticated and derived the MSK and EMSK keys (4). The IdP checks its policies to determine whether the Client and the RP are authorized to perform the requested action, and which authorization information

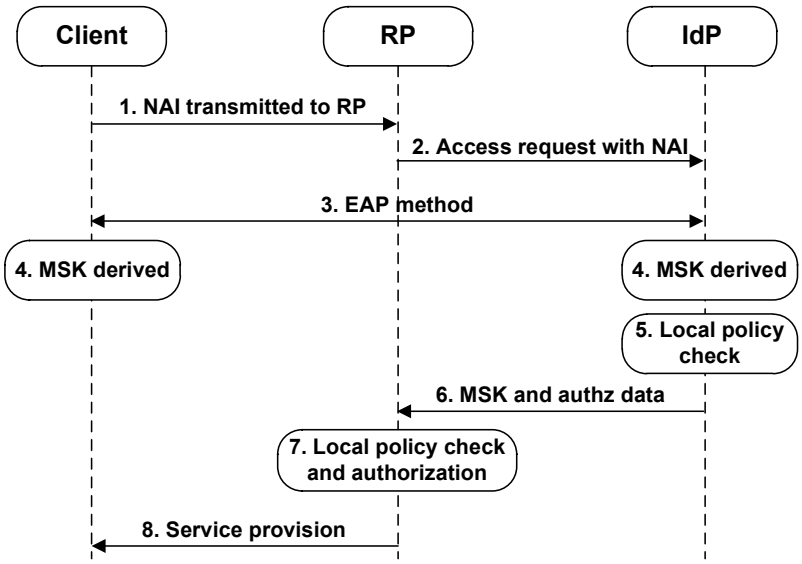


Figure 2.8: ABFAB operation.

will be provided to the RP (5). The IdP provides the RP with the MSK and optionally with a collection of AAA and SAML attributes (6). The RP processes the received information against its local policies to take an authorization decision (7). The RP provides the service to the Client (8). The MSK may be used to protect the application protocol messages.

The ABFAB architecture assumes updates to the existing entities in order to accomplish the required functionality. Namely, it requires the modification of the application services (RPs), as they will need to include support for the GSS-EAP mechanism and to be able to understand the newly defined RADIUS and SAML GSS naming attributes for authorization. The Client also needs to be updated to support the GSS-EAP mechanism. Finally, the IdP needs to be updated to generate (or retrieve from a third party) the required SAML assertions and attributes, and to support their transport on top of the AAA protocol.

2.5 Conclusions

This chapter has described the access control technologies with the highest relevance for this thesis, detailing how they work, the kind of services they intend to protect, and how they operate in federated environments. Moreover, this chapter also introduced some state-of-the-art proposals that, grounded on these access control technologies, aim to provide some level of federated authentication and authorization to services based on the

AAA infrastructures. Namely, these proposals are *eduroam*, which provides a world-wide production environment for federated access to the network service; *DAMe* which extends *eduroam* architecture to provide finer-grained authorization capabilities, and which also provides a preliminary proposal for a cross-layer SSO solution to access web applications; and *Moonshot/ABFAB*, which provides an access control interface for application services that includes federated authentication and authorization using GSS-API and EAP.

Although these state-of-the-art solutions are completely valid for the purposes they were defined to, none of them completely addresses the objectives described for this thesis in Section 1.3. Hence, this document defines new proposals and architectures that, using these technologies, provide a better solution to the stated problem. The following chapters of this dissertation will provide details of the of these new proposals, and will analyse further research works directly related with each one of them.

Chapter 3

FedKERB: Integrating Kerberos with AAA and advanced authorization infrastructures

3.1 Introduction

As described in Chapter 1 and 2, nowadays Kerberos is becoming one of the most widely deployed standards for authentication and key distribution in application services [121]. Indeed, most operating systems, and different network applications such as SSH, NFS, or XMPP, already support, or even require Kerberos to perform the access control process. One of its stronger points is its SSO capability. However, whereas many service providers use this protocol to control the access of their own subscribers, they do not usually deploy Kerberos-based federations (*cross-realm*) to handle subscribers coming from other organizations.

We have also seen that *Authentication*, *Authorization* and *Accounting* (AAA) infrastructures have gained popularity for the access control to the network service. In this context, the *Extensible Authentication Protocol* (EAP) has become the most used authentication protocol, as it provides flexible authentication and easy integration with underlying AAA infrastructures. Due to its success, organizations have formed federations (AAA-based federations), to allow subscribers to access the network from other's organizations. One example is *eduroam*, described in Section 2.4.1.

The reasons behind not using Kerberos *cross-realm* are mainly grounded on some recognized issues in the deployment of Kerberos cross-realm. But more importantly, establishing a *cross-realm* infrastructure would imply the deployment and

interconnection of the so-called Key Distribution Center (KDCs), in charge of distributing Kerberos credentials, in all organizations comprising the federation. As mentioned in Chapter 1.2, maintaining different federation infrastructures requires a significantly higher administrative effort. Moreover, Kerberos lacks of a complete authorization management that allows performing fine grain access control in federated environments.

In this scenario, it would be extremely beneficial for an organization to have a single federation infrastructure for all the offered services, regardless they are the network service itself, or any application service deployed on the application layer. Starting from this motivation, this chapter focuses on defining an architecture that provides a viable solution to this problem. In particular, this chapter tries to address the following objectives, extracted from Section 1.3:

O1 To design a solution allowing the bootstrapping of Kerberos credentials on the service provider by enabling the Kerberos infrastructure to make a direct use of the AAA infrastructure to perform a federated authentication.

O4 To design an authorization model enabling the Kerberos infrastructure on the service provider to be fed with end user identity information coming from the home organization after authentication, and to use it to make fine-grained access control decisions.

These objectives can be broken down into the following sub-objectives:

- To design a general architecture for the proposed scenario based on Kerberos and AAA infrastructures.
- To define the integration between AAA and Kerberos infrastructures to provide a federated authentication and authorization process which also provides support for SSO.
- To define how to integrate fine-grained authorization management in the proposed solution.
- To define a generic and extensible architecture useful in different scenarios and contexts.
- To avoid the modification of the existing standards.

The integration of Kerberos and AAA infrastructures has motivated an incipient effort in standardization bodies. One of the most earlier works in this area can be found in [122], where authors propose that the KDC located in the service provider distributes a TGT to the end user (client) after a successful EAP authentication for the network service. However, this solution does not follow the Kerberos specification. It assumes an initial network access process that replaces the *KRB_AS_REQ/KRB_AS_REP* exchange. As a result of the network access authentication, the client is provided with a TGT. This contradicts Kerberos specification, where it is stated that the TGT is provided to the client as a result of a *KRB_AS_REQ/KRB_AS_REP* exchange.

Some other works have focused on the definition of a pre-authentication framework that favours the integration of different authentication methods for Kerberos. Following this approach, authors in [123] propose to include an initial phase to the Kerberos protocol where the client authenticates to a new entity called *pre-authentication server*, in order to obtain an *authentication ticket*. This ticket must then be included in the pre-authentication data field of the *KRB_AS_REQ* message sent to the KDC, where it is validated. However, this proposal requires to extend the Kerberos protocol in order to support a new type of exchange, and the deployment of new entities (pre-authentication servers).

Another solution with the same objective that avoids these drawbacks can be found in [103]. This document specifies a generic standard framework to guide the definition of pre-authentication mechanisms. This framework also includes the definition of a common set of generic functions that may be used by these mechanisms, such as a protected channel established between the end user and the KDC, that can be used to exchange pre-authentication data. That is, designers of pre-authentication mechanisms can rely on the facilities provided by this secure channel, such as binding of request and response messages, confidentiality or freshness. However, none of the pre-authentication mechanisms defined so far provide integration with AAA infrastructures. This chapter will define such a pre-authentication mechanism as part of the proposed solution.

Regarding authorization, Kerberos defines the *authorization-data* field in TGTs and STs. This field provides a transparent transport of authorization data from the AS to the TGS, and from the TGS to the application service. It is composed of one or more *authorization-data-elements* (ADE), each one consisting on an identifier of the type of the transported information, followed by the specific representation of that authorization information. However, there is no consolidated standard defining how this authorization information must be encoded or processed. Microsoft defined the *Privilege Attribute Certificate* (PAC) [124], as its proprietary form of representing user privilege information for the Active Directory [125]. Conversely, the Kerberos WG [126] started defining another

structure called *Principal Authorization Data* (PAD) [127], with a similar purpose but different format. However, both structures are intended to be used in Kerberos-only infrastructures, where the home authorization information is generated by a KDC in the home organization.

In order to be used in a wider range of scenarios and deployments, a more general purpose representation would be desirable. In this sense, one of the most relevant and widely deployed technologies is SAML. It has gained popularity between organizations thanks to solutions such as Shibboleth [87] or OASIS Web Services Security (WSS) [94]. As described in Chapter 2, SAML is mainly used in web environments, but nothing precludes its use in other types of applications, as it does not impose any specific binding. The integration of SAML and Kerberos is still an open issue [128], and it is under study in different standardization bodies like IETF and OASIS. Two models of integration have been outlined: Kerberos-in-SAML and SAML-in-Kerberos. Regarding the former, [129] describes how to define a new SAML *SubjectConfirmation* method for the *AuthnStatement* element when the end user has been authenticated by means of Kerberos, and [130] describes how to encapsulate Kerberos messages (such as *KRB_AP_REQ*) inside *AttributeStatement* elements. Finally, [131] proposes a new version of the SAMLv2 Web SSO profile where the end user authentication is performed through Kerberos credentials. However, those proposals are typically oriented to web services. Regarding SAML-in-Kerberos, to the best of author's knowledge, no solution has been provided yet.

There are other proposals of integrating AAA infrastructures in the access control to applications that do not make use of Kerberos. The most relevant one is the Moonshot project [37], described in chapter 2.4.3. While Moonshot provides a valid solution for the federated access control to application services based on the AAA infrastructure, it requires to modify these servers to support both, the GSS-EAP mechanism, and the processing of SAML assertions. Moreover, unlike Kerberos, Moonshot does not provide a full SSO solution, requiring to perform a complete EAP authentication for every different application service being accessed.

As existing proposals in the literature do not provide a full coverage of the objectives enumerated above, this chapter proposes an unified architecture, called FedKERB, aiming to fill the gaps. Indeed, this architecture integrates AAA-based federations (such as eduroam) with the access control to application services carried out by Kerberos. Specifically, this chapter provides a design of a novel Kerberos pre-authentication mechanism, based on EAP, which allows end users from any organization in the federation to leverage the AAA infrastructure to perform authentication with the service provider's KDC. As a consequence, this proposal removes the need of deploying a parallel Kerberos

cross-realm infrastructure in the federation. Moreover, although Kerberos does not specify how authorization is performed, it does provide some elements to transport authorization information as opaque data. This chapter uses these elements to incorporate fine-grained authorization into the defined architecture, by means of the integration with the well-known SAML standard.

This chapter is structured as follows. Section 3.2 presents the proposed architecture. Section 3.3 discusses the two possible alternatives to implement EAP-based pre-authentication. Section 3.4 details how a federated end user obtains a Service Ticket for an application service in the service provider's domain. This section is divided into two parts: subSection 3.4.1 describes how to perform the KRB_AS_REQ/REP exchange, including EAP-based pre-authentication; while subSection 3.4.2 describes how the TGS_REQ exchange is carried out, including fine-grained authorization. Section 3.5 discusses some considerations that must be taken into account for the deployment of the proposed solution. Section 3.6 provides a security analysis of the proposal. Finally, Section 3.7 presents some conclusions and summarizes the standardization actions that have resulted from the work on this chapter.

3.2 Proposed architecture

In order to define the architecture, let us assume two different organizations (service provider and home organization), belonging to the same AAA-based federation. In this scenario, an end user, subscriber of the home organization, wants to gain access to an application service deployed in the service provider.

The architecture has the following requirements:

- As members of the federation, both organizations deploy AAA servers that allow them to exchange authentication, authorization and accounting information.
- The service provider deploys a Kerberos KDC to distribute service tickets to access the application services deployed on the organization.
- The application service uses Kerberos for performing access control.
- The end user makes use of EAP to perform authentication with the home organization's AAA server, using the KDC as EAP authenticator.
- The home organization deploys a SAML IdP to distribute authorization information (e.g. authentication and attribute statements) to the KDC.

- The service provider deploys a *Policy Decision Point* (PDP), in order to manage access control policies, and to assist the KDC to take authorization decisions.

- The federation may deploy a *Metadata Service* (MDS), to make information about the services available within the federation (e.g. URL and public keys of application services, IdPs or KDCs) available to the members.

Although the text and examples in this chapter assume that the service provider and the home organization are different, nothing precludes the use of the mechanism described in this chapter in scenarios where they are, in fact, the same organization.

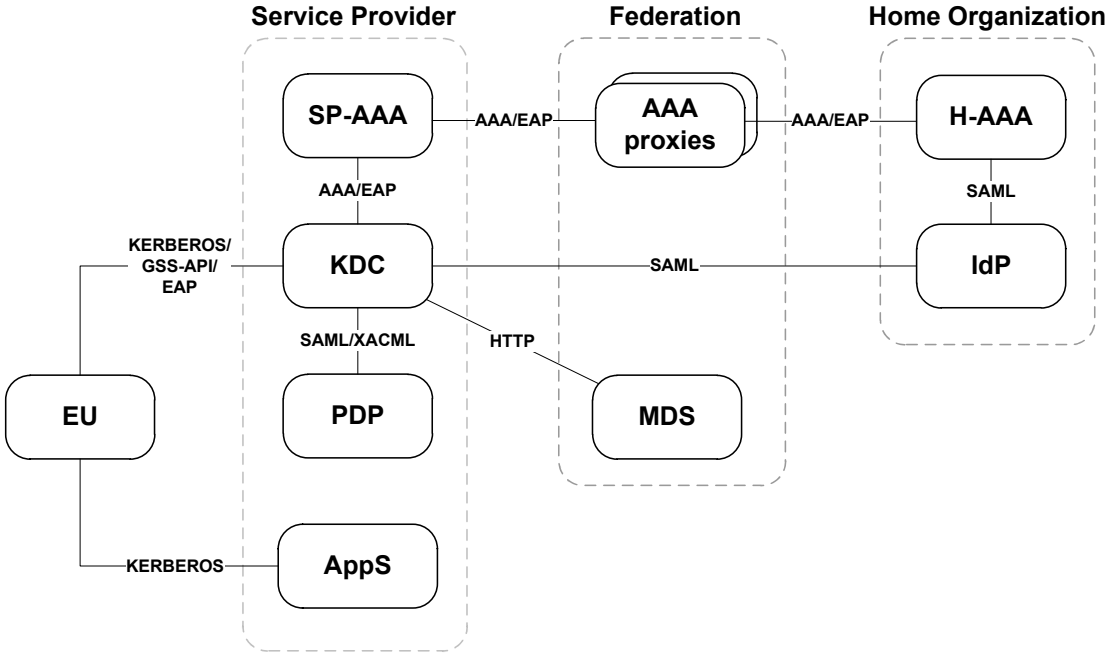


Figure 3.1: FedKERB architecture.

Based on the requirements described above, the resulting architecture is composed by the following components, depicted in Figure 3.1:

- *End User (EU).* This component is interested in accessing to a particular application service that uses Kerberos as the access control mechanism. To obtain the required Service Ticket (ST), it first interacts with the service provider's KDC. It uses EAP as the pre-authentication mechanism, where EAP packets are exchanged with the home AAA server using the KDC as intermediary (EAP authenticator).

- *Application Service (AppS).* It is the component providing the specific application service the EU wants to access to (e.g. SSH, FTP, NFS, SMTP, etc.). Access control to the AppS is based on the use of Kerberos STs. If the EU presents a valid ST, she is assumed to be authenticated and authorized to access the service.

- *Key Distribution Center (KDC).* The KDC is the main component of the proposed architecture. It is deployed by the service provider to handle access control to its application services. During Kerberos pre-authentication, the KDC plays the role of an EAP authenticator, forwarding EAP request/response packets between the EU and the home AAA server. Moreover, the KDC may perform an authorization process before providing the end user with the requested ST. For that, it acts as a *Policy Enforcement Point* (PEP), contacting the IdP to retrieve EU's identity information, and querying the *Policy Decision Point* (PDP) to obtain an authorization decision.

- *Service provider's AAA server (SP-AAA).* This is the component that interconnects the service provider to the AAA federation. It is contacted by the KDC to transport the EAP packets from the EU to the home AAA server, and vice-versa. The transport is made through the AAA proxies that integrate the federation.

- *Home organization's AAA server (H-AAA).* It is the component responsible for authenticating the end user, acting as EAP server. Besides, it also contacts the IdP to obtain identity information about the end user and send it to the KDC.

- *Identity Provider (IdP).* To be compliant with current widely deployed technologies based on SAML such as Shibboleth or Liberty Alliance [132], the proposed architecture assumes the home organization is running an IdP. The IdP receives SAML authentication requests from the H-AAA, and may also receive attribute queries from the KDC. This component may be collocated with the H-AAA.

- *Policy Decision Point (PDP).* It is the component that manages the set of access control policies in the service provider. It may receive authorization decision queries from the KDC, which acts as *Policy Enforcement Point* (PEP). These queries are matched against existing polices to take an authorization decision.

- *MetaData Service (MDS).* When authorization processes are carried out in federated scenarios, the MDS is a special service (Section 2.2.3 where organizations may publish information (e.g URL, public key...) about the services willing to be provided within the federation (e.g. application services, IdPs, KDC...). In particular, the IdP

location is published in the MDS. Usually, MDS location is pre-configured inside federation members and it is managed at a federation level. The deployment of this entity is optional.

The architecture depicted in Figure 3.1 describes the interfaces required to successfully interconnect the KDC with the AAA server and the IdP in the home organization. Through these interfaces, the KDC is able to obtain enough information about the end user to take a complete access control decision. In particular, a Kerberos pre-authentication process between the end user and the KDC is carried out using EAP as the authentication protocol. In that process, the KDC acts as the EAP authenticator, while the end user and the home AAA server act as the EAP peer and the EAP server respectively. Therefore, since the KDC is directly connected to the AAA-based federation, there is no need to deploy a Kerberos cross-realm infrastructure in the federation. This is an important deployment advantage, as it will be analysed in Section 3.5.2.

More specifically, by means of a *single* EAP authentication process, it is possible to bootstrap the required security association between the end user and the KDC to obtain the TGT. With it, the end user can request additional service tickets (STs) to access application services within the service provider without performing any further EAP authentication process (within a specific session lifetime). The distribution and use of these tickets are the core of the SSO operation provided by Kerberos. Thus, the end user does not need to maintain several credentials for different application services, but only a valid credential to perform a successful EAP authentication with its home organization through the AAA infrastructure.

In addition to this federated authentication process, the architecture described in this chapter is conceived to allow service providers to manage end user attributes, and to take access control decisions by interfacing with existing authorization infrastructures. In those cases, all the authorization management responsibilities lie on the KDC, being completely transparent to application services that remain unmodified.

3.3 EAP-based pre-authentication

In order to address the goal of integrating Kerberos pre-authentication with EAP authentication, this thesis has analysed two different models:

- Kerberos/EAP model: EAP packets are transported directly on the pre-authentication field of the Kerberos messages exchanged between the EU and the KDC.

- Kerneros/GSS-API/EAP model: EAP packets are encapsulated in GSS-API tokens, and then transported on the pre-authentication field of the Kerberos messages exchanged between the EU and the KDC.

A thorough comparative of these two models can be found in one of our published papers [133], where they have been analysed and described in high detail. The remaining of this subsection summarizes part of this comparative, and provides the rationale behind the selection of the preferred model for this chapter's architecture.

In terms of functionality both solutions are equivalent. The main differences between them appear in the internal interfaces. The main advantage of the Kerberos/EAP model is its simplicity, since Kerberos is able to directly interface with the EAP stack, acting as EAP lower-layer, without the need of any additional level of abstraction. Moreover, the interface between EAP and any EAP lower-layer is already defined in [134], which really eases implementation tasks. With respect to Kerberos/GSS-API/EAP model, implementation can be more complex since it must be carried out in two levels: first, the use of GSS-API for pre-authentication must be introduced in a Kerberos implementation; second, the integration of EAP in GSS-API also needs to be implemented.

However, the main advantage of the second model is that by using it, Kerberos can potentially use any available GSS-API authentication mechanism, and it is therefore not restricted to just EAP. In this manner, other authentication and key management protocols could potentially be used without changing the Kerberos elements.

Furthermore, the ABFAB WG has already completed the definition of the GSS-EAP mechanism (as described in Section 2.4.3, while Moonshot maintains an updated an robust open source implementation of that mechanism. These two conditions greatly simplify the specification and implementation of the Kerberos/GSS-API/EAP model, which now consists of the definition of a Kerberos pre-authentication mechanism based on the GSS-API, and then use the GSS-EAP mechanism. Thus, the specification of such a pre-authentication mechanism is sensibly simpler than the definition of one directly based on EAP, as this way the KDC is abstracted from the complexities of implementing an EAP authenticator (e.g. managing sequence numbers, handling channel bindings or securely deriving cryptographic material).

Therefore, based on these conditions, the second model has been the one selected to be used for this proposal, due to its:

- Easier implementation
- Abstract the KDC from the EAP details.

- Extensibility to support other GSS-based mechanism for pre-authentication.

This second model, defined in this thesis, has been described in great detail in an IETF draft [135], and proposed for standardization within the Kerberos WG. Furthermore, during the stay associated to this thesis, it has also been implemented as a MIT Kerberos [136] pre-authentication plug-in, and made publicly available as open-source code [137].

3.4 General operation

Once the architecture has been presented, this section explains in detail how its components interact to achieve the goal of federated authentication and authorization. We can distinguish two main phases.

1. *TGT acquisition (KRB_AS_REQ/REP exchange).* The EU performs Kerberos pre-authentication with the KDC (AS server), based on the GSS-API and the GSS-EAP mechanism. As a result, she obtains a TGT which also contains some SAML authorization information coming from her home organization's IdP.

2. *ST acquisition (KRB_TGS_REQ/REP exchange).* The end user asks the KDC (TGS server) for a ST to access a specific AppS deployed in the service provider. She includes the TGT in the request. The KDC analyses the authorization information within the TGT, (optionally) queries the IdP to obtain further end user's identity information, and (optionally) queries the PDP to obtain an authorization decision, based on a pre-established set of policies. If the decision is to grant the access to the AppS, the EU is provided with the requested ST.

With the ST, the EU can eventually access the AppS by means of a *KRB_AP_REQ/REP* exchange. Usually, this exchange is carried out by means of the GSS-KRB mechanism [106], widely used nowadays. If the EU presents a valid ST to the AppS, she is assumed to be correctly authenticated and authorized to access the service. Therefore, the AppS does not need to be modified as the federated access control is entirely performed by the KDC.

Following subsections provide in-detail description of these two steps. Due to the quantity of protocols we have used in this proposal, and for sake of clarity, this description will provide details only for those fields/messages/attributes that represent a change when compared to their respective standard usage.

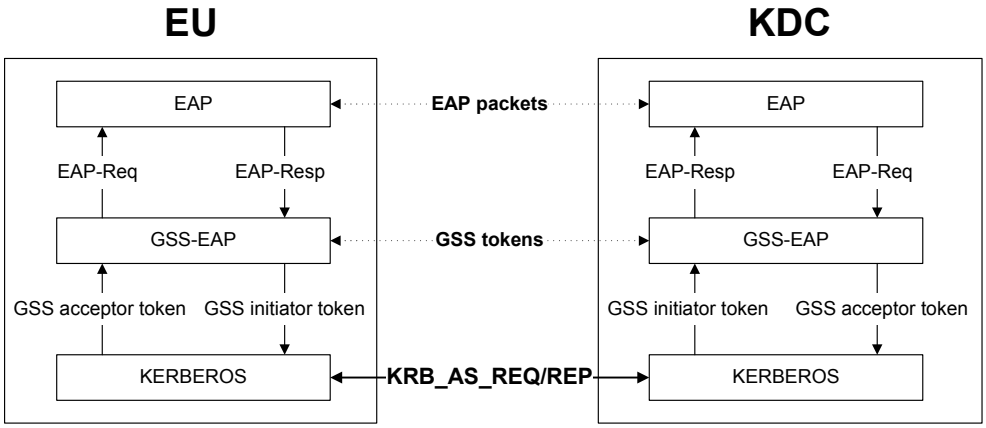


Figure 3.2: Different layers at which the EU and KDC operate.

3.4.1 TGT acquisition (KRB_AS_REQ/REP exchange)

As explained in Section 2.3.2, application protocols can use the GSS-API to obtain security services such as authentication, integrity protection or confidentiality. Having this in mind, this chapter defines how Kerberos itself can make use of the GSS-API to pre-authenticate the EU. That is, we define a new pre-authentication method based on the use of the GSS-API, where the messages of the KRB_AS_REQ/REP exchange will contain a GSS-API token. Hence, by using this mechanism, the EU and the KDC can use any of the authentication mechanisms available through the GSS-API. In particular, one of the mechanisms under study is based on EAP [107], where a GSS-API token carries an EAP packet along with some control information. Figure 3.2 depicts the different layers at which the EU and KDC operate.

The remainder of this subsection provides a detailed description of the steps required to successfully accomplish the Kerberos pre-authentication process, and the provision of the TGT from the KDC to the EU. Figure 3.3 details this exchange of messages:

1. The EU starts the process by calling to the *GSS_Init_sec_context* function. This function returns a GSS token that must be sent to the KDC. In particular, as the GSS-EAP mechanism is in use, the obtained token encodes an empty value indicating the start of the EAP authentication. In addition to the GSS token, the *GSS_Init_sec_context* function also returns a *GSS_S_CONTINUE_NEEDED* status value, indicating that more data is required from the KDC to complete the authentication. Then, the GSS token is encoded in a new type of *padata*, defined by this thesis, called *PA-GSS*. This *padata* is included into the *KRB_AS_REQ* message

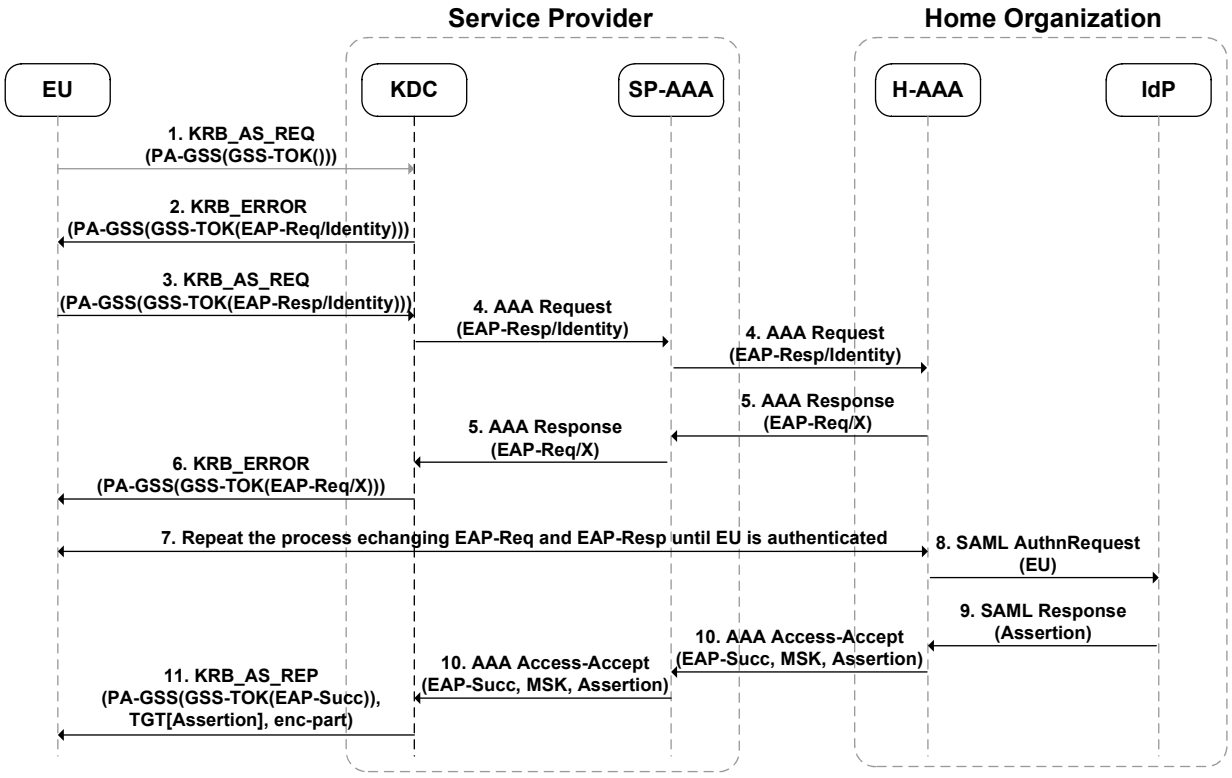
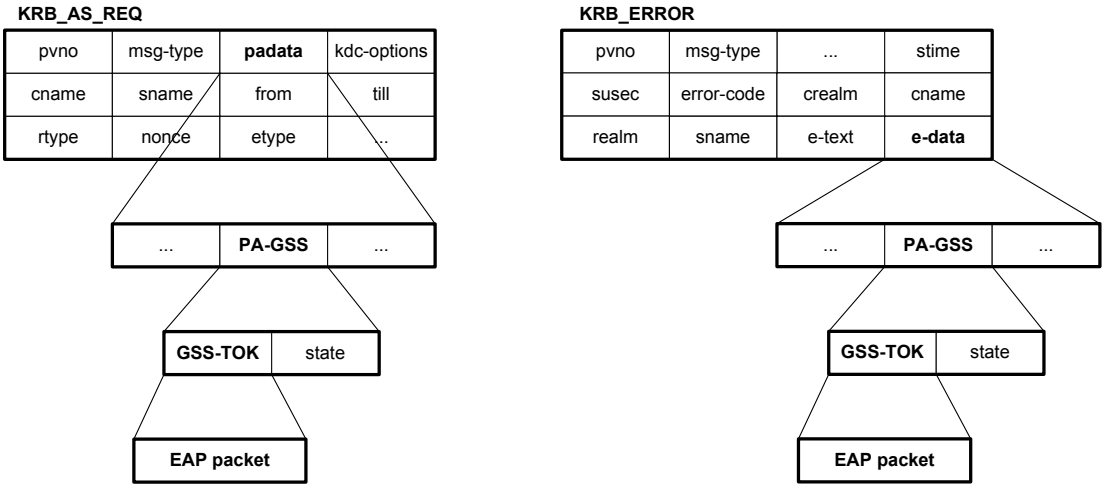


Figure 3.3: Kerberos KRB_AS_REQ/REP exchange: TGT acquisition.



(a) KRB_AS_REQ message. (b) KRB_ERROR message.

Figure 3.4: Encapsulation of GSS-EAP tokens in Kerberos messages.

sent to the KDC. Figure 3.4a details how this encapsulation is performed.

2. When the KDC receives the message, it invokes the *GSS_Accept_sec_context* function, using the received GSS token as input data. As a result, the KDC obtains a GSS token to be sent to the EU. In this case, the token contains an *EAP-Req/Identity* packet. The KDC also obtains a *GSS_S_CONTINUE_NEEDED* status value, indicating the authentication requires more interactions to be completed. Therefore, the GSS token is encoded in a *PA-GSS padata*, and included into the *e-data* field of a *KRB_ERROR* message. Figure 3.4b details how this encapsulation is performed. The error code of this message is *KDC_ERR_MORE_PREAUTH_DATA_REQUIRED*, as indicated in [103] for multi-roundtrip pre-authentication mechanisms.

3. The EU calls the *GSS_Init_sec_context* using the GSS token received within the PA-GSS *padata*. As a result, the EU obtains a return value of *GSS_S_CONTINUE_NEEDED*, and a new GSS token containing the corresponding *EAP-Resp/Identity* packet. This EAP packet encodes the EU's identity (e.g. *alice@home.org*, or *anonymous@home.org* if a tunnelled EAP method is used). Similarly to the previous step, this GSS token is sent to the KDC in a KRB_AS_REQ message.

4. When the KDC invokes the *GSS_Accept_sec_context* function to process the received GSS token, the GSS-EAP mechanism forwards the EAP packet within the token to the H-AAA, using the SP-AAA as intermediary, and the AAA protocol as transport. Note that the KDC is not aware of that process, and it just keeps waiting for the resulting GSS token and return value.

5. The H-AAA answers the request with a new AAA response including the first EAP packet of the EAP-method selected for the end user (*EAP-Req/X*).

6. When this message reaches the KDC (which was waiting until the execution of *GSS_Accept_sec_context* was completed), the GSS-EAP layer encapsulates it into a GSS token and provides it to the Kerberos layer. Then, the token is sent to the EU in a KRB_ERROR, in a similar way than in the previous interactions.

7. From this moment, EU and KDC engage in a series of *KRB_AS_REQ/KRB_ERROR* exchanges that continue until the H-AAA considers the EAP method is finished.

8. Once the EAP method reports the EU has been successfully authenticated (i.e. EAP-Success), the H-AAA requests a SAML assertion from the IdP. This request is performed by means of a SAML *AuthnRequest* message, where the *Subject* element

represents the identity of the EU. Appendix B provides an example of such an assertion.

9. As a response to this message, the IdP generates an assertion containing a SAML *AuthnStatement*, which points out that the EU has been successfully authenticated, and provides a transient pseudonym in the *Subject* element, as introduced in SAML (Section 2.2.3. This pseudonym can be used later during the KRB_TGS_REQ/REP exchange (Section 3.4.2) to retrieve further identity information about the EU (i.e. authorization attributes). It is worth noting that the SAML standard allow an assertion to also include attribute statements. This would allow omitting some of the steps described for the KRB_TGS_REQ/REP exchange. The main problem of this approach is that, at this point, the IdP cannot discriminate what attributes should be returned, as the target AppS is still unknown to the H-AAA server and to the KDC.

   The provided assertion is valid only for a limited period of time (defined by the IdP), after which the EU would be required to repeat the authentication process to obtain a new assertion. Some discussion about the trust relationships required for these exchanges, and the different alternatives for this protected channel can be found in Section 3.6.

10. When the H-AAA receives the assertion, it sends the AAA *Access-Accept* message to the KDC. This message contains the *EAP-Success* packet for the EU, the MSK derived by the EAP method, and the SAML assertion received from the IdP. While the transport of the first two elements is part of the standard AAA protocols (e.g. EAP-Message, MPPE-Recv-Key, and MPPE-Send-Key attributes [138] in RADIUS), a new RADIUS attribute has been defined to transport the SAML assertion to the KDC [139]. Section 3.7 provides more details about this aspect.

11. When the KDC receives the *Access-Accept* message, the *GSS_Accept_sec_context* call (that was stalled waiting for a response) returns the *EAP-Success* within a GSS token, as well as a status value of *GSS_S_COMPLETED*, indicating that the authentication has been completed.

    At this point, the KDC can use the *GSS-API naming extensions* [118, 119] calls to obtain the SAML assertion received from the H-AAA (*urn:ietf:params:gss:federated-saml-assertion*). As the authorization process will be performed during the KRB_TGS_REQ/REP exchange, the assertion is

included in the TGT that will be sent to the EU. In particular, the assertion is included into the *authorization-data* field of the TGT. For that purpose, this thesis has defined a new *authorization-data element*, called *gss-authorization-data*, that contains the authorization data obtained through the GSS naming extensions. As the TGT is encrypted and integrity protected with the TGS key, it provides a secure transport for the SAML assertion, as it will not be possible to eavesdrop or tamper it by any entity other than the KDC.

Once the TGT has been generated, the KDC sends the *KRB_AS_REP* message to the EU, containing the obtained GSS token (transporting the EAP-Success packet) within a *PA-GSS* element. However, as the EU and the KDC do not share any previous shared secret (e.g. user password), they first need to agree on a fresh *reply key*, used to protect the *enc-part* of the *KRB_AS_REP* message. The *enc-part* of the message contains the session keys the EU needs to know to make use of the TGT in following interactions. The cryptographic material exported by the EAP method (i.e. MSK) is the perfect candidate to be used as root key to derive the *reply key*. Although the MSK is not directly available to the Kerberos layer, it is internally used by the GSS-EAP layer as a source of entropy for the *GSS_Pseudo_random* function [107]. Hence, both the EU and the KDC can use that function to derive a common *reply key*.

12. Similarly, when the EU receives the packet, it processes the GSS token and obtains a *GSS_S_COMPLETED* status value. Then, it uses the *GSS_Pseudo_random* call to derive the *reply key*, and then decrypt the received information.

After a successful authentication, the EU ends up with a TGT in possession. This TGT contains, in addition to the standard Kerberos information, some authorization information obtained from the EU's IdP. The following section focuses on the authorization tasks required during the issue of STs.

3.4.2 ST acquisition (KRB_TGS_REQ/REP exchange)

Once the EU has the TGT, she can use it to request one or more STs to access the different application services deployed by the service provider, following the standard Kerberos procedure (i.e. KRB_TGS_REQ/REP exchange). This section describes how standard authorization technologies can be integrated with this process, in a way that the KDC is provided with a rich set of identity information about the EU to determine whether she should be granted to access the requested application services.

In particular, this authorization process is defined by means of the SAML assertion contained in the TGT. We also also propose the use of XACML, which provides a standard policy language (used to define access control policies), and a well-defined authorization framework, to let the KDC to query a PDP to obtain an authorization decision.

The authorization management workflow is depicted in Figure 3.5, and explained in high detail in the following:

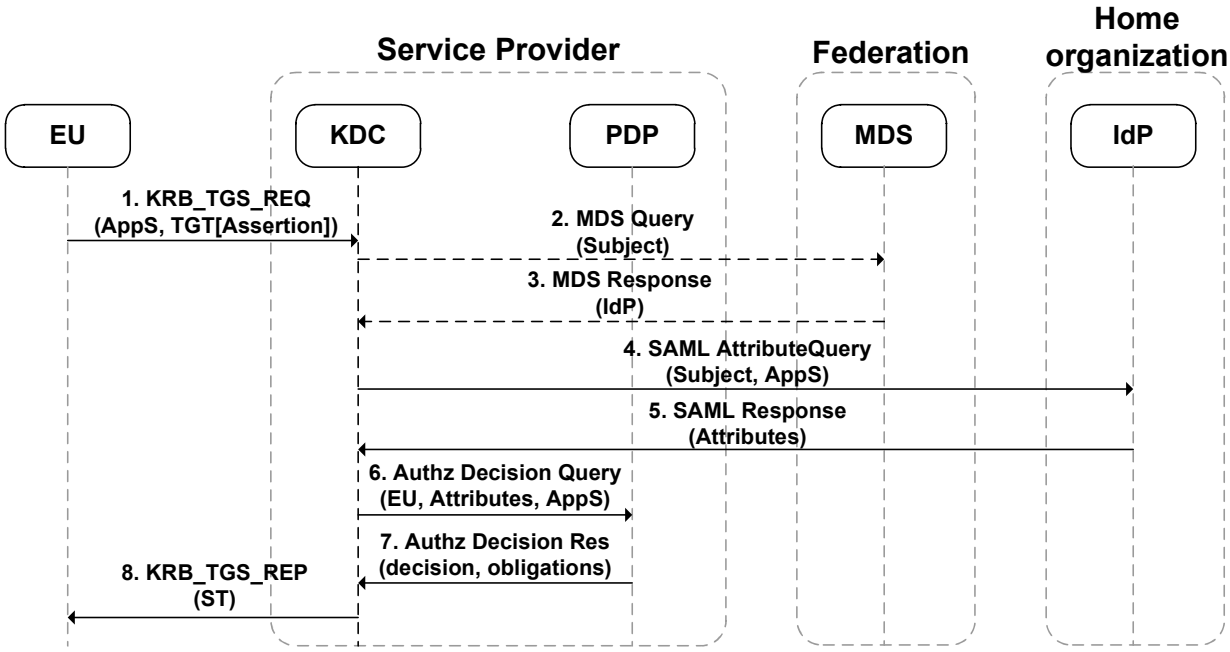


Figure 3.5: Kerberos KRB_TGS_REQ/REP exchange: ST acquisition.

1. The EU starts the Kerberos KRB_TGS_REQ/REP exchange by sending a KRB_TGS_REQ message to the KDC. This message includes the identifier of the AppS the EU wants to access to, and the TGT obtained as a result of the Kerberos KRB_AS_REQ/REP exchange.

2. The KDC verifies the TGT and extracts the SAML assertion contained within it. If the identity information contained within the assertion (e.g. end user attributes) is enough to take and authorization decision for the specified AppS, the process continues as explained for step 6. On the contrary, if the KDC requires additional information, it will request it from the EU's IdP. In order to discover IdP's location (e.g. URL), the KDC may consult the MDS. In such a case, the KDC sends a *MDS Query* message to the MDS, indicating the assertion's Subject element, used as EU's

identifier. This step may be omitted for intra-domain or small federated scenarios, where this information may be pre-configured on every member.

3. The MDS replies to this request providing the IdP's location.

4. Once the KDC has discovered the location of the IdP, it sends a new SAML *AttributeQuery* message. This message contains the *Subject* element from the assertion, in order to identify the EU. It also includes the information of the AppS the EU wants to access. Optionally, it could also include information about the set of required attributes (e.g. role, entitlement, subscription type, etc.). As commented before, typically SAML messages are digitally signed by both peers, and transported over a secure channel (e.g. HTTPS/SOAP), hence confidentiality is assured.

5. After receiving the request, the IdP check whether the requested attributes can be provided to the requesting AppS, according to some attribute release policies established either at an organizational level or personalized by each EU. Authors of [140] propose the use of XACML to manage these policies. After that, the IdP issues one or more *AttributeStatement* to the KDC, containing the subset of attributes that are releasable, according to those policies.

6. Finally, the KDC, acting like a PEP, issues a SAML/XACML *AuthzDecisionQuery* to the PDP controlling the XACML access control policies for the service provider. These policies describe the circumstances under which access to the AppS will be granted. For example, access to a specific service at one university may be granted only to those EUs presenting an attribute called *professor/lecturer*, indicating they are part of the teaching staff of any of the universities affiliated to the federation. The KDC provides the PDP with the obtained EU's attributes, the AppS identifier, and the required action (typically *access*).

7. After checking the policies, the PDP issues a SAML/XACML *AuthzDecisionStatement*, where a PERMIT/DENY answer is given. The PDP may also include some *Obligations* to adjust the service provisioning to the EU.

8. If PDP permits the access, the KDC creates the ST and delivers it to the EU.

With the ST, the EU can eventually access the service by means of a standard *KRB_AP_REQ/REP* exchange with the AppS. Usually, this exchange is carried out by means of the GSS-KRB mechanism. If the EU presents a valid ST, she is assumed as correctly authenticated and authorized to access the provided AppS. Therefore, the

deployed application services do not need to be modified as the federated access control is entirely performed by the KDC.

An alternative way to perform the authorization process would be delegating it to the application services. In such a case, the assertion received in the TGT would not be processed by the KDC, but inserted in the *authorization-data* field of the ST. IdP discovery, end user attributes retrieval, and authorization decision requests would then be performed by the AppS instead. Note that in this case, application services would need to be modified to perform all this processing.

3.5 Discussion

Previous section has described the proposal's operation in detail. After that, this chapter continues with the discussion of some important aspects that must be taken into account when deploying the proposal.

3.5.1 Federated user name in Kerberos

This chapter's proposal implies that the EU's name and password will not be present in the KDC's database during pre-authentication. This would typically result into the generation of an error of type *KDC_ERR_C_PRINCIPAL_UNKNOWN*. To avoid this situation, this chapter proposes the definition of a special identifier to be used in the *cname* field of the *KRB_AS_REQ* message. When the KDC receives a request with such a client name (Section 3.4.1), it will avoid checking the local end users' database, and will completely rely on the client information (i.e. client name and reply key) provided by GSS-API pre-authentication mechanism (e.g. *alice@home.org*).

In particular, Kerberos allows the definition of principal names that have special meanings [141]. These identifiers are called *Well-Known Kerberos Principal Names*, and consist of two components separated by the "/" symbol: the "WELLKNOWN" keyword identifying it as a well-know name, followed by a specific keyword identifying its particular meaning. For example, [142] defines the "WELLKNOWN/ANONYMOUS" name to identify an anonymous user. In this line, this chapter proposes the "WELLKNOWN/FEDERATED" principal name to be used as *cname* whenever the EU tries to access to a KDC out of her home domain.

3.5.2 Kerberos cross-realm vs Kerberos with AAA integration

For situations where the EU requests access to AppS not located in her home organization, Kerberos already defines the cross-realm operation (Section 2.3.1). According to the standard Kerberos specification, summarized in Section 2.3.1, the EU needs to interact with all the KDCs that take part of the path from her home organization to the service provider.

In contrast, this chapter proposes a new approach, where the EU only needs to interact with a single KDC: the service provider's KDC. More specifically, the Kerberos pre-authentication process (executed against the service provider's KDC) is performed through an underlying AAA infrastructure, which could imply one or more organizations in federated environments. As it can be observed, this solution avoids intermediate organizations to deploy KDCs to assist the process.

This is specially relevant on organizations where an AAA infrastructure is already deployed, typically for controlling the access to the network service. For them, deploying an additional cross-realm infrastructure may imply high administrative efforts (e.g. manage more firewall rules or having redundant user databases), as well as the introduction of new points of failure on the organization.

3.5.3 KDC state management

As described in [103], KDCs are stateless due to there is no requirement that an EU will choose the same KDC for the second (and following) requests in a typical Kerberos conversation (for example those described in Section 3.4.1. Moreover, even if the same KDC is selected, the KDC process is allowed to be stopped and started again between EU's requests, losing any state that it could possibly have. This is inconvenient for any multi-roundtrip pre-authentication mechanism, such as the one described in this chapter, which will usually require the maintenance of some context to bind the whole conversation. The Kerberos pre-authentication framework (Section 2.3.1) points out that such a context must be exported and delivered to the EU instead, which must treat it as an opaque piece of data, and include it in the next request sent to the next KDC. Upon reception, the KDC will reconstruct the context, and thus, be able to process the request.

In particular, exporting a context is pretty simple from a GSS-API standpoint, as there is a call for that specific purpose, *GSS_Export_sec_context*, which provides a binary representation of the context, abstracting the KDC from the complexities of the exporting process. Although current GSS-API specification precludes the exportation of security

contexts until they have been completely established, the GSS-EAP mechanism omits this restriction and allows it before establishment. Indeed, it is expected that any future multi-roundtrip GSS mechanism will allow it too, forcing the GSS-API specification to be updated accordingly.

There are some security implications associated to the context transfer. On the one hand, the context may contain sensitive information that should not be accessible to anyone except for the KDC. This can be easily solved by encrypting the context token using a key that all the KDCs in an organization share: the TGS key, typically used to encrypt the TGT. On the other hand, even when encrypted, an attacker may try to replay the context from one conversation to another, in an attempt of either gaining access to the resource or just provoking a Denial of Service (DoS) attack [143]. This is mostly avoided by including a timestamp into the context, reducing its usability to a limited period of time.

Therefore, the *PA-GSS padata* defined in Section 3.4.1 must include, in addition to the GSS-API token, a *state* element that contains the exported GSS-API context along with the timestamp of the moment of generation. In particular, the following is the ASN.1 description of the PA-GSS element, including all the required elements:

```
 PA-GSS ::= SEQUENCE {
   sec-ctx-token [0] OCTET STRING,  -- contains GSS-TOKEN
   state [1] EncryptedData OPTIONAL -- contains PA-GSS-STATE
}

PA-GSS-STATE ::= SEQUENCE {
   timestamp [0] KerberosTime,
   exported-sec-ctx-token [1] OCTET STRING
}
```

3.5.4 Transport of authorization information in RADIUS

So far this chapter has assumed a generic AAA transport for the proposed architecture. However, there is a major limitation when RADIUS is used as the AAA protocol. In particular, RADIUS limits the packet size to a maximum of 4096 bytes (Section 2.1.1). Although this has been sufficient for many scenarios and applications along the years, the transport of amounts of data over that limit, required for some scenarios, is not possible. In particular, the proposal described in this chapter requires the AAA infrastructure to transport a SAML assertion from the H-AAA to the KDC (Section 3.4.1). As this assertion may exceed the 4096 bytes limit, a solution is required when RADIUS is used.

Although a trivial answer to this problem would be to use Diameter instead, which uses TCP as its transport and, therefore, does not suffer from the aforementioned limitation, there are scenarios where switching from RADIUS to Diameter is not a viable solution. This is specially relevant for large federations. Many organizations have specific hardware implementing the RADIUS protocol (e.g. routers). As switching to Diameter requires to upgrade the whole AAA infrastructure (all the intermediary proxies need to be upgraded as well), it would imply a significant economic effort that not every organization in the federation will be willing to assume. Specially when the capability of sending/receive large AAA packets may be restricted to a small subset of the federation members. A clear example of such a RADIUS-based federation would be *eduroam*.

The IETF have tried to find out some solutions to overcome this issue. However, they do not fully cover the requirements of the scenario described in this chapter. For example, RFC 6158 [48] recommends the use of references instead of values, similar to the *Hash and URL* mechanism defined for IKEv2, or the *Filter-Id* attribute of RADIUS. However, the use of references rather than values is not applicable when the nature of the data to be sent is highly dynamic, such as happens with SAML statements (each SAML assertion has its own ID and timestamps).

Under this scenario, the RADEXT WG was requested to include this problem within its charter topics, as it was considered as an important issue of current RADIUS specification. Moreover, as part of the work of this thesis, a new draft [51] was submitted to that WG with a preliminary proposal for a fragmentation mechanism for RADIUS. This mechanism allows splitting a single RADIUS packet into smaller pieces of data called *chunks*. Each chunk is a RADIUS packet on its own and, thus, they follow all the rules and restrictions applicable to them. These chunks are sent from the sender to the receptor following the standard RADIUS exchanges. Once all the chunks have been received by the receptor, the original packet is reconstructed and processed as a whole. As one of its objectives, this fragmentation mechanism works through unmodified RADIUS proxies, allowing two updated peers to exchange large RADIUS packets through any existing RADIUS infrastructure.

This draft has evolved through several versions, and it was adopted as WG document on August 27th, 2013. Although this fragmentation mechanism has been designed as a consequence of the requirements of this thesis, it is not an essential part of the thesis. Hence, the reader is instead referred to the draft text available in [51].

In addition to the specification, we developed an open source proof of concept

implementation[1] [144], based on the FreeRADIUS code [145].

The RADEXT WG has recently adopted another proposal [146], based on the use of TCP and the removal of the artificial 4096 limit. Although this proposal is simpler than ours, it does require the modification of all the proxies of any existing RADIUS federation to support it, suffering from almost the same drawbacks than switching to Diameter. Therefore, both proposals target scenarios are different. While our proposal focuses on already deployed RADIUS infrastructures, this one focuses on to-be deployed infrastructures. That is the reason why the RADEXT WG opted for adopting both of them.

3.6 Security analysis

The use of EAP for Kerberos pre-authentication has also some implications in terms of security, specially regarding to the key management and distribution. In fact, the security analysis described in [147] for AAA key management, and the one described in [65] for EAP are applicable here. Indeed, the MSK exported by the H-AAA must be transmitted to the KDC, which is acting as EAP authenticator. Potentially, intermediate AAA proxies placed between the KDC and the H-AAA server can observe the distributed MSK that will be used to derive the Kerberos *reply key*. This kind of behaviour might affect to the security of this key.

However, the trust model in such federated environments assumes that intermediate AAA proxies can be considered as trusted entities, and therefore MSK can be safely distributed to the KDC through the AAA infrastructure. As [147] explains, some key wrapping techniques can be applied to provide confidentiality, integrity and replay protection to the distributed key material between each pair of AAA entities (e.g. AAA proxies).

Besides, this work assumes there is a transitive trust relationship for authentication between the involved organizations thanks to the deployed AAA infrastructure, as described in Section 3.2. This trust is usually based on pre-shared keys or on PKI architectures. Either way, they need to be pre-configured on the AAA entities within the federation.

Regarding the authorization process, it is generally assumed the use of a direct trust relationship, allowing the protection of SAML messages between organizations (see Figure 3.5, step *(4)*). Usually PKI architectures are used to deploy this trust [148],

[1]Funded by Telefonica I+D

and public key certificates and private keys are implemented in services and identity providers, which allow the establishment of secure channels and protecting messages (it is worth noting that end users still make use of login/password credentials). Following this approach, IdPs and KDCs should be fed with this cryptographic material.

Nevertheless, a different approach could be followed to avoid the deployment of an additional infrastructure (e.g. PKI). The transitive trust relationship defined by the deployed AAA infrastructure could be leveraged to convey and perform the attribute recovery process in a protected manner (SAML AttributeQuery/Response exchanges). That is, the AAA protocol could serve as a transport for the SAML protocol between the KDC and the IdP, where confidentiality, integrity protection and authenticity is provided by the trust relationships established within the AAA-based federation, and the mechanisms provisioned by the used AAA protocol. Moreover, as being investigated by Moonshot, a new key management entity called *Trust Router* [149] can be used to dynamically bootstrap a direct trust relationship between the service provider's AAA server and the home AAA server. This would allow the dynamic building of AAA-based federations.

Although using this last approach would be plausible, it was discarded for this proposal as currently deployed IdPs such as Shibboleth do not support receiving attribute queries through AAA protocols. Besides, the *Trust Router* is still on early stages of specification and cannot be considered stable. Hence, this approach is left as a possible future work (Section 7.2.4).

3.7 Conclusions

This chapter has analysed several well-known standard protocols and technologies which are used to authenticate and authorize end users when accessing to application services in identity federations. In particular, Kerberos is being widely used for authentication and key distribution in application services within a single organization. However, Kerberos cross-realm infrastructures are not widely deployed for federated environments. Conversely, organizations are usually interconnected by means of AAA infrastructures to control access to the network service. This separation of technologies (service access vs. network access) makes enormously difficult that end users subscribed to one organization can access by using Kerberos to the application services provided by other organization within the federation.

The proposed architecture offers important benefits to both, service providers and end users. The formers are benefited of a federated authentication process that,

without modifying the deployed application services, allows them to increase the potential number of users and, thus, the business opportunities. Besides, thanks to the advanced authorization, they have a more precise control over who can access the application services, and under which conditions. The end users are benefited as well from the federated authentication process, as they do not need to remember dozens of different credentials for different service providers. Moreover, thanks to the use of Kerberos, they are also able to access different application services within the same service provider by means of the SSO mechanism, that is, without incurring in repeated authentication processes with their home organization.

Finally, this chapter has analysed and discussed the most important deployment and security aspects, as well as the standardization status of the different technologies involved in the architecture. The functional and performance analysis has been delayed to Chapter 6, for a comparison with the rest of contributions of this thesis.

It is worth noting that all the components required by this chapter are either consolidated standards or being considered for their standardization. In particular, the ABFAB WG has standardised the GSS-EAP specification [107, 119], as well as a RADIUS transport for SAML [139]. As a result of the work in this chapter, a GSS-EAP pre-authentication mechanism for Kerberos [150] has been defined. Besides, a GSS-API pre-authentication mechanism [135] has been proposed within the Kerberos WG [126]. Finally, the RADIUS fragmentation mechanism [51] is also being under discussion within the RADEXT WG [50].

Although FedKERB provides a solid solution to provide federated access to a wide range of different application services, it still presents some aspects that could be improved for specific scenarios. For instance, although application services do not need to be modified, this proposal does require the modification of the KDC element to support the GSS-API pre-authentication. In some scenarios modifying the deployed Kerberos infrastructure may not be a viable solution. Besides, when the end user performs a network access control process to access the network, using the same AAA infrastructure (as it happens in *eduroam*), the authentication with the KDC implies an additional (and redundant) EAP authentication that may be avoided by some kind of cross-layer communication. Chapters 4 and 5 propose alternative architectures dealing with these two scenarios, respectively.

Chapter 4

PanaKERB: Out-of-band federated authentication for Kerberos based on PANA

4.1 Introduction

The previous chapter of this thesis has proposed FedKERB, a unified architecture which is able to bring the benefits of identity federations to any kind of application service, even to those that were not designed to support a federated operation (e.g. SSH or NFS). This is achieved by the integration of Kerberos and AAA infrastructures, in a way that the resulting architecture benefits from the advantages they have separately. However, although reduced to a minimum, this integration has required some extensions and modifications to the existing Kerberos implementations, in order to support the definition of a new pre-authentication process based on the use of the GSS-API and, more specifically, of the GSS-EAP mechanism. This implies that the KDCs deployed at service providers, as well as Kerberos client at end users, need to be updated accordingly.

However, some service providers may be reluctant to inflict changes on their network equipment and software. For those providers, this aforementioned requirement may preclude the adoption of the FedKERB proposal, therefore leaving them with no possibility of joining the identity federation. Moreover, even those service providers that would agree to introduce the required changes in a long-term plan, may still have a preference for a interim and less-intrusive way to provide federated access to their services, for instance, to evaluate the actual interest of their users on such a service.

For these reasons, this chapter proposes an alternative architecture to allow the

integration of Kerberos and AAA infrastructures without the modification of the existing elements. In particular, this chapter tries to address the following objective, extracted from Section 1.3:

O2 To design a solution enabling the bootstrapping of Kerberos credentials on the service provider as a result of a federated AAA-based authentication process performed by means of an independent protocol (*out-of-band*) with native support for AAA infrastructures, avoiding the modification of neither the elements of the Kerberos infrastructure, nor the application services.

This objective can be broken down into the following sub-objectives:

- To design an architecture for federated access to application services by means of an *out-of-band* Kerberos authentication processes.
- To reduce the architecture complexity by reusing standards as much as possible, minimizing the impact on current technologies.

This chapter presents a novel solution, called PanaKERB, following a different strategy to solve the aforementioned problems: it defines an authentication mechanism parallel and independent of Kerberos (*out-of-band mechanism*) that authenticates and authorizes the end user against the EAP/AAA-based federation. The out-of-band authentication process is completely transparent to the Kerberos infrastructure (i.e. KDC) deployed by the service provider. That is, the KDC is not aware of the authentication process. As a result of a successful authentication, a new state is enforced in the KDC. This state allows authenticating the end user, and establishing a security association with her. With this security association, the end user obtains a Kerberos credential to access the application service, following the standard Kerberos operation. To convey the EAP authentication information between the end user and the service provider, the *Protocol for Carrying Authentication for Network Access* (PANA) is used as the out-of-band protocol (Section 2.1.3). The main reason for using PANA are described in Section 4.2.1: a) it is a consolidated standard for the purpose of authenticating end users using an AAA infrastructure, and then enforcing a state on a *Enforcement Point* [43]; and b) it provides a lightweight mode of operation when compared to other protocols [151].

Hence, this solution addresses the above-mentioned drawbacks of FedKERB: first, applications services that currently support Kerberos do not need to be modified; second, neither Kerberos protocol nor existing implementations need to be modified. The simplicity

of this approach will make its adoption and deployment easier, and it is especially interesting to organizations that have kerberized application services already deployed.

In addition to the proposal described in this chapter, there exist other alternatives that also try to make use of an out-of-band mechanism to perform an EAP/AAA authentication in order to bootstrap a Kerberos security association afterwards. The most relevant solution applying this strategy can be found in the *Bootstrapping Kerberos* [122] proposal. This work proposes to provide the end user with a TGT valid for the service provider's domain after a successful network access control process based on EAP and PANA. In other words, instead of using the standard *KRB_AS_REQ/REP* exchange, the TGT is directly distributed to the end user along with the *EAP-Success* message. However, this procedure not only violates the Kerberos specification (which states that TGTs must be provided to the client through a *KRB_AS_REQ/REP* exchange), but also requires the AS functionality to be integrated within the AAA server at the home organization. Moreover, this work also inflicts changes in the *EAP Key Management Framework* (EAP KMF) [65], since the MSK is directly inserted in the TGT. Hence, the *Bootstrapping Kerberos* solution is not suitable for the problem statement of this thesis.

This chapter is structured as follows. Section 4.2 describes the different entities that take part of the proposed architecture. Section 4.3 explains how these entities interact to achieve the objective of granting access to federated users. Section 4.4 discusses the security considerations of this proposal. Section 4.5 outlines some conclusions.

4.2 Proposed Architecture

This section presents the proposed out-of-band federated authentication mechanism for Kerberos, pointing out some initial considerations that have to be taken into account. It also describes the architecture components.

4.2.1 Preliminary considerations

This chapter proposes an architecture with the following requirements:

- As members of the federation, both organizations deploy AAA servers that allow them to exchange authentication, authorization and accounting (AAA) information.
- The service provider deploys a Kerberos KDC to distribute service tickets to access the application services deployed on the organization.

- The application service uses Kerberos as the means for performing access control.
- The service provider deploys a *PANA Authentication Agent* (PAA) to enable the out-of-band authentication for Kerberos.
- The end user makes use of EAP to perform authentication with the home organization's AAA server, using the PAA as EAP authenticator.

In our architecture, PANA protocol is used as a glue layer between these two infrastructures, allowing federated end users to carry out an authentication process based on the EAP/AAA federation, resulting in the bootstrapping of the required Kerberos credentials in the service provider. In particular, a Kerberos credential (i.e. principal name and password) will be derived from the *PaC-EP Master Key* (PEMK) after the PANA authentication, and enforced in the KDC afterwards. For this enforcement, the PAA will use the KADM [7] interface defined by the Kerberos standard for that purpose.

One key question to answer is why PANA. In general, a protocol which is able to operate as EAP lower-layer between the EAP peer (end user's device) and the EAP authenticator through multiple IP hops is needed. This is required to solve the proposed scenario since, in the most typical situation, the end user will not be physically connected to the service provider's network. As discussed in [152] and [151], IKEv2 and PANA are valid candidate technologies. However, as it is also concluded in these references, PANA provides a lighter operation since it only involves an HMAC [153] operation for providing packet authenticity. In contrast IKEv2 always requires asymmetric cryptography and, in general, encryption for providing confidentiality and authenticity. Indeed, regardless of cryptographic operations performed by the EAP method itself, IKEv2 requires the execution of an additional Diffie-Hellman key exchange [154], which is computationally more expensive than the symmetric cryptography operations required to establish the *PANA Security Association* (PANA SA). Moreover, the standard IKEv2 does not allow exporting key material after a successful authentication for other purpose different than establishing an IPsec tunnel. Thus, the password generation and posterior transference are not possible using IKEv2.

It also worth noting that PANA was initially designed for transporting EAP during network access service. The main motivation is that EAP was originally conceived to perform the authentication for network access service. However, EAP applicability has been recently updated and its usage has been extended for authenticating end users for application services (Section 2.1.2. In consequence, PANA can be potentially used for

application services since it is merely a transport of EAP. In fact, PANA is being used in environments where an application service bootstrapping is required, as described in [151].

Finally, the service provider is required only to deploy a new component: the PAA. This component will be the bridge between the PANA authentication process and the KDC in the service provider, so that the KDC will be configured with the required end user information to allow access to the local application services.

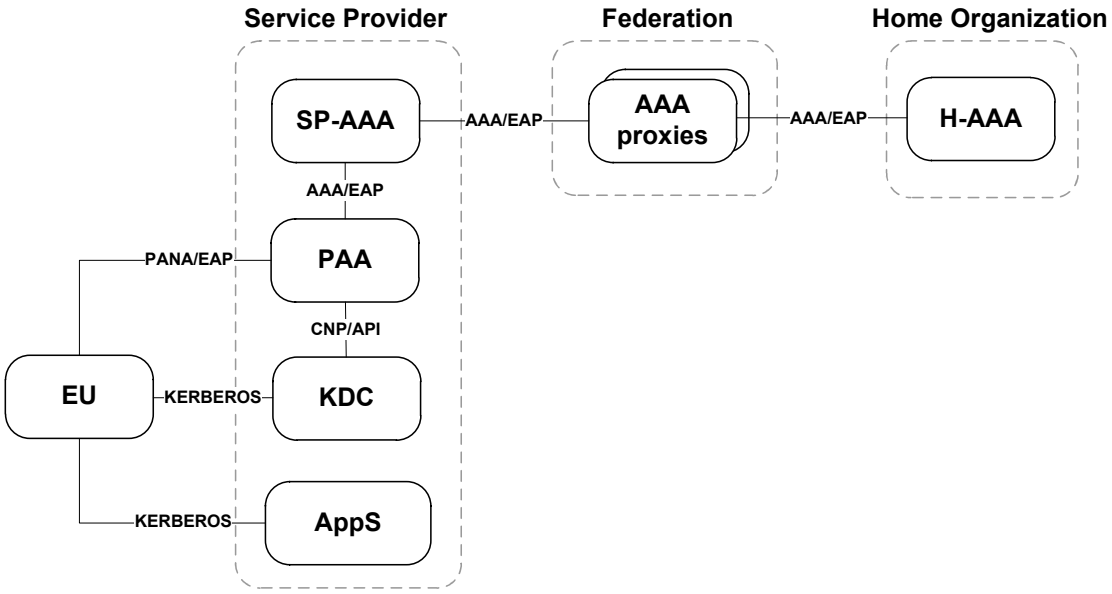


Figure 4.1: PanaKERB architecture.

4.2.2 Components

Taking into account these considerations, Figure 4.1 depicts the proposed architecture. As observed, it is composed of several components that communicate among themselves using different technologies:

- *End user* (EU). She is interested on accessing an application service (also called *kerberized* service) provided by the service provider. She makes use of her credentials from her home organization to authenticate through the PAA by means of EAP (Section 4.3.1). After that, she makes use of standard Kerberos in order to access the service, contacting with the KDC to obtain the required tickets. Thus, the end user acts as a *PANA Client* (PaC) first, and as a Kerberos client afterwards. In the use case of Section 1.2, this component corresponds to *Alice's device*.

Component	EAP	PANA	Kerberos	AAA
EU	Peer	PaC	Client	-
SP-AAA	-	-	-	Proxy
H-AAA	Server	AS	-	Server
PAA	Authenticator	PAA	-	Client
KDC	-	EP	KDC	-
AppS	-	-	AppS	-

Table 4.1: Roles played by the components on the different protocols.

- *Service provider's AAA server* (SP-AAA). This is the element that joins the service provider to the AAA federation. It is contacted by the PAA to transport the EAP packets from the EU to the AAA server in home organization, and vice-versa. The transport is performed through the AAA proxies that integrate the federation.

- *Home AAA server* (H-AAA). This is the entity which is able to verify the EU's credentials, acting as the EAP server. When the authentication ends successfully, a shared secret with the EU (the MSK) is derived and delivered to the PAA through the AAA transport (Section 4.3.1).

- *PANA Authentication Agent* (PAA). This is the element that communicates the Kerberos infrastructure with the AAA federation in the service provider. On the one hand, it authenticates the EU by means of EAP and the AAA infrastructure. On the other, once the user is successfully authenticated, it enforces a new principal name and password into the KDC's user database.

- *KDC*. It is the central entity of the Kerberos infrastructure, and its main responsibility is to authenticate EUs and provide them with tickets to access services. In this proposal, the KDC also acts as a PANA *Enforcement Point* (EP).

- *Application Service* (AppS). This element provides a valuable service to EUs. Access control is managed by Kerberos, so the service will be provided only to those presenting a valid ST. From the application service's point of view, there will be no difference between EUs from the service provider and EUs from other organizations.

Table 4.1 summarises the different roles played by the components of the architecture for each one of the protocols involved in the proposed solution. For the sake of simplicity, hereafter only the names on the top-left column will be used to refer to these components.

4.3 General operation

The process is structured in four different phases, which are explained in detail below:

- Phase 1: PANA authentication.
- Phase 2: Kerberos enforcement.
- Phase 3: Kerberos authentication.
- Phase 4: Obtaining service tickets and accessing the service.

4.3.1 Phase 1: PANA authentication

In this phase the EU is authenticated by the PAA, by using PANA and EAP. Through the EAP/AAA federation, the H-AAA is the entity which actually verifies the EU's credentials, asserting she has been successfully authenticated. The EU is assumed to have network connectivity before performing this phase.

The process starts when the EU wants to access an application service available on a service provider, which belongs to the same AAA federation as her home organization. Nevertheless, the EU needs to know whether the AppS supports Kerberos as its means of authentication. How the EU knows this fact may vary from one deployment to another. For example, the EU may know it beforehand based on federation agreements. Or the specific application protocol may announce the supported authentication mechanisms (e.g. GSS-API, Kerberos, SASL...). Nevertheless, this discovery mechanism is out of the scope of this work, and this chapter will assume the EU already knows that Kerberos is supported.

Regardless the EU already has a valid ST for that specific AppS, or if she has a valid TGT for that organization, she proceeds with the standard Kerberos procedure, as described in Section 4.3.4). However, if the EU has no Kerberos credentials, she starts the authentication with the PAA, using the credentials she already has from her home organization. The PANA authentication process may be automatically triggered when no suitable TGT is available. Figure 4.2 depicts this authentication process.

The first step of this phase is aimed to establish the PANA SA. The process starts negotiating a pseudo-random function (PRF) and an integrity algorithm (1, 2 and 3). They will be used to derive keys and to protect PANA messages respectively. After that, the PAA starts the EAP authentication by sending a *PANA-Auth-Request* message containing an *EAP-Request/Identity* packet and a *nonce* value (to avoid replay attacks)(4). The EU replies to this request by sending a *PANA-Auth-Answer* message with another *nonce*

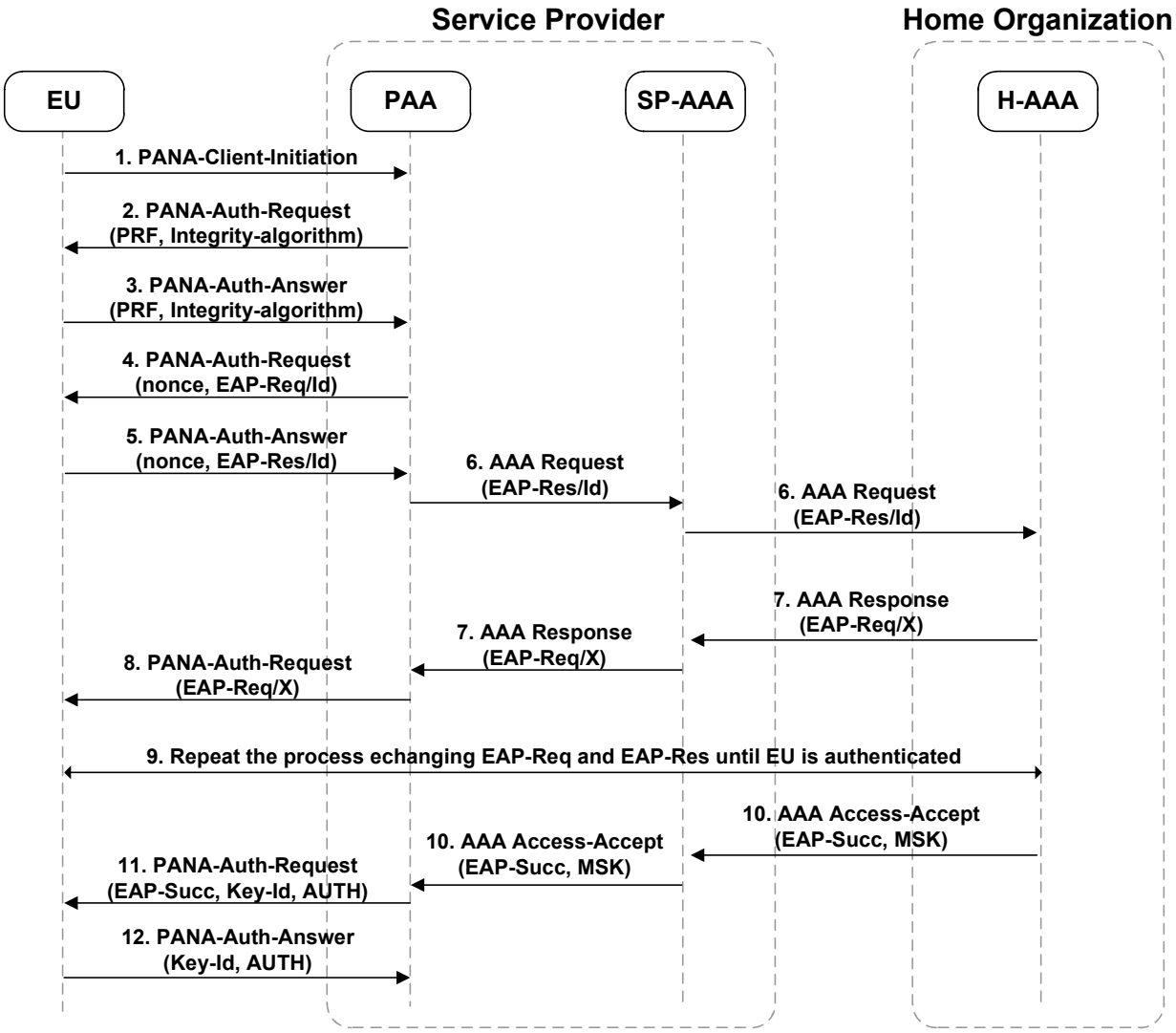


Figure 4.2: Phase 1: PANA authentication.

value and with an *EAP-Response/Identity* packet (5). The *EAP-Response/Identity* packet transports the end user's identifier (e.g. alice@home.org).

During the PANA authentication process, the EAP packets received by the PAA from the EU are forwarded to the H-AAA through the AAA infrastructure (6). Similarly, the EAP packets coming from the H-AAA are forwarded to the EU (7 and 8). Depending on the EAP method being used, the authentication process may require several round tips. Hence, this forwarding process is repeated until the EAP authentication finishes (9). For instance, the EAP-MD5 method requires two round-trips to be completed (one to propose a challenge, and the other for generating and verifying the hash), while other more complex

methods such as EAP-TLS may require several round-trips to complete.

Once the authentication process has successfully finished, the H-AAA includes the generated MSK in the final AAA *Access-Accept* message (10). This information is transported through the AAA infrastructure to the PAA, which ends the establishment of the PANA session and the creation of the PANA SA with the EU by the exchange of the *Key-Id* and *AUTH* AVPs (12 and 13). These AVPs are used to derive the PANA_AUTH_KEY [43] and the PEMK [155].

At this point, the EU has been authenticated by the PAA by means of the EAP/AAA federation, and some cryptographic material has been derived from this authentication (i.e. MSK). This PANA SA has been established between EU and PAA.

4.3.2 Phase 2: Kerberos enforcement

Right after executing phase 1, the PAA contacts with the KDC to enforce the access control state associated with the authenticated EU. More specifically, the PAA interacts with the KADM server in the KDC through the KADM interface. This server allows external entities to manage the KDC's database. Although the Kerberos specification does not define how this database should be, the *Kerberos Information Model* defined in [156] provides a common model to represent all this information, specifying the cornerstone elements that must take part on every KDC database (i.e. principals, keysets, keys and policies). It also describes their attributes and how they are interrelated. This KADM interface, which serves to manage this information model, can be implemented using several protocols, such as LDAP [110], SOAP or NETCONF [157]. As its specific implementation will depend on the particularities of each deployment, its selection is out of the scope of this thesis.

Without any loss of generality, we assume that the KADM server offers two methods to the PAA. On the one hand, the *KADM_create_principal* call allows the creation of a new Kerberos principal, indicating the *principal_name*, the *password*, and a *lifetime* after which all the tickets issued to the EU must expire. On the other hand, the *KADM_delete_principal* call allows the deletion of a principal from the database, using the *principal_name* as identifier. The arguments of these calls are mappable to the *principalName*, *keyValue* and *principalNotUsedAfter* attributes defined in [156] respectively.

The process of the Kerberos enforcement is depicted in Figure 4.3. First, the PAA makes use of the received MSK to derive a PEMK, following the indications given in [155]. The PEMK is a 64 byte-length binary key that will be used to derive the temporary principal name and password. For that, it needs to be adapted to the specific requirements of the KADM interface before it can be enforced. As most Kerberos implementations expect

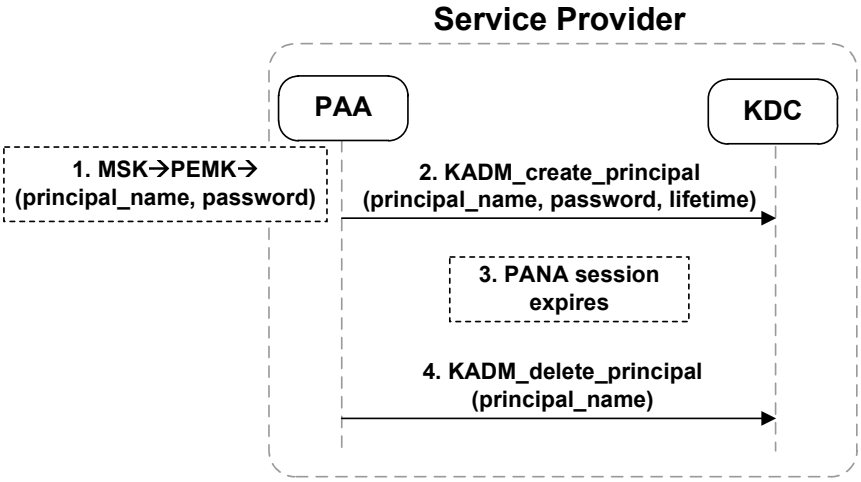


Figure 4.3: Phase 2: Kerberos enforcement.

human end users to introduce textual principal names and passwords instead of binary ones, KADM interfaces are typically prepared to receive variable-length text strings for those parameters. The password will then internally converted into the required key using the string-to-key function [7]. Hence, we propose to transform the PEMK into a textual representation by means of base64 encoding [158]. The resulting string will be 88 bytes length. The first 16 bytes will be used as principal name, whereas the last 72 bytes will be used as the textual password.

$$pemk = generate\_pemk\_rfc5807(msk)$$
$$pemk64 = base64\_encode(pemk)$$
$$principal\_name = substring(pemk64, 0, 15)$$
$$password = substring(pemk64, 16, 87)$$

Once the PAA has derived the EU's principal name and password (1), it can enforce the access control state into the KDC (2). This state includes the principal name and the password generated for the EU, and the lifetime of the new state. The principal name will be concatenated with the realm of the home organization. This will be useful for accounting purposes (e.g. know where the EU comes from). For instance, considering a home organization called *homeorg.com* and a service provider called *remoteorg.com*, an example of principal name created by the PAA into the KDC's database would be *MTIzNDU2Nzg5MDEy@homeorg.com@REMOTE.ORG*, where *MTIzNDU2Nzg5MDEy@homeorg.com* would be the principal name, and REMOTE.ORG would be the Kerberos realm corresponding to *remoteorg.com*. The lifetime will be established based on the PANA session lifetime.

When the PANA session expires (3), the PAA must take care of deleting the state associated to the EU from the KDC as part of the PANA termination phase. This action is also performed through the KADM interface (4).

4.3.3 Phase 3: Kerberos authentication

After the PANA authentication and enforcement processes, the EU can access any application service by following the normal procedure of the Kerberos protocol. Therefore, the first step of this operation is to obtain a TGT from the KDC, by means of a *KRB_AS_REQ/REP* exchange. This TGT will be used later to obtain STs for the different application services offered within the service provider. Note that if the EU might start Phase 3 before Phase 2 is completed. In such a case, the KDC will complain about not having such a end user on its database. The EU should then wait a small amount of time (e.g. 500 ms.) and retry again. Figure 4.4 represents the Kerberos authentication phase, showing the most relevant information transmitted between entities.

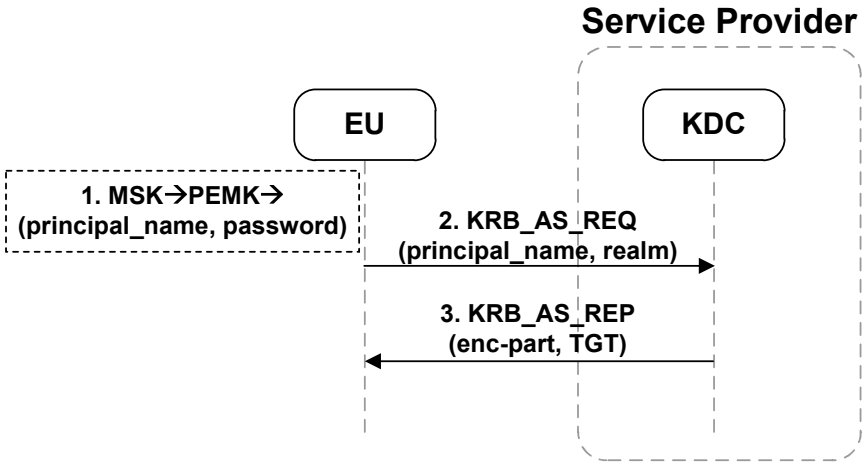


Figure 4.4: Phase 3: Kerberos authentication.

In order to authenticate with the KDC, the EU needs to make use of the same principal name and password that were enforced in the KDC. Therefore, she follows the same key derivation process as performed by the PAA in phase 2, obtaining the PEMK from the MSK, and then transforming it into a textual principal name and password (1) by applying a base64 encoding. With this principal name and password the EU is able to perform the standard *KRB_AS_REQ/REP* exchange with the KDC, sending a *KRB_AS_REQ* message (2) and receiving a *KRB_AS_REP* message containing a *enc-part* and a TGT (3).

Therefore, the KDC and the EU are provided with enough information to perform a standard Kerberos authentication as a result of the PANA authentication, without the need to modify any of these two entities.

4.3.4 Phase 4: Obtaining service tickets and accessing the service

This phase is entirely based on the standard Kerberos operation described in Section 2.3.1 for accessing an application service and, consequently, nothing needs to be modified in any of the involved entities (i.e. the KDC or the EU). Once the EU has obtained the TGT in the previous Phase 3, she can request an ST to access the desired application service by following two different steps: a KRB_TGS_REQ/REP exchange and a KRB_AP_REQ/REP exchange.

As a result, the EU can access an application service deployed in a service provider, by just using the credentials she shares with her home organization. As per described in this chapter, this proposal does not provides support for advanced authorization. In fact, an end user is considered authorized to access an application service as long as she is successfully authenticated. This is so because including additional authorization capabilities to the KDC would inevitably require to modify existing deployments, which would break with one of the objective of this out-of-band proposal: not modifying existing infrastructures.

However, if this requirement were relaxed, as some organizations may not oppose to introduce minor modifications, this proposal could be easily extended to support the authorization model described in the previous chapter (Section 3.4.2, with just a few minor modifications. In particular, the SAML assertion associated to the EU would be received by the PAA through the AAA infrastructure after the EAP authentication process. Therefore, to make it available to the KDC, the PAA would need to include the assertion into the data enforced into the KDC. This should not result into any inconvenient, as most KADM implementations allow the enforcement of opaque generic data, in addition to the basic elements of the information model described above.

Once the AS retrieves authenticates the EU, it retrieves the assertion from the database and introduces it into the TGT, as described in Chapter 3. From this point, the authorization process goes on as specified in Section 3.4.2.

4.4 Security considerations

As stated, this proposal aims to provide a complete solution that is able to adapt to a wide variety of scenarios, and that requires the minimal changes to the existing deployments and

implementations. Therefore, neither PANA, Kerberos, EAP nor AAA standard protocols have to be modified. Their specifications and their security properties remain unchanged, by keeping their weaknesses and strengths intact. This section summarizes some of these important security considerations which impact on our solution. Moreover, it discusses the main security aspects that arise from the combination of these technologies.

4.4.1 Key distribution after authentication

The distribution of key material is a critical security point, as unauthorized access by third parties may lead to impersonation attacks. Two different key distribution processes take place for the distribution of the MSK and EU's password.

In the first one, the MSK is transmitted from the H-AAA to the PAA as a result of a successful authentication process. In the federation, both entities may be connected through one or more intermediate AAA proxy servers placed on a domain on the path between the service provider and the home organization. Therefore, the same analysis as the one done for FedKERB in Section 3.6 is applicable here. That is, AAA proxies should be considered as trusted entities, and some key wrapping techniques can be applied to provide confidentiality, integrity and authenticity to this distribution.

The second one takes place when the Kerberos password derived from the MSK is transmitted from the PAA to the KDC via the KADM interface. As stated in [156], the password must not be transported in clear over the network. This aspect is covered with further details in Section 4.4.3.

4.4.2 Kerberos Password derivation

This chapter defines that the password between a federated EU and the KDC in the service provider is bootstrapped from the keying material resulting from the EAP authentication. The generation process of this password consists of the derivation of a key from the MSK named PEMK (Section 2.1.3) following the rules defined in the standard [155]. Then, this key is converted to its base64 notation to obtain a textual representation.

Besides the already commented benefits in terms of scalability, usability and simple deployment, the use of dynamically generated keys (transformed into passwords) for Kerberos provides a robust protection against off-line dictionary attacks [7, 103], when compared to standard Kerberos, where fixed textual passwords are used to authenticate EUs. Even if an attacker were able to recover the *reply-key* from a Kerberos conversation, that key would only allow decrypting that specific conversation, but will not jeopardise any

future or past Kerberos interaction. Besides, if the dynamic Kerberos principal already expired, the attacker will also be unable to generate new tickets to impersonate the EU.

It is worth noting that this dynamic password generation is only possible when a key-generating EAP method (e.g. EAP-TLS [59], EAP-TTLS [60]...) is selected. The use of EAP key generating methods is a requirement not only demanded by this chapter, but also recommended by existing standards [147, 159].

Once the key material has been bootstrapped between EU and the KDC, this solution not differ from the standard Kerberos operation in a single-realm scenario. Therefore, it does not preclude the usage of any of the security enhancements defined for Kerberos such as pre-authentication or FAST tunnelling [103].

4.4.3 Authenticated and Authorized enforcement in the KDC

Another relevant aspect is the access control to the KADM interface. That is, it must be ensured that only authenticated and authorized entities (i.e. the PAA) can enforce new principals in the users database. Moreover, it must also be ensured that they can only be modified by their rightful owner. Special care must be taken with those objects that are critical for the proper functionality of the whole Kerberos infrastructure, like the principal and keys shared between the AS and TGS servers. Therefore, KADM implementations are strongly encouraged to provide the means of performing access control (e.g. based on pre-shared keys or certificates). For example, the successful establishment of the TLS security channel between the PAA and the KADM can provide the required level of authentication and authorization. This does not preclude the use of other access control models, such as IPsec, or any secure *Remote Procedure Call* (RPC) mechanism offered by the Operating System (e.g. UNIX sockets), specially when the PAA and the KDC are collocated in the same machine.

4.4.4 Filtering the access to the PANA server

Since PANA operates over multiple IP hops, the EU may contact with the PAA in the service provider without being physically connected to the service provider's network. In general, this is an advantage since it allows the EU to access application services from any point of a network (not only from the service provider). This implies that the service provider may need to open port 716/UDP in firewalls, in order to allow PANA authentication coming from external users. Under certain circumstances it may be reluctant to enable this filtering policy, especially to avoid security attacks (e.g. DoS).

In such a case, the service provider may only allow the access to the PAA when the EU has actually roamed to its network.

This situation would also be possible with the KDC and the Kerberos protocol. Service provider firewalls might reject traffic to port 88/UDP, avoiding EUs to contact the KDC from outside their network. However, this would have a minor impact then the filtering of PANA traffic, since IAKERB [160] could be used to tunnel Kerberos traffic from the EU to the KDC through the AppS, overcoming the issue.

4.5 Conclusions

This chapter describes how an out-of-band protocol can be used to integrate Kerberos and AAA infrastructures in an organization, allowing EUs belonging to any member organization of the EAP/AAA federation to access the available application services in a seamless way, without requiring the introduction of any change to either the AAA or the Kerberos infrastructures. Hence, since there is no need to modify the Kerberos standard, the deployment of our proposal can be performed in a more straightforward way, by just including the new required entities (i.e. the PAA) in the involved organizations.

In this scenario, EU authentication is carried out by means of PANA and EAP. At the end of a successful authentication, the freshly generated Kerberos credentials (i.e. principal name and password) are bootstrapped and enforced in the KDC database, and distributed to the EU. Using the standard Kerberos protocol, the EU is then able to use those credentials to obtain Kerberos tickets from the KDC and to access the kerberized services offered. The SSO-based access among services located in the service provider is ensured by the standard behaviour of Kerberos.

Finally, this proposal can be extended to support the authorization model described in Chapter 3, with minor modifications (the SAML assertion introduced into the KDC during *Phase 2: PANA enforcement*, and included into the TGT during *Phase 3:Kerberos authentication*). This would allow the KDC to have a better control of the access to their services. However, it has the disadvantage of requiring some modifications to the KDC in order to include SAML processing capabilities.

Chapter 5

EduKERB: A cross-layer SSO solution for federating access to application services in the eduroam/DAMe network

5.1 Introduction

The proposals described in Chapter 3 and 4 assume the end user is already connected to the network, being therefore able to interact with the entities deployed within the service provider (KDC, PAA, AppS, etc.). Under this assumption, the two previous proposals provide means by which, after the execution of an EAP exchange, the end user is authenticated by the KDC. Moreover, the KDC may also have access to a SAML assertion containing authentication and authorization information about the end user, which can be used to retrieve additional identity information in form of attributes (e.g. nationality, age, role, etc.).

However, in some scenarios, the end user is not granted to access the network by default. Instead, she is required to perform an access control process before attaching to the network. This process can be also carried out in a federated way, allowing an end user, roaming from her *home organization*, to obtain network access to any other organization (*visited organization*) in the federation. The most successful example of federated network is eduroam, briefly described in Section 2.4.1.

In this scenario, a roaming end user would need to complete two different EAP authentications: one to be granted to access the network service (i.e. eduroam), and

the other to complete the federated authentication with the KDC (when using any of the proposals described in the previous chapters). This is also true for other initiatives for federated access to applications, such as Moonshot, where this second authentication would be performed directly with each application service, instead of with the KDC.

However, sometimes the visited organization also acts as the service provider. That is, it provides the network access service and the application service. Hence, in these cases, two EAP authentications would be performed as part of the separated access control processes (authentication and authorization) to access these services. As their purpose is to authenticate the same end user with the same organization, the latter one may be considered as redundant. Therefore, it would be desirable to have a mechanism that allows the end user to take advantage of the initial network authentication to enable the access to any other application service deployed in the service provider, without the need of any additional and expensive (in terms of time and network resources) authentication process. Such a mechanism is called in this thesis as *cross-layer SSO*.

Starting from this motivation, this chapter focuses on defining an architecture that provides a viable solution to this issue. In particular, this chapter aims to address the following objective, extracted from Section 1.3:

O3 To design a solution enabling the bootstrapping of Kerberos credentials as a result of the federated AAA-based authentication performed during the access to the network service, providing a *cross-layer* federated authentication for Kerberos that integrates the access to the network service with the access to upper-layer application services.

This objective can be broken down into several sub-objectives, described in Section 5.2.

The usage of the network authentication process as a means to bootstrap or distribute authentication tokens to the end user, with the purpose of accessing to upper-layer applications, is not new. It has been approached several times in the literature. For example, [122] defines a mechanism to provide the end user with a TGT valid for the service provider's domain after a successful network access control process based on EAP. However, this procedure not only violates the Kerberos specification (which states that TGTs must be provided to the client through a *KRB_AS_REQ/REP* exchange), but also requires the AS functionality to be integrated within the AAA server at the home organization. Moreover, this work also inflicts changes in the *EAP Key Management Framework* (EAP KMF) [65], since the MSK is directly inserted in the TGT. Within the ABFAB WG some effort was started in this direction [161], but it was finally abandoned as it did not provide any concrete proposal. Finally, DAMe (Section 2.4.2), which aims to

provide advanced authorization services to eduroam, also proposes the distribution of an authentication token to the end user, allowing her to request access to application services afterwards, thus achieving a cross-layer SSO solution. However, this mechanism was not standardized and it was defined only for web-applications as a proof of concept.

The chapter proposes an alternative solution that avoids the aforementioned drawbacks, called *EduKERB*. This solution is specific for the *eduroam* RADIUS infrastructure, although it does not preclude its utilisation in other scenarios if so is desired by the involved parties. EduKERB is based on the combined use of Kerberos and DAMe to allow organizations to federate their application services by just activating the support of Kerberos on their applications, without requiring further changes to them. In particular, this proposal takes advantage of the authentication token distributed by DAMe to bootstrap a security association between the end user and the KDC in the visited organization. Using it, the end user is able to obtain Kerberos tickets to access to the services deployed on that organization. Finally, authorization support can optionally be enabled following the model described in Chapter 3, being transparent to the application services.

The rest of this chapter is structured as follows. Section 5.2 describes the sub-objectives and architectural requirements of the solution. Section 5.3 presents the proposed architecture. Section 5.4 describes how the defined entities interact with each other to accomplish the required functionality. Section 5.5 provides a security analysis of the proposed solution. Finally, Section 5.6 extracts some conclusions.

5.2 Objectives and requirements

To accomplish the high level objectives of this chapter, we propose that service providers within eduroam deploy Kerberos for controlling the access to their services. By using a token obtained after a successful network access authentication (*eduToken*), the end user can obtain a valid TGT from the KDC. Although the eduToken is originally defined in [115], for the purpose of this Chapter we have re-defined it to adapt it to the SAMLv2 notation, resulting into an assertion with the same format and contents as the one used in Chapters 3 and 4, and exemplified in Appendix B. Afterwards, by making use of the standard Kerberos SSO operation, the end user can make use of this TGT to obtain STs to access the existing services within the service provider. Furthermore, the solution will also enable end user authorization when accessing to application services. More precisely, the solution described in this chapter aims to accomplish the following sub-objectives:

- Definition of a federated access to the different services available at the *eduroam* organizations, regardless they are web-based or not.
- Provision of access control to application services based on Kerberos. The solution will support a federated access by using existing and fully operative AAA infrastructures (as it happens in *eduroam*).
- Provision of a cross-layer SSO to end users. The solution will take advantage of the SSO capabilities offered by Kerberos, which allows a user to access several application services through a single authentication with the KDC. Furthermore, the Kerberos authentication is bound with the authentication performed during the access to the *eduroam* service.
- Definition of an authorization process during application service access, based on the infrastructure already deployed for DAMe. Thus, the final decision on whether the end user will be granted to access to a specific application service (i.e. she is provided with a ST) will be determined not only by her identifier, but also by her set of attributes which are accessible in the service provider. Hence, the decision can be taken based on fine-grained information.

To accomplish the aforementioned objectives, the design of the architecture has taken into consideration the following general requirements:

- The solution should favour an easy deployment and integration with the existing or already defined components. This implies the re-utilisation of the entities already described in the eduroam/DAMe architecture (e.g. identity and policy management entities), minimizing the number of new elements and protocols. In particular, the modifications proposed in this solution will only affect to the home and service providers.
- The solution should be as much transparent as possible to the end user. That is, the end user should not be aware of the details of the process, which will not differ much from the operation performed by the end user during network access authentication. After that, the access to the services should be performed without requiring the end user to provide further long-term credentials to her home organization for verification (i.e. application of SSO mechanism).
- The application services that are deployed in the service provider will not require any update of existing software. Their only requirement will be to support a

Kerberos-based authentication such as *GSS-API Kerberos V5* or *Kerberos V5 SASL*. We believe this is a reasonable assumption taking into account that, nowadays, many application services already support these mechanisms.

Besides these general requirements, security has a paramount importance in the design of an architecture. The following general security requirements have also guided the definition of the architecture:

- Authentication of end users with the service provider's KDC, in such a way that the KDC can know that the end user trying to obtain a ticket is who she claims to be, and not anyone else. The authentication is performed by means of a combined use of the *eduToken*, and the cryptographic material resulting from the EAP process performed during the network access. The KDC does not need to have any pre-established state or information about the roaming end user.

- Protected distribution of sensitive information. As part of the proposal, sensitive information will be securely distributed between parties (e.g. identity information and keying material). This process is susceptible to a variety of attacks, ranging from *eavesdropping* to *tampering* [162]. Therefore, it will be performed through protected channels, providing confidentiality, integrity protection, authenticity and replay protection.

- Secure derivation of the cryptographic material used to protect the sensitive information. In this proposal, many of the keys are derived from a common root key resulting from the network authentication. It is thus required that the cryptographic strength of the derivation process avoids the compromise of any of these keys degrades the security of the other keys (derived or root), as described in [66].

- Protection of the end user's identifiers through *pseudonymity*. Only the entities in the home organization will know the real end user's identifier. Entities in the service provider will instead be provided with pseudonyms, so that the solution provides a basic privacy protection.

5.3 Proposed architecture

Most of the components required for the proposed architecture are already present in *eduroam*, though some of them will include additional functionality to accomplish the

objective of this proposal. In the following, an enumeration of these components and a brief description of their functionality are presented. Figure 5.1 shows the components of the architecture and the protocols that allow communication between each pair of entities.

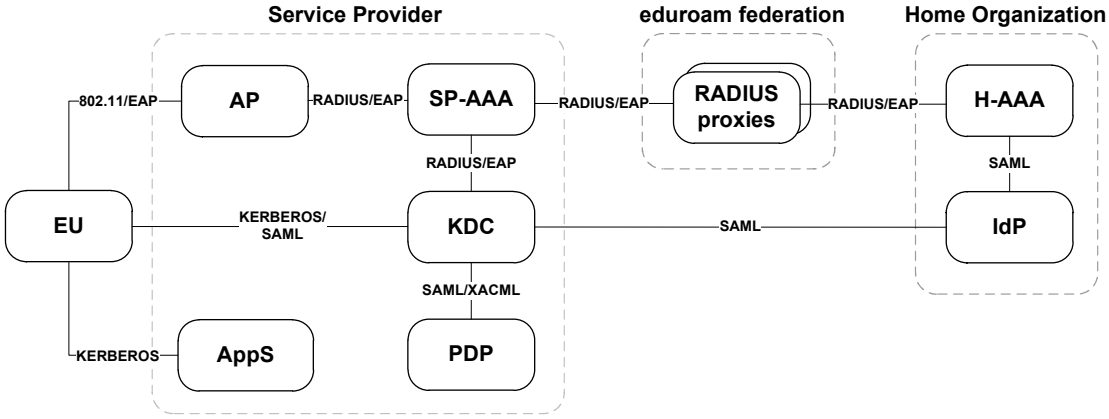


Figure 5.1: EduKERB architecture.

- *End user* (EU). This component represents an end user that first authenticates to access the network service provided by the service provider using eduroam/DAMe and, after that, desires to access a application service deployed on that service provider. To do so, the end user makes use of the *eduToken* received during network authentication, along with a key derived from the EAP authentication with her home RADIUS server. With this information, the end user can access the KDC to obtain a TGT. By following the standard Kerberos operation, the end user can use this TGT to request additional STs as she wants to access the different services under the control of the service provider.

- *Home organization's RADIUS server* (H-AAA). This component is already present in the eduroam architecture. It integrates the *EAP Server* functionality. Besides, it also communicates with the *IdP* to obtain the *eduToken* related with the end user. For this proposal, this component is also required to generate a special key derived from the EMSK, named *Domain-Specific Root Key* (DSRK) [66], to honour key distribution requests coming from the service provider's RADIUS server after a successful authentication of the end user.

- *Identity Provider* (IdP). This component, defined in the DAMe architecture, is responsible for providing the *eduToken* to the H-AAA after a successful network

access authentication. The IdP uses SAML to build the *eduToken*. It plays the same rol as in FedKERB (Chapter 3) and PanaKERB (Chapter 4).

- *Service provider's RADIUS server* (SP-AAA). It is a component already deployed in the eduroam architecture. During the network access process, it receives EAP packets from the *Access Point* (AP) using RADIUS. Based on the end user's identifier, it is able to forward the EAP packets to the H-AAA. With the DSRK, the SP-AAA can derive a so-called *Domain-Specific Usage-Specific Root Keys* (DSUSRKs) for the KDC. This key will be used as a shared secret between the EU and the KDC. The analysis of this derivation is discussed in Section 5.5.3. Note that the MSK cannot be used for this purpose, as it is derived exclusively to be used between the EU and the AP.

- *Access Point* (AP). This component is present on eduroam, and it acts as the point of attachment to the service provider's network.

- *KDC*. Once the end user gains network connectivity at the service provider and wants to access an application service, it has to contact with the KDC. The end user authentication with the KDC is performed by means of a new Kerberos pre-authentication mechanism, which uses the *eduToken* that the end user obtained during network access and retrieves the specific *DSUSRK* from the SP-AAA. Thanks to the DSUSRK, the KDC can derive the *reply key* (used to protect the *KRB_AS_REP* message). Additionally, by using the information in the *edutoken*, the KDC can use the authorization model described for FedKERB in Section 3.4.2.

- *Policy Decision Point* (PDP). It manages the access control policy set of the service provider. Its functionality is identical to the one already described in Chapter 3.

- *Application service* (AppS). It provides the specific application service requested by the end user (e.g. SSH, web service, etc.). Its functionality is identical to the one already described in Chapter 3.

These components interact among them to accomplish the functionality defined in this chapter. The next section provides a detailed description of the operation, as well as the details of the information exchanged during the process.

5.4 General operation

This section describes the general operation of the solution when an EU, which roams to a service provider and remains there during a period of time, desires to access an AppS located on that organization. The whole process consists of three main phases:

- *Phase 1.* Access to the network: authentication, *eduToken* distribution and keying material.
- *Phase 2.* Kerberos pre-authentication and TGT acquisition.
- *Phase 3.* ST acquisition and access to the application service.

All these phases must be performed before being able to access an application service. However, once the end user has obtained the TGT (phase 2), thanks to the SSO capabilities provided by Kerberos, only phase 3 needs to be performed for each access to other application services deployed in the same service provider. This reduces the interactions required between the end user and the infrastructure in order to perform the service access control. Indeed, the only phase where the end user has to provide her long-term credentials is during network access authentication (phase 1), which is performed once per service provider and network access session.

5.4.1 Notation

This section describes the flow of information that happens between the components when performing each one of the steps listed above. The diagrams show a simplified representation of the information that would actually be transmitted "on the wire", depicting only the most relevant one. For a more complete and formal description of the exchange the reader might refer to the Appendix C.

The notation provided in Table 5.1 is used in the next subsections to describe the information exchanges that are required between the involved parties. This contribution requires a more detailed notation than the previous ones, since it is more complex from a security standpoint, as it involves the exchange of several keys and sensitive information between different layers and components.

Table 5.1: Notation to describe the exchanges.

Notation	Description
E	End user.
A	Access Point.
S	SP's RADIUS server.
H	Home RADIUS server.
I	Identity Provider.
K	Key Distribution Center.
P	Policy Decision Point.
$\{data\}_{key}$	*data* is encrypted using symmetric key *key*.
$[data]_{key}$	*data* is authenticated and integrity protected using symmetric key *key*.
$\{data\}_{E}$	*data* is encrypted using E's public key.
$\{data\}_{E^{-1}}$	*data* is signed using E's private key.
$radius\_XY$	RADIUS pre-shared key established between entities X and Y.
mac_XY	Pre-shared key established between entities X and Y to generate *Message-Authentication-Code* RADIUS attributes [163] used to authenticate RADIUS packets.
$keymat\_XY$	Pre-shared key established between entities X and Y to encrypt *Keying-Material* RADIUS attributes [163] used to encrypt keying material.
$tunnel\_key$	Key established between the end user and the H-AAA as result of the tunnelled EAP authentication.
key_KDC	Pre-shared key established between the AS and the TGS within the KDC.
ts	Time-stamp.

5.4.2 Phase 1: Access to the network: authentication, distribution of the eduToken and keying material

Before accessing any federated application service, the end user must be authenticated at network layer by the service provider, with the support of the eduroam AAA infrastructure. The process is depicted in Figure 5.2. The authentication is performed by means of 802.11 and EAP (more specifically, using a tunnelled EAP method), and involves the existence of a federation of RADIUS servers that allow EAP packets to be transferred from the wireless AP (Access Point) in the service provider to the H-AAA, and vice-versa. In this way, the end user authenticates herself by means of the EAP credentials shared with her home organization. RADIUS packets are integrity protected hop-by-hop, using shared keys (named *radius_XY* in Figure 5.2) established between each pair of RADIUS servers in the path from the AP to the H-AAA. To provide privacy, the end user sets *anonymous@home* (where *home* is the home organization's domain name) as the identity for the EAP authentication (2), while her real identifier is sent in the inner protected tunnel constituted during the execution of the tunnelled EAP method. At the end of the authentication process, both the H-AAA and the end user derive two keys: MSK and EMSK. The MSK is sent by the H-AAA to the AP through the RADIUS transport as usual (13 and 14). This key is used by the AP and EU to protect data traffic, as described in Section 2.1.3.

Additionally, DAMe extends this network authentication process by introducing authorization management of the end user, as described in Section 2.4.2. More specifically, after a successful network access authentication, and before sending the *RADIUS Access-Accept* message, the H-AAA contacts the IdP of the home organization to obtain the *eduToken* associated to the end user (9). To do so, the H-AAA sends a SAML *AuthnRequest* message to the IdP, indicating the real identifier of the end user. As a response, the H-AAA obtains a *SAML Assertion* (called *eduToken* [115]), whose *Subject* field is a *pseudonym*, as a means for identification while protecting end user's privacy (10). The H-AAA forwards this *eduToken* to the end user (11), within the protected TLS tunnel established during the execution of the EAP method. Additionally, the H-AAA sends the value of the *pseudonym* to the SP-AAA (13), which may use it to retrieve identity information about the end user in future interactions (e.g. network access authorization). Section 5.5.2 analyses the security of the distribution of the *eduToken*.

Everything explained to this point has already been defined by *eduroam* and *DAMe*. Our proposal starts from here, and extends the eduroam/DAMe exchanges to allow the SP-AAA to request and receive a DSRK for the service provider. With this DSRK, the

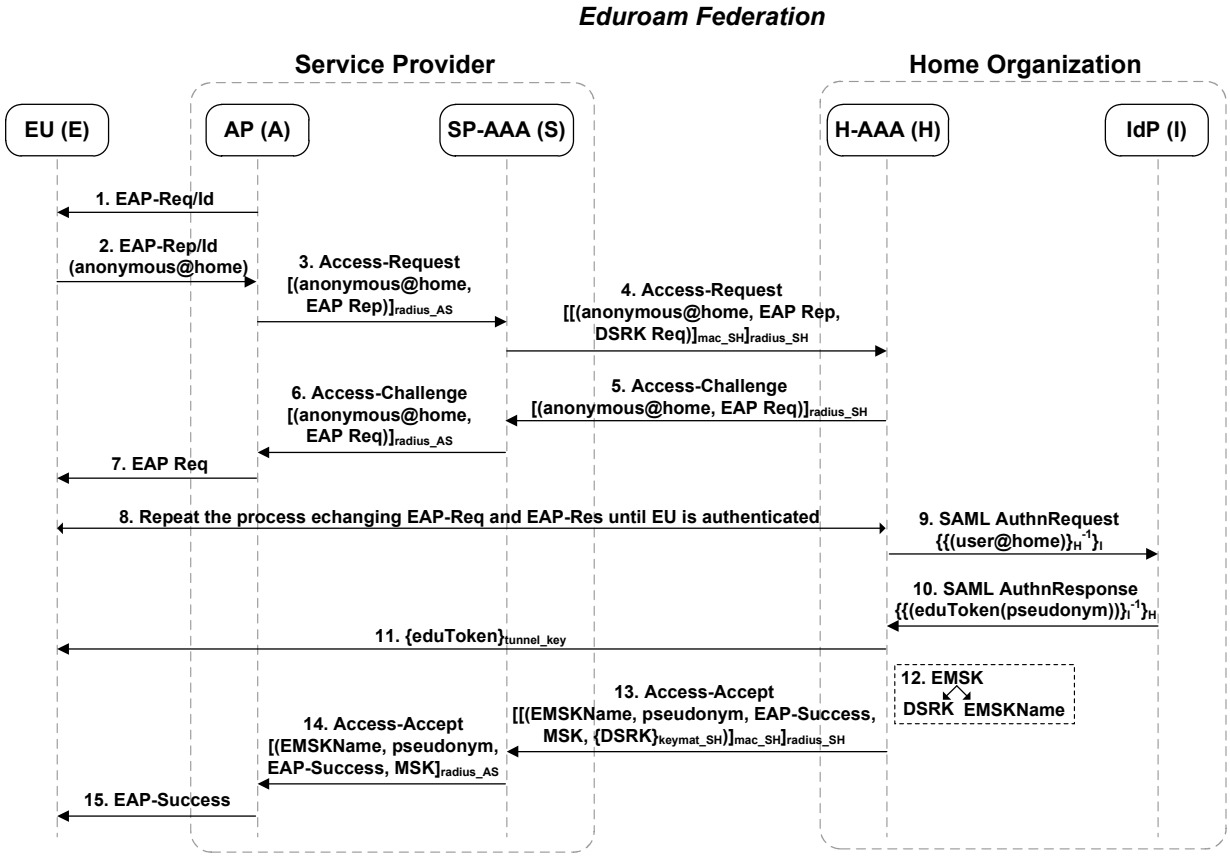


Figure 5.2: Phase 1: Access to the network: authentication, distribution of the eduToken and keying material.

SP-AAA derives and delivers a DSUSRK for the KDC upon request. This key is used by the KDC to derive a fresh Kerberos *reply key* for the EU.

The DSRK distribution is initiated by the SP-AAA by including a set of additional RADIUS attributes in every *Access-Request* message sent to the H-AAA (4). The inclusion of these attributes allow soliciting the distribution of keying material derived from the keys generated during a previous EAP authentication process. When the H-AAA successfully authenticates the end user, in addition to the derivation of the *MSK* and *EMSK*, two operations are performed. On the one hand, a name associated to the EMSK is generated (referred to as *EMSKName*). On the other hand, a *DSRK* for the service provider is derived from the EMSK (12). Note that the key hierarchy is expressed with arrows going from the root key to the derived key.

The DSRK is sent to the SP-AAA in the *Access-Accept* packet, using the attributes described in [163]. The keys used to protect these packets are *keymat_XY* (confidentiality)

and *mac_XY* (integrity and authenticity) (13). Conversely, the SP-AAA is informed about the *EMSKName* by using the standard *User-Name* [10] RADIUS attribute (13). The SP-AAA stores this *DSRK* associated to the value of the *EMSKName*.

At the end of the process, the end user has been authenticated to access the network; she has obtained an *eduToken* in a protected way (through the TLS channel established by tunnelled EAP method), and she has derived a DSRK (from its EMSK), which also shares with the SP-AAA. Additionally, the SP-AAA has obtained an *EMSKName* and a *DSRK* key associated to the end user. The *eduToken* must be stored by the end user in a secure way (e.g. within an encrypted and integrity protected file) in the end user's device, since it is a sensitive piece of information that may be used to impersonate the end user if revealed to unauthorized parties.

5.4.3 Phase 2: Kerberos pre-authentication and TGT acquisition

The objective of this phase is to perform a pre-authentication process between the EU and the KDC to obtain a TGT. This process is based on the use of the *eduToken* and the DSRK distributed to the EU and the SP-AAA, respectively, during phase 1.

As studied in Chapter 3, to accomplish this objective, Kerberos allows the exchange of pre-authentication data (*padata*) as a way to extend the authentication process. Moreover, as already described in Chapter 3, [103] defines a pre-authentication framework which specifies how new pre-authentication mechanisms should be defined, indicating the most important security and functional requirements to be satisfied. Additionally, [103] also defines the *Flexible Authentication Secure Tunnelling* (FAST) pre-authentication mechanism, as a tool to establish a protected channel between the end user and the KDC for pre-authentication. The pre-authentication data elements carried within FAST are called *FAST factors*. Hence, our proposal defines a new FAST factor, called *PA-EDUTOKEN*, which transports the *eduToken* and the *EMSKName* from the end user to the KDC, providing the required security properties (Section 5.5.2). Furthermore, the *PA-EDUTOKEN* contains a timestamp which is integrity protected with the *reply key*.

After the network access, the end user learns the Kerberos realm where the service belongs to and the location of its respective KDC (e.g. through DNS [7] or DHCP [164]). With this information she can start the process of pre-authentication with the KDC (see Figure 5.3). According to FAST operation, the end user needs to obtain an *armor TGT* before starting the exchange of pre-authentication data with the KDC. This *armor TGT* contains a so-called *armor key* that constitutes a shared secret between the end user and the KDC used to build the protected FAST tunnel to encapsulate the rest of Kerberos

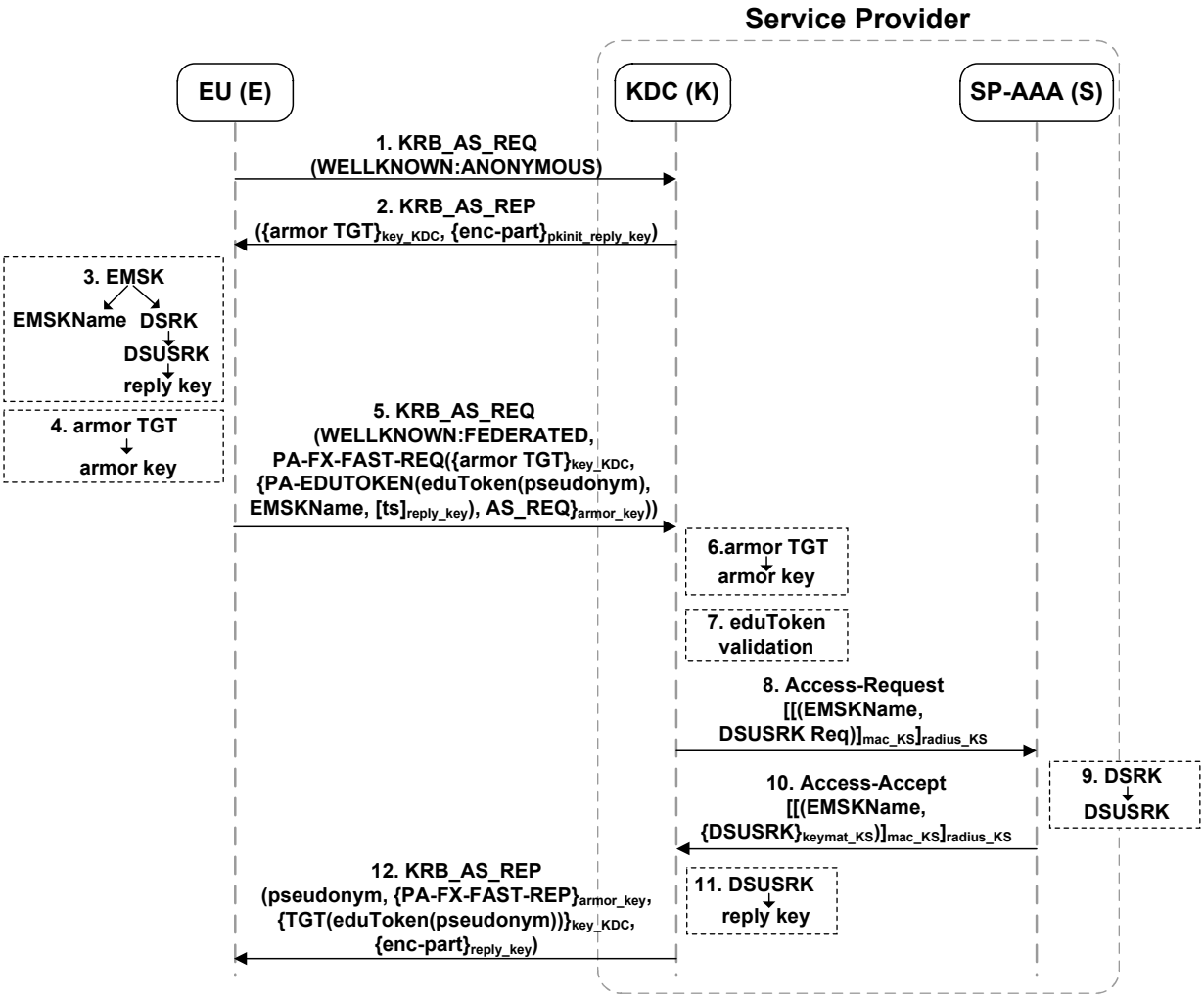


Figure 5.3: Phase 2: Kerberos pre-authentication and TGT acquisition.

exchanges. To obtain this armor TGT, the end user can perform an anonymous PKINIT process [165] with the KDC (1 and 2), where the end user does not authenticate with the KDC, but the KDC is authenticated by means of certificates distributed by *eduPKI* [116], already deployed in eduroam/DAMe. To validate the KDC's certificate, the end user makes use of the CA's certificate, which is pre-configured amongst all the members of the eduroam federation for this purpose.

Once the *armor TGT* is obtained, the end user derives the *DSRK* for the service provider using the *EMSK* produced during the EAP network access authentication as the root key. It also derives the *EMSKName* and, from the *DSRK*, it derives a *DSUSRK* for the specific KDC. Finally, the end user derives the *reply key* from the DSUSRK, completing the

key hierarchy for this proposal (3). The key derivation process is described in Section 5.5.3.

With this information the end user generates a KRB_AS_REQ message, including the *PA-EDUTOKEN*. This whole message is encrypted with the *armor key* and encapsulated within a *PA-FX-FAST-REQUEST padata*, as described in [103]. This *padata* is then included into a KRB_AS_REQ message, and sent to the KDC (5). This encapsulation is depicted in Figure 5.4. In addition to the field *padata*, the figure shows other fields of the Kerberos messages, such as *pvno*, or *msg-type*. They are shown with the purpose of placing *padata* in context. These are not further explained in this chapter as they are no relevant for this proposal. More information about the complete Kerberos message format can be found in [7].

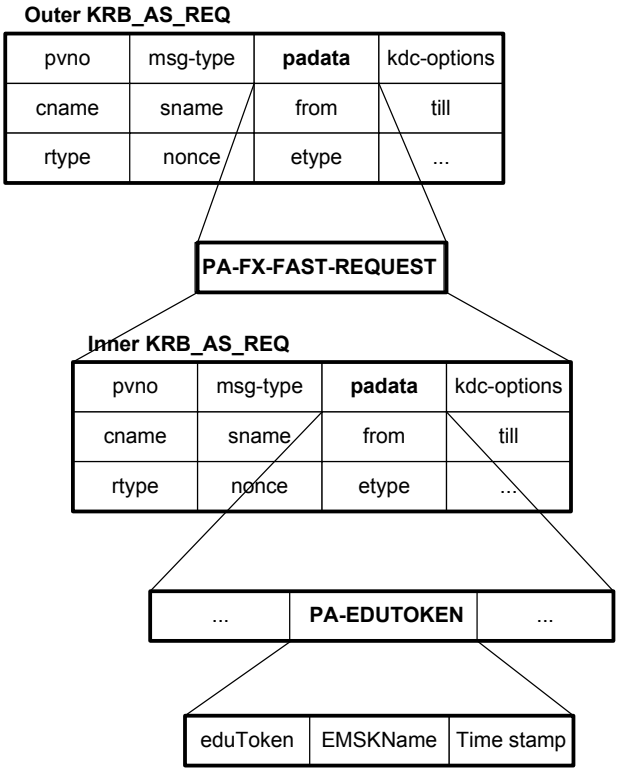


Figure 5.4: Encapsulation of PA-EDUTOKEN as a FAST factor.

It is important to note that standard Kerberos specifies that the KDC must generate an error if the Kerberos client identifier specified in the *KRB_AS_REQ* message is not found in the local database of the KDC. Since one of the objectives of this proposal is indeed to allow access to federated users not pre-registered in the KDC, the WELLKNOWN/FEDERATED end user's identifier defined in Chapter 3, Section 3.5.1 is also used here. Therefore, the KDC is notified that the EU is not in the local database, and

that the pre-authentication mechanism will be in charge of verifying the actual identity and deriving a *reply key*.

When the KDC receives the FAST request (5), it makes use of the *armor key* to decrypt the content of the *PA-FX-FAST-REQUEST padata*, recovering the *KRB_AS_REQ* message and the *PA-EDUTOKEN*. The KDC validates the signature and syntax of the *eduToken*, and performs other verifications, such as whether the *Issuer* is trusted or, the token validity period, etc.

In the next step, the KDC retrieves a *DSUSRK* from the SP-AAA. Similarly to the DSRK key distribution in phase 1, the DSUSRK distribution is carried out by using the guidelines defined in [163] for the delivery of key material. More precisely, the KDC sends an *Access-Request* message to the SP-AAA to request the distribution of the DSUSRK. This message contains the *Keying-Material, MAC-Randomizer, and Message-Authentication-Code* attributes, defined in [163]. Additionally, the RADIUS *User-Name* attribute is set to the value of the *EMSKName* obtained from the *PA-EDUTOKEN*, which allows to identify the end user (step 8). After that, the SP-AAA processes the *Access-Request* message and derives a *DSUSRK* for the KDC, using the *DSRK* associated to the *EMSKName*. The *DSUSRK* is sent to the KDC in an *Access-Accept* message by using the RADIUS attributes defined in [163]. Then, this key is used by the KDC to derive the *reply key* necessary to verify the timestamp included into the *PA-EDUTOKEN* and to protect the information contained in the KRB_AS_REP message.

Finally, the KDC includes the received *eduToken* within the generated TGT, with the purpose of receiving it again during the KRB_TGS exchange to perform authorization functions. A new *authorization-data element* needs to be defined to represent the *eduToken*. In this way, the KDC will be able to perform authorization decisions before issuing any *ST*. The KRB_AS_REP message is sent to the end user as the final step of the authentication exchange (12). It is worth mentioning that the end user's identifier specified in both the KRB_AS_REP and the TGT, is set to the value of the *pseudonym* extracted from the *eduToken* (Section 5.5.4).

After this step, the end user can request individual *STs* for the different application services offered by the service provider.

5.4.4 Phase 3: ST acquisition and access to the application service

Once the end user has received the TGT from the AS, she can request a ST for the specific AppS she wants to access to, by executing a standard KRB_TGS exchange with the TGS. Prior to the issue of the ST, the TGS performs an authorization process using the information contained in the eduToken (SAML Assertion) transported on the TGT. This process is identical to the authorization process described in Chapter 3, Section 3.4.2.

The following Figure 5.5 extends the one provided in Chapter 3 (Figure 3.4.2), by including details on how the transmission of data is protected. Note that, in this case, the MDS is not present for sake of simplicity. Hence, the KDC is assumed to be able to locate the IdP based on the information present in the eduToken.

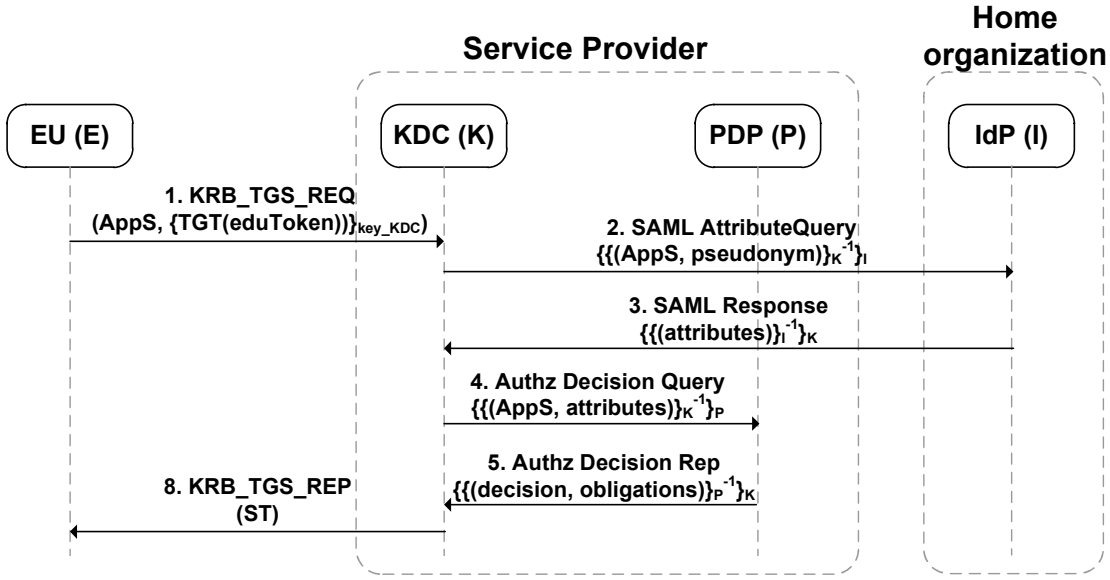


Figure 5.5: Phase 3: ST acquisition and access to the application service.

In particular, the TGT in the KRB_TGS_REQ message (1) is protected by means of the key_KDC, which is only know to the KDC. The SAML AttributeQuery (2) is signed by the KDC using the private key K^{-1}, and encrypted for the IdP using the public key I. Conversely, the subsequent SAML Response (3) is signed by the IdP using the private key I^{-1}, and encrypted for the KDC using the public key K. A similar process is followed for the SAML/XACML Authorization Decision Query/Response, between the KDC and the PDP

After the EU receives a valid ST, she performs standard KRB_AP_REQ/REP exchange

with the target AppS to access the service (Section 2.3.1).

5.5 Security Analysis

This section describes how the security requirements indicated in Section 5.2 have been achieved by the current proposal.

5.5.1 End User Authentication

Network Access Authentication

In order to achieve a cross-layer SSO solution, this proposal uses the initial network access authentication based on EAP, to distribute an *eduToken*, and key material afterwards, to securely enable access to application services in a SSO-fashion using Kerberos. That is, without performing lengthy additional EAP authentications. Thus, security of this solution stems from this initial step.

In eduroam, two EAP methods are mandatory to deploy for network access authentication: PEAP and EAP-TTLS. Both methods establish a TLS channel between the end user and the H-AAA providing confidentiality, integrity and anti-replay protection. The authenticity of the server is assured thanks to the existing PKI. The inner authentication method is used by the end user to demonstrate the knowledge of the long-term credential associated to the claimed identity. Nevertheless, as described in [166], EAP-TTLS may present some security problems. For example, it does not establish any cryptographic binding between the EAP-TTLS and the inner EAP method, which may lead to a man-in-the-middle attack [167]. This attack is possible since there is no way to link that the same entity that performed the outer EAP method (EAP-TTLS) also performed the inner one. Moreover, PEAP is also vulnerable when the inner EAP method does not generate keying material. With this condition, the attacker can execute the tunnelled method with the EAP server, as it only requires server-side authentication. Once the EAP server starts the execution of the inner non-key generating cryptographic method (e.g. based on passwords), the attacker forwards the EAP packets to a valid end user. This end user would think the EAP server is requesting the execution of the inner method without the TLS tunnel, and will probably accept the conversation, replying to the presented challenge using her long-term credentials. The attacker would introduce this reply into the TLS tunnel, making the H-AAA think it is indeed the legitimate end user. This type of attack can be mitigated by prohibiting the use of these non-key generating methods (e.g.

CHAP, PAP, MD5...) out of the TLS tunnel established by EAP-TTLS and PEAP, but it requires that the EU's supplicant software supports configuring this.

Moreover, after a successful authentication with the selected EAP method, an EMSK is exported. This key is assumed to have strong cryptographic properties in terms of pseudo-randomness and key separation (otherwise these EAP methods should be simply discarded to protect any network). Moreover, this EMSK is never exported out the end user (EAP peer) or the H-AAA (EAP server). Due to this, the EMSK is used as root key for the key hierarchy that our solution requires. Section 5.5.3 analyses the security of this key derivation process.

Application Service Access Authentication

With the *eduToken* and the *reply key*, the KDC can prove the end user was already authenticated during network access authentication and is authorized to obtain a TGT from the KDC. Indeed, the end user demonstrates the knowledge of the *reply key*, derived from the EMSK resulting from the EAP authentication which, in turn, involves the end user's long-term credentials. To prove this knowledge to the KDC, the end user includes a time-stamp in the pre-authentication data PA-EDUTOKEN, which is integrity protected with the *reply-key*. The PA-EDUTOKEN, which also contains the *eduToken* is, in turn, securely transported within FAST tunnel. The establishment of FAST provides confidentiality, integrity protection and authenticity of the KDC. In particular, since the armor key used to build the tunnel is based on the armor TGT obtained after an anonymous PKINIT process, the end user can authenticate the KDC by means of the PKI (i.e. eduPKI).

Once the end user obtains the TGT, she can request service tickets (STs) for accessing different services following the standard way of operation in Kerberos (Section 2.3.1). Thus, under security standpoint, the security of standard Kerberos remains unaltered during this process.

5.5.2 Distribution of the eduToken

The *eduToken* may contain sensitive authorization information that should not be disclosed to, or modified by, unauthorized parties (e.g. identifiers, expiration dates, attributes, etc.). Hence, its transport must provide confidentiality, integrity and authenticity, as one of the security requirements of the architecture is to protect the distribution sensitive information (Section 5.2).

The *eduToken* is distributed in three different moments. First, it is sent from the H-AAA to the end user during the initial network authentication, whose security properties have been discussed in 5.5.1. The security requirements of the transport are satisfied by the tunnelled EAP method, where the *eduToken* is delivered to the end user within the protected TLS tunnel established during the execution of the EAP method. Second, it is delivered from the end user to the KDC during the Kerberos pre-authentication. In this step, FAST is used to establish a secure channel providing integrity and confidentiality for the transport of the *PA-EDUTOKEN* (see 5.4.3), so providing the required security properties. Finally, the KDC includes the *eduToken* within the TGT provided to the end user, which is protected with confidentiality, authenticity and integrity with the TGS key, as defined in the standard Kerberos.

5.5.3 Key derivation and distribution

As visiting end users belong to a different organization, they are not expected to be registered into the KDC's database of the service provider. Thus, the *reply key* to be used to encrypt the *enc-part* of the KRB_AS_REP message must be dynamically generated for each visiting user during the Kerberos pre-authentication. This requirement has been addressed by deriving the *reply key* from a dynamically generated shared key between the end user and the KDC: the DSUSRK. The DSUSRK is derived, following a key hierarchy, from the EMSK, which is generated after a successful authentication by some EAP methods (e.g. PEAP). The EMSK is assumed secure in terms of pseudo-randomness and key separation. For the key derivation, the general key derivation framework in [66] has been followed, which uses a *Key Derivation Function* (KDF) as follows:

$$\text{derivedkey} = \text{KDF}(\text{root key, key label, optional data, length}).$$

This derivation includes a *root key*, a *key label*, *optional data* and output *length*. By default, this KDF is taken from the *Pseudo Random Function+* (PRF+) key expansion defined in IKEv2, being HMAC-SHA-256 [168] the default PRF. According to the security analysis in [169], this kind of PRF-based key derivation procedure ensures that the key strength of the derived key is at least as good of the root key used to derive it. Hence, the requirement of secure key derivation is accomplished (Section 5.2)

Moreover, using a dynamically generated DSUSRK as root key for the derivation of the *reply key*, and limiting the TGT lifetime to the validity of the received *eduToken*, the impact of dictionary attacks is minimized in comparison with standard Kerberos, where the *reply key* is derived there directly from a long-term password. The reason is threefold. First,

because the dynamically generated one is derived from a binary blob, and thus unlikely to be on a dictionary of most common used words of any language. Second, because even if an attacker is able to reveal the reply key using a pure brute force attack (i.e. all binary combinations), it will be very unlikely do it before the key expires and, therefore, it will not be able to obtain tickets for its own benefit. And third, because decrypting the key will not allow the attacker to have access to a small portion of all end-users conversations.

Besides, the derived keys DSRK and DSUSRK must be transported to the SP-AAA and to the KDC, respectively, by authenticated and authorized parties, assuring integrity and confidentiality of the keys. This is addressed following the distribution schema described in [170], and using the attributes *Keying-Material, MAC-Randomizer, and Message-Authentication-Code*, defined in [163]. Indeed, using these attributes, information is protected hop-by-hop by the use of pre-established shared keys between RADIUS servers. Integrity, authenticity, and encryption are assured by the use of the keys named as *mac_XY* and *keymat_XY*, defined in 5.1.

It is worth noting that, though these security properties are provided hop-by-hop (so that any RADIUS server in the path between the home and service providers would be able to access those keys), the RADIUS servers that conform the federation are considered trusted entities that will treat the information with the required security considerations.

5.5.4 Pseudonymity

Identifiers are required in most of the exchanges performed in this proposal to recognize the principal interacting with the entities. As the protection of the end user's identifier is one of the security requirements of the architecture (Section 5.2), the entities in the service provider are provided with pseudonyms while the real end user identifier is only known by the trusted entities in the home organization.

For the EAP authentication, the use of a tunnelled EAP method such as PEAP allows hiding the end user's NAI to the intermediate RADIUS servers between the end user and the H-AAA. Indeed, the real end user's identifier is transmitted in the inner EAP method protected within a TLS tunnel. At the end of the EAP authentication, the *EMSKName* derived from the EMSK is used as pseudonym in the RADIUS *User-Name* attribute. This value identifies the end user for the key derivation and distribution process.

Besides, the *eduToken* provided by the IdP also contains a pseudonym. This pseudonym is used by the KDC to retrieve further end user's attributes from the home IdP, and also as the user identifier in the generated tickets.

The co-existence of these two pseudonyms is required as they are generated by two

different entities, the H-AAA and the IdP, and have different purposes (identifying the end user for keying material distribution and identifying the end user for attribute retrieval, respectively).

5.5.5 Formal verification

Due to the complexity of the solution in terms of the exchange of keys and sensitive information across layers and components, we have performed a formal verification of the EduKERB proposal. In order to formally verify the accomplishment of the security properties described in subsections 5.5.1 to 5.5.4, a security analysis tool has been used. In general, this kind of tool performs an automatic search of security flaws in network protocols, by simulating the behaviour of honest entities along with the existence of malicious agents, which try to break the security properties of the system. Hence, they are useful to validate the phases integrating the proposed solution, and which are defined in Section 5.4.

In particular, the AVISPA tool v1.1 [171] has been chosen. This tool has been extensively and successfully used in the literature (e.g. [172, 173]), specially to verify standard protocols. AVISPA allows verifying the satisfaction of a set of security goals for protocols defined by means of the *High Level Protocol Specification Language* (HLPSL) [174]. This language is based on the definition of a set of roles, which are played by agents. For each role, the initial knowledge is provided, and the expected behaviour is specified by using finite state machines. Input is represented as a received message, while output is considered a sent message. AVISPA defines a special agent called *intruder*, which tries to perform different attacks. This intruder follows the Dolev-Yao model [175], where the attacker can inject, overhear and intercept messages between two entities, although it is unable to break the underlying cryptographic operations. Although AVISPA potentially allows other type of models (not implemented yet), Dolev-Yao is one of the most general and powerful ones [176, 177].

Instead of starting the HLPSL specification of this proposal from scratch, it has been preferred to use the library of protocol specifications provided by AVISPA as starting point. It is assumed that these protocol specifications are well modelled and strongly reviewed by the AVISPA community, so they can be taken a solid base to include the extensions proposed in this work. Namely, two existing specifications have been used: *EAP_TTLS_CHAP.hlpsl*[1] to model the phase 1 (defined in Section 5.4.2), and

[1] http://www.avispa-project.org/library/EAP_TTLS_CHAP.html

Goal	Description	Related subsection
authentication_on np	SP-AAA authenticates the EU	5.5.1
authentication_on ns	EU authenticates the SP-AAA	5.5.1
authentication_on t0	KDC (AS) authenticates the EU	5.5.1
authentication_on t1	KDC (TGS) authenticates the EU	5.5.1
authentication_on t2a	AppS authenticates the EU	5.5.1
authentication_on n1	EU authenticates the KDC (AS)	5.5.1
authentication_on n2	EU authenticates the KDC (TGS)	5.5.1
authentication_on t2b	EU authenticates the AppS	5.5.1
secrecy_of edutoken	Secrecy of the eduToken	5.5.2, 5.5.4
secrecy_of sec_clientK	Secrecy of the TLS key for EU encryption	5.5.3
secrecy_of sec_serverK	Secrecy of the TLS key for AppS encryption	5.5.3
secrecy_of sec_emsk	Secrecy of the EMSK	5.5.3
secrecy_of sec_dsrk	Secrecy of the DSRK	5.5.3
secrecy_of sec_dsusrk	Secrecy of the DSUSRK	5.5.3
secrecy_of sec_reply_key	Secrecy of the Kerberos reply key	5.5.3
secrecy_of sec_Karmor	Secrecy of the Kerberos armor key	5.5.3
secrecy_of sec_a_Kcg, sec_t_Kcg, sec_c_Kcg	Secrecy of the TGS session key	5.5.3
secrecy_of sec_t_Kcs, sec_s_Kcs, sec_c_Kcs	Secrecy of the service session key	5.5.3
secrecy_of sec_uname	Secrecy of the user name for EAP authentication	5.5.4
secrecy_of sec_Emskname	Secrecy of the EMSKName	5.5.4

Table 5.2: Security goals.

Combination	OFMC	Cl-Atse
Single session	SAFE	SAFE
Two sessions	SAFE	SAFE*
Two sessions (intruder=EU)	SAFE	SAFE
Two sessions (intruder=AP)	SAFE	SAFE

Table 5.3: Phase 1 analysis results.

Kerberos-preauth.hlpsl[2] to model phases 2, 3 and 4 (which are defined in sections 5.4.3, 5.4.4 and 5.4.4, respectively). The complete HLPSL specification for this proposal can be found in Appendix D.

AVISPA allows the verification of two kind of security goals: *secrecy* and *authentication*. A *secrecy goal* indicates that a specific piece of information must not be accessible for any agent other than those explicitly enumerated. An *authentication goal* indicates that a specific agent (e.g. *Agent*1) must be authenticated against another agent (e.g. *Agent*2), based on the specified information. Table 5.2 indicates the specific security goals that have been defined in the HLPSL specification of this proposal (see Appendix D to verify the security requirements described in previous Sections 5.5.1 to 5.5.4).

AVISPA provides four model checkers to validate the HLPSL specification: OFMC,

[2]http://www.avispa-project.org/library/Kerb-preauth.html

Combination	OFMC	Cl-Atse
Single session	SAFE	SAFE
Two sessions	SAFE	SAFE*
Two sessions (intruder=EU)	SAFE	SAFE
Two sessions (intruder=AppS)	SAFE	SAFE*

Table 5.4: Phase 2, 3 and 4 analysis results.

Cl-Atse, SATMC and TA4SP. According to [171], the Cl-Atse model checker has shown better properties. Nevertheless, EduKERB's specification has been checked using all of them, in order to provide a complete security analysis. However, checkers SATMC and TA4SP were not applicable. For phase 1, the SATMC checker resulted into memory exhaustion (it required more than 3 Gigabytes), while the TA4SP was unable to process the protocol, due to a limitation of this checker. For phases 2, 3 and 4, both the SATMC and TA4SP checkers were unable to handle the *exponential* function, required for the Diffie-Hellman exchange. Hence, only checkers OFMC and Cl-Atse were used to verify the proposal.

For each one of these checkers, four combinations of sessions were executed: one single session of the protocol, two parallel sessions of the protocol, a session of the protocol running in parallel with another session where the intruder is the end user, and a session of the protocol running in parallel with a session where the intruder is the Access Point (AP) or the application service (AppS). The motivation for the execution of two parallel sessions is to let AVISPA simulating replay attacks, using information from one session into the other. It is assumed that the RADIUS servers in both organizations, the KDC, the IdP and the PDP are trusted entities and the intruder cannot supplant any of them. While Table 5.3 shows the obtained results for the analysis of phase 1, Table 5.4 shows the results for the analysis of phases 2, 3 and 4.

The OFMC checker returned *SAFE* (no attacks were found) for every combination of sessions. The same happened with the Cl-Atse checker, but under some considerations. Specifically, the results marked with an (*) indicates that the analysis did not ended in a reasonable amount of time (in this case, less than 48h), due to the complexity of the specification, that contains many different roles and transitions. For these cases, simplified models were created and analysed instead, where the IdP and the PDP agents (and their transitions) were removed. These simplified models are still valid, as the changes do not affect to any of the *authentication* security goals, or any of the *secrecy* goals involving

keying material. Specifically, for phase 1, the generation of the *edutoken* is performed by the RADIUS server. For phase 3, the authorization phase is just omitted. With these changes, the simulation ended in a reasonable amount of time, indicating a SAFE result.

5.6 Conclusions

The proposals described in Chapters 3 and 4 assume the end user is already connected to the network. However, in some scenarios, the end user is not granted to access the network by default. Instead, she may be required to perform an access control process before attaching to the network. Moreover, this network access process can also be carried out in a federated way, using EAP and AAA to transport authentication information between the end user, the visited organization (and service provider), and the home organization. The most successful example of federated network is eduroam, briefly described in Section 2.4.1.

This chapter describes how the world-wide spread eduroam network can be extended to provide end users with SSO access to federated services beyond the web, as a result of a successful network access authentication. DAMe contribution provides authorization and token distribution to organizations willing to offer added value network access service to roaming users. By integrating Kerberos into this infrastructure, end users can, after performing the network access authentication, make use of the obtained token and the derived cryptographic material (i.e. EMSK) to perform a Kerberos pre-authentication process before obtaining a TGT. Hence, the network federation evolves into a cross-layer service federation, ranging from the network access to any kind of application services (SSH, FTP, SMTP, XMPP, Cloud, etc.). Besides, the distribution of the token to the KDC provides our solution with enhanced authorization capabilities to decide whether the visiting end user should be granted to obtain a specific service ticket (ST) or not.

This chapter has described in high detail the architecture of this proposal, and provided a thorough description of the exchanges that are required to achieve the required functionality. Moreover, this chapter also provided a complete security analysis of the proposal, which includes its validation using a reputed security tool: AVISPA.

Chapter 6

Performance evaluation and functional validation

6.1 Introduction

Chapters from 3 to 5 have presented three different proposals to provide SSO access to kerberized applications in AAA-based federations, each one of them solving different deployment requirements. In addition to a clear motivation for using Kerberos and AAA, these chapters have provided design of the architectures, detailed messages and information flows, and security analysis. However, the performance and functional validation of these proposals stands so far just on theoretical premises. What about the practical perspective? How complicated is their implementation? Are they reasonable in terms of how much time they need to complete an authentication and authorization process? Therefore, there is a necessity of validating these proposals in terms of both, performance and functionality.

In particular, it is important to determine whether any of these proposals requires a reasonable quantity of time and resources to be executed (efficiency). A valid solution that requires too much time to complete will exasperate end users, that will use instead a faster alternative. Besides, this time is a direct consequence of either computational time and transmission time (network overhead cost), both of them worth of being reduced for sake of resource optimization.

Furthermore, although a proper design is the cornerstone of any good solution, a reference implementation provides further confidence on its validity in terms of functionality (effectiveness). Besides, trying to implement a solution is one of the best ways to find flaws or inconsistencies on its specification, as can be seen in [178]. As one of the IETF slogans praises: "*We believe in rough consensus and running code*" [179].

Therefore, due to the importance of having an implementation, it has been implemented a prototype for each one of the proposals explained in this thesis, with the purpose of demonstrate their functional feasibility.

Hence, this chapter focuses on the performance evaluation and the functional validation of the different architectures proposed on the previous chapters, in order to demonstrate their feasibility beyond their theoretical standpoint. First, a high level performance model is provided. This model describes, with a reduced number of variables, the main time-consuming tasks required to complete the execution of each one of the proposals, estimating their magnitude. The main goal of this model is to demonstrate that the three proposals are similar in terms of performance, and that the differences that might exist between them should not be an important decision element when it comes to select one of them over the other. Second, the prototypes developed for this thesis are described. These prototypes provide a functional validation of the three proposals described in this thesis. Third, an empirical performance analysis is provided, based on the results obtained from the execution of the developed prototypes. These results have the purpose of confirming the accuracy of the model, showing overall execution and computational times for each one of the phases of the proposals.

6.2 Performance model

The total time (TT) required by an end user to authenticate with her H-AAA, obtain a valid TGT in the service provider domain, retrieve a ST for the application service, and then access the service, can be broken down into a set of nine individual variables, representing the time spent on each one of the main performed tasks. Given that the three proposals of this thesis are based on similar technologies, these variables are shared by all of them. The following equation is a generic representation of the total time valid for the three proposals, and represents the general (and simple) performance model:

$$TT = T_{EAP} + T_{LOW} + T_{DER} + T_{AUTHZ1} + T_{ENF} + T_{AS} + T_{TGS} + T_{AUTHZ2} + T_{AP} \quad (6.1)$$

The following list enumerates these variables, explaining what they represent, and showing the difference that might exist between their value on each one of the proposals:

- $\boldsymbol{T_{EAP}}$ represents the time associated with the EAP authentication. It includes the computational time of the EAP peer, authenticator and server for both, the EAP tunnelled and inner methods. It also includes the network transmission times of

the multiple round-trips required to complete the execution of the EAP method. Finally, it includes the time associated with the derivation of the MSK and EMSK. The value of this variable will highly depend on the selected EAP method to be executed. As all the proposals use an EAP-tunnelled method (i.e. EAP-TTLS or PEAP), this variable is expected to be the one with the highest value of the model, as it will require the execution of different cryptographic operations, and a considerable number of round-trips to establish the TLS channel (due to the fragmentation of the TLS exchanges performed by these mechanisms). This variable will be similar for the three proposals.

- $\boldsymbol{T_{LOW}}$ represents the time associated with the EAP lower-layer functionality. It includes the encapsulation of the EAP packets into protocol specific structures, as well as its protection (if any). This time is repeated for each EAP exchange performed between the EU and the EAP authenticator (i.e KDC, AP, or PAA). It is expected to have a medium impact, as it should be a matter of copying data and performing light cryptographic operations for each round-trip. It is different for each one of the proposals. In particular:

  - $\boldsymbol{T_{LOW_{FedKERB}}}$. FedKERB uses Kerberos as EAP lower-layer, although some of the EAP lower-layer responsibilities lie on the GSS-EAP mechanism. Due to the inherent stateless nature of the KDC, any state associated with the EAP authentication must be exported from the KDC, encrypted using symmetric cryptography, and sent to the EU (Section 3.5.3. They are re-sent from the EU to the KDC, decrypted and imported. This process is repeated for every EAP round-trip, as explained in [135]. This state import/export and the corresponding encryption requires an increased amount of time when compared to stateful protocols.

  - $\boldsymbol{T_{LOW_{PanaKERB}}}$. PanaKERB uses PANA as EAP lower-layer. Although PANA also makes use of symmetric cryptography to protect the protocol messages, the PANA protocol is stateful, meaning there is no need to export and import the state constantly. Therefore, the overload introduced by PANA is expected to be lower than the one from Kerberos.

  - $\boldsymbol{T_{LOW_{EduKERB}}}$. EduKERB uses 802.11 as EAP lower-layer for network access. It is lighter than Kerberos and PANA, as it operates at link layer and does not introduce any kind of encryption during the EAP negotiation.

Thus, the following relationship can be established between them:

$$T_{LOW_{FedKERB}} > T_{LOW_{PanaKERB}} > T_{LOW_{EduKERB}} \tag{6.2}$$

- $\boldsymbol{T_{DER}}$ represents the time associated with performing the key derivation process required to generate the *reply key* for Kerberos, starting from the MSK/EMSK keys derived after the EAP authentication. This presents slight differences between proposals. Nevertheless, it is expected to have a low impact as it is executed only once, and the key derivation process typically consists of one or more hash operations.
  - $\boldsymbol{T_{DER_{FedKERB}}}$. FedKERB derives the *reply key* directly from the MSK, that has already been distributed to the KDC as part of the EAP authentication.
  - $\boldsymbol{T_{DER_{PanaKERB}}}$. PanaKERB derives the *reply key* from a password, which is derived from the MSK, distributed to the PAA as part of the EAP authentication. This derivation process involves an additional step than the one for FedKERB.
  - $\boldsymbol{T_{DER_{EduKERB}}}$. EduKERB derives the *reply key* from the DSUSRK. This key is derived from the DSRK, which is in turn derived from the EMSK. Besides, it requires some specific round-trips for key distribution. Therefore, this variable is expected to be significantly higher on this proposal than on the previous ones.

  The following relationship between them is established:

$$T_{DER_{EduKERB}} > T_{DER_{PanaKERB}} > T_{DER_{FedKERB}} \tag{6.3}$$

- $\boldsymbol{T_{AUTHZ1}}$ represents the time associated with the generation and distribution of the SAML assertion. This variable is expected to have a high impact on the total time, as it will typically require to establish a TLS channel between the H-AAA and the IdP, and the use of asymmetric cryptography to sign the SAML assertion. This variable will take the same value for all the proposals.

- $\boldsymbol{T_{ENF}}$ represents the time associated with the distribution and installation of the *reply key* on the KDC. This process requires a very different amount of time on each one of the proposals.
  - $\boldsymbol{T_{ENF_{FedKERB}}}$. FedKERB does not have a proper enforcement phase, as the key is derived directly on the KDC. This process would consist of just a memory

copy process, making this time negligible (i.e. $< 1ms$) in the overall time of this proposal.

- $\boldsymbol{T_{ENF_{PanaKERB}}}$. PanaKERB enforces the *reply key* from the PAA to the KDC. This enforcement is based on the KADM interface, and will be highly dependent on how this interface is implemented (e.g. use of security channels such as IPsec or TLS). For this thesis it is assumed the collocation of the PAA and the KDC, requiring no security channel for this interface.
- $\boldsymbol{T_{ENF_{EduKERB}}}$ EduKERB enforces the *reply key* from the SP-AAA to the KDC. This enforcement is based on a single RADIUS exchange (request and response), using the RADIUS attributes described in chapter 5.

From the descriptions above it is concluded that:

$$T_{ENF_{EduKERB}} > T_{ENF_{PanaKERB}} > T_{ENF_{FedKERB}} \tag{6.4}$$

- $\boldsymbol{T_{AS}}$ represents the time associated with the Kerberos AS exchange. It includes the creation and processing of the KRB_AS_REQ and KRB_AS_REP messages, and the generation of the TGT. This time presents sensible differences between proposals:
  - $\boldsymbol{T_{AS_{FedKERB}}}$. FedKERB does not introduce any particular addition to the standard Kerberos AS exchange (note that all the GSS-EAP processing will be assumed by the T_{EAP} and T_{LOW} variables). Therefore, this time is expected to be low, since Kerberos security is based on symmetric cryptography [180].
  - $\boldsymbol{T_{AS_{PanaKERB}}}$. PanaKERB does not introduce any particular addition to the standard Kerberos exchange at all. Therefore, this time is expected to be low for the same reasons as $T_{AS_{FedKERB}}$.
  - $\boldsymbol{T_{AS_{EduKERB}}}$. EduKERB requires to perform an anonymous PKINIT exchange, and then the usage of FAST as a way of protecting the transmission of the *eduToken*. The former is based on asymmetric cryptography and requires a non-negligible amount of time to complete. The later is based on symmetric cryptography, but still requires more processing than the standard AS exchange used by the other two proposals.

Thus, the following relationship is established between them:

$$T_{AS_{EduKERB}} > T_{AS_{FedKERB}} \approx T_{AS_{PanaKERB}} \tag{6.5}$$

- $\boldsymbol{T_{TGS}}$. Represents the time associated with the Kerberos TGS exchange. It includes the creation and processing of the KRB_TGS_REQ and KRB_TGS_REP messages, the processing of the TGT and the generation of the ST. This time is again expected to be low, for the same reasons as T_{AS}. This variable will take the same value for all the proposals.

- $\boldsymbol{T_{AUTHZ2}}$. Represents the time associated with the authorization performed by the KDC, based on the SAML assertion contained in the TGT. This time is expected to have a high impact, as it will typically include the verification of the assertion, the optional query to the MDS, the attribute query to the IdP, the verification of the received attribute statements, and the query to the PDP, including all the policy matching process. This variable will take the same value for all the proposals, as this process is identical for all of them.

- $\boldsymbol{T_{AP}}$. Represents the time associated with the Kerberos AP exchange. It includes the creation and processing of the KRB_AP_REQ and KRB_AP_REP messages, and the processing of the ST. This variable is expected to be low [180], and identical for the three proposals.

From the previous descriptions and equations, it is concluded that the variable with the highest impact on the overall execution time (TT) is T_{EAP}, as it requires multiple round-trips and several cryptographic operations. On the second place we can find T_{AUTHZ1} and T_{AUTHZ2}. These two variables require the use of asymmetric cryptography and complex XML processing (generation and validation of the SAML assertion). The rest of variables have almost a low cost when compared to these three variables. The following equation summarizes the impact that each variable has on the overall execution time:

$$T_{EAP} > T_{AUTHZ1} \approx T_{AUTHZ2} >> (T_{LOW} + T_{DER} + T_{ENF} + T_{AS} + T_{TGS} + T_{AP}) \quad (6.6)$$

As it can be observed, the variables with the highest impact (i.e. T_{EAP}, T_{AUTHZ1}, and T_{AUTHZ2}), and therefore expending most of the execution time, are almost identical for all three proposals. The main differences are presented in variables with a low or medium impact (e.g. T_{LOW}). This makes us assume the three proposal will perform very similarly.

Although these variables appear on all the proposals, they occur at different phases of the process for each one of them. Table 6.1 shows how these times are distributed along their different phases. The processing associated to the variable T_{DER} is sometimes split across different phases. In those cases, the variable is denoted as T^*_{DER}. In particular, in

PanaKERB, T_{DER} is partly performed in phase 2, where the PAA derives the password from the MSK and the KDC derives the *reply-key* from it; and partly in phase 3, where the EU derives the *reply-key* following the same process. In EduKERB, T_{DER} is performed partly in phase 1, where the H-AAA derives the DSRK from the EMSK; and partly on phase 2, where the SP-AAA derives the DSUSRK from the DSRK, the KDC derives the *reply-key* from it, and the EU derives both following the same process.

Table 6.1: Distribution of variables for the different proposals.

Phase	**Time**
(1) Kerberos preauth.	$T_{EAP} + T_{LOW} + T_{DER} + T_{ENF} + T_{AS} + T_{AUTHZ1}$
(2.a) TGS exchange	$T_{TGS} + T_{AUTHZ2}$
(2.b) AP exchange	T_{AP}

(a) Distribution of variables for the FedKERB proposal.

Phase	**Time**
(1) PANA auth.	$T_{EAP} + T_{LOW} + T_{AUTHZ1}$
(2) KRB enforc.	$T^*_{DER} + T_{ENF}$
(3) Kerberos auth.	$T^*_{DER} + T_{AS}$
(4.a) TGS exchange	$T_{TGS} + T_{AUTHZ2}$
(4.b) AP exchange	T_{AP}

(b) Distribution of variables for the PanaKERB proposal.

Phase	**Time**
(1) Network auth.	$T_{EAP} + T_{LOW} + T^*_{DER} + T_{AUTHZ1}$
(2) Kerberos preauth.	$T_{AS} + T^*_{DER} + T_{ENF}$
(3.a) TGS exchange	$T_{TGS} + T_{AUTHZ2}$
(3.b) AP exchange	T_{AP}

(c) Distribution of variables for the EduKERB proposal.

6.3 Prototypes description

In order to validate the three proposals described in this thesis, to demonstrate their feasibility and efficiency, and to contrast the model described in Section 6.2 with data from an actual implementation, a prototype for each one of them has been developed. All these prototypes share the same ultimate objective: to bootstrap a Kerberos TGT from the KDC of a service provider, and then use it to access to an out-of-the-box SSH server (example application service) deployed on the same service provider. Each prototype use a different approach that will be more adequate to a specific scenario, as described in

Chapters 3, 4, and 5.

It is important to comment that these prototypes implement only the authentication steps. The authorization ones have not been implemented for this thesis, since the three proposals share a common authorization model, which was already implemented and evaluated during the development of the *EduKERB* proposal and, thus, there are previous references of its performance [181]. These values are provided in Subsection 6.4.2.

To implement the different components of these architectures, we have used existing open-source software. These solutions provide most of the required functionality, avoiding the need for implementing the whole system from scratch. Tables 6.2a, 6.2b and 6.2c contain the list of the software used for each prototype, as well as their specific versions. These tables also include a column indicating whether the open source implementations have required additional modifications to their source code to accomplish the required functionality or not.

As it can be observed from the tables, none of the solutions have required the modification of the *application services*. This is a common sub-objective of the three contributions: avoid the modification of already deployed and working application services, since it may suppose a high inconvenient for many service providers, therefore limiting its appeal, and making its adoption more difficult.

Regarding the rest of components, FedKERB has only required the modification of the EU and the KDC. More specifically, these two components have been modified to implement the Kerberos GSS-API pre-authentication mechanism, as described in chapter 3.4.1. The implementation of this mechanism has been brought beyond the limits of just a prototype. Indeed, it has been performed in collaboration with the Cardiff University [185], under the supervision of Dr. Rhys Smith, and has been published as a MIT Kerberos pre-authentication plugin [137], following all the coding styles and documentation guidelines required by MIT.

PanaKERB is the one contribution which inflicts less modifications, requiring only small changes to the *openpana* source code. The implementations of the rest of components have been used in their out-of-the-box form. Indeed, only small parts of the PAA and EU need to be adapted. More specifically, the PAA has been modified to enforce the bootstrapped Kerberos credentials (i.e. *username* and *password*) after the EU authentication, while the EU was modified to make use of these Kerberos credentials in order to be authenticated by the KDC.

Finally, the EduKERB proposal has required the modification of several components. Most of these modifications have been slight additions that have just required a few lines of code. The *wpa_supplicant* [184] software has been modified to 1) receive the *edutoken*

Component	Software	Version	Modified
EU	MIT Kerberos [136]	1.10.2	Yes
	Moonshot [37]	-	No
SP-AAA	FreeRadius [145]	2.1.10	No
H-AAA	FreeRadius	2.1.10	No
KDC	MIT Kerberos	1.10.2	Yes
AppS	OpenSSH [182]	5.8	No

(a) Software used in the FedKERB prototype.

Component	Software	Version	Modified
EU	MIT Kerberos	1.10.2	No
	openpana [183]	0.1	Yes
SP-AAA	FreeRadius	2.1.10	No
H-AAA	FreeRadius	2.1.10	No
PAA	openpana	0.1	Yes
KDC	MIT Kerberos	1.10.2	No
AppS	OpenSSH	5.8	No

(b) Software used for the PanaKERB prototype.

Component	Software	Version	Modified
EU	MIT Kerberos	1.10.2	Yes
	wpa_supplicant [184]	0.7.3	Yes
SP-AAA	FreeRadius	2.1.10	Yes
H-AAA	FreeRadius	2.1.10	Yes
KDC	MIT Kerberos	1.10.2	Yes
AppS	OpenSSH	5.8	No

(c) Software used for the EduKERB prototype.

Table 6.2: Software used for the prototypes.

from the H-AAA; and 2) to export the *edutoken* and the MSK to two different files, in order to make them available to the *kinit* program later on. The EU has also been modified to implement the Kerberos pre-authentication mechanism that uses those values, as described in Chapter 5. The H-AAA (FreeRadius) has been modified to provide the *edutoken* to the EU, and to provide the EMSKName and the DSRK to the SP-AAA. The SP-AAA (FreeRadius) has been modified to request the DSRK from the H-AAA and to provide the DSUSRK to the KDC. Finally, the KDC has been modified to implement the

Kerberos pre-authentication mechanism based on the *edutoken* and the EMSK, including the request of the DSUSRK to the SP-AAA.

In terms of functionality, all these three prototypes have been very successful, as they all have accomplished with the require functionality. They have behaved as expected and provided the foreseen results with no major complications. Hence, from a functional perspective, the architectures proposed in previous chapters have been validated.

6.4 Performance measurements

After having modelled the performance from an analytical perspective, this chapter provides an empirical analysis based on the results obtained from the execution of the prototypes. These tests have the purpose of confirming the accuracy of the model, showing overall execution and computational times for each one of the phases of the proposals.

To carry out this performance analysis, the tests and time measurements needed to be performed under equal conditions, and following a common procedure. For that reason, a testbed that represents the architectures described in sections 3.2, 4.2, and 5.3 has been set up. This testbed allows staging a federated scenario, replicating the conditions of a real production environment.

6.4.1 Testbed description

The testbed components are distributed in two different locations, representing two different members of the federation: University of Murcia[1] and Janet[2]. More precisely, the University of Murcia plays the role of service provider, while Janet acts as the home organization. Figure 6.1 depicts the deployment of the different machines and components that take part of the testbed.

In particular, the deployment at the University of Murcia comprises three virtual machines (VMs), a laptop, and an AP. The VMs are deployed at three different hosts from the GAIA experimentation infrastructure [186]. The computational wing of this infrastructure is based on the XEN [187] virtualization software running on top of 22 different machines, each provisioned with a dual-core *AMD Opteron(tm) Processor 246* at 2 GHz, and 2 GiB of RAM. Each VM is assigned with 128 MB of RAM, and has *Debian Squeeze* [188] installed as operating system. The laptop is equipped with a dual-core *Intel(R) Core(TM) i3-3227U* at 1.9 GHz, with 4 GiB of RAM. It has *Arch Linux* [188]

[1] www.um.es

[2] www.ja.net. UK National Research and Education Network (NREN).

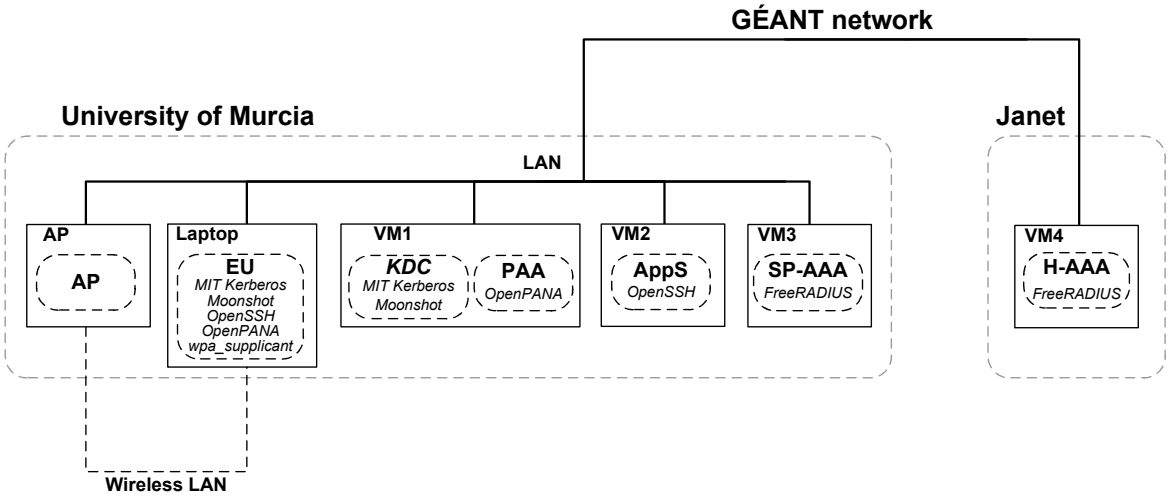


Figure 6.1: Components deployment.

installed as operating system. The interconnection between these VMs, the laptop and the AP is performed through a 100 Mbps LAN. There is also a 54 Mbps wireless connection between the laptop and the AP.

On the other hand, the deployment at Janet comprises one virtual machine, at one host. Its computational wing is also based on the XEN virtualization software running on top of a machine provisioned with a quad-core *AMD Opteron(tm) Processor 6176* at 2.3 GHz. The VM is assigned with 2GiB of RAM, and has *Debian Squeeze* [188] installed as operating system.

The communication between both organizations is performed through GÉANT, which provides a high-speed pan-European connection.

The results extracted from this testbed can be considered equivalent in every sense to those from a real deployment. The reason is twofold. On the one hand, computational times are representative, as of being virtualized does not differ much from what it is being done in several organizations, where many services in production are deployed within virtual machines (e.g. cloud computing). On the other hand, as this testbed involves two different locations, transmission times are also representative, as there are several intermediate hops and hundreds of kilometres between them.

As the three proposals are composed by almost the same components, a common deployment can be used for them. In particular, each component of the architectures was deployed into an individual VM. One exception was the PAA (Section 4.2), which was collocated with the KDC in the same VM as, in this prototype, the KADM interface consists of a local interprocess communication between these two components.

To make this testbed even more realistic, the H-AAA and the SP-AAA components are directly connected to the actual eduroam RADIUS infrastructure. Section 6.5 provides some considerations on this aspect. With this connection, RADIUS packets from one organization to the other follow the same path as they would do in a production environment, with an additional level for these subdomains. The RADIUS realms for these two components were configured as ms-perf.dev.ja.net (H-AAA), and moonshot.inf.um.es (SP-AAA). A few testing user accounts have been created on H-AAA to perform the tests (e.g. alex@ms-perf.dev.ja.net).

Figure 6.2 depicts the hierarchy of eduroam's RADIUS servers interconnecting both components.

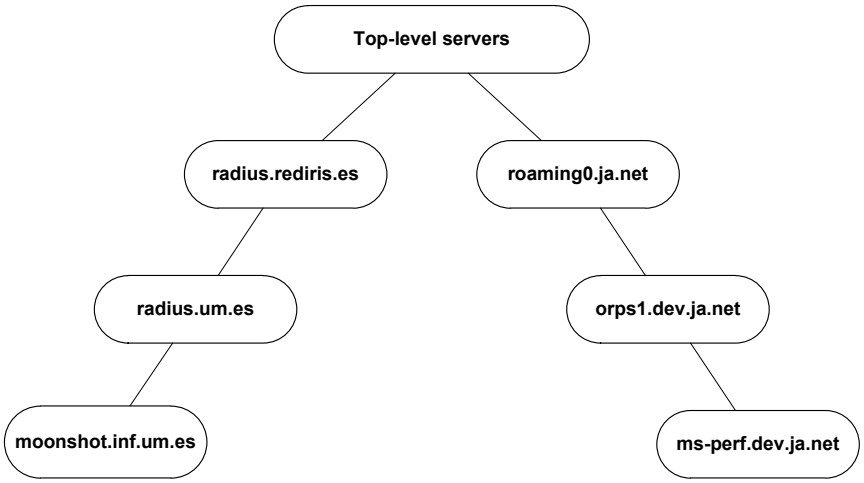


Figure 6.2: Extended eduroam's RADIUS hierarchy.

6.4.2 Execution of the tests

Using the testbed described above, around 500 executions were executed for each one of the three proposals under evaluation. These executions aimed to obtain an estimation of the time required to perform a complete federated access to an application service. That is, the time elapsed since the EU tries to access a particular application service deployed in a service provider, without having any kind of pre-shared state with it, until she is actually provisioned with the requested service. This time has been decoupled into:

- *Computational time.* It measures the total time (represented as a 95% confidence interval) spent by every component on the computations associated with a particular phase of each proposal. In particular, the computational time imputable to a

component is calculated as the sum of the time elapsed since a message is received by that component until a subsequent message is sent to another component. For the EU component, this time also includes the time elapsed from the start of the phase until the first message is sent, and the time elapsed since the last message is received until the end of the phase. The computational time is measured from the output generated by the Wireshark [189] network protocol analyser, running locally on each of the components. There is no need to synchronise clocks among the different components, as the computational time is measured on each VM as a difference of local timestamps.

- *Execution time.* It measures the total time (represented as a 95% confidence interval) required by the EU component to complete a particular phase of one of the proposals. It includes the computational time spent by all the involved components, as well as the network transmission times. The execution time has been measured based on the timestamps from the Wireshark's log output generated by the EU component.

In order to make these three proposals comparable, a similar EAP method was selected for all the scenarios. In particular, EAP-TTLS was selected, executing EAP-MD5 as the inner method. EAP-TTLS was selected because 1) it is a key generating method, as required for all the proposals; 2) it is a widely used EAP method in federations (e.g. eduroam). However, for the *EduKERB* prototype PEAP has been used instead. The reason is merely operational, as the existing code for DAMe is based on PEAP. Nevertheless, both are very similar in terms of performance. Although PEAP requires one additional round-trip to complete, it does not have a great impact on the overall execution time. Furthermore, all the executions were performed during the same time frame in order to assure a similar usage level of the eduroam infrastructure.

It is also important to mention (due to its repercussion on the total time) that all the RSA keys used in these simulations are 2048-bits long, while all the Diffie-Hellman operations have been performed with a 1024 bit modulus.

Table 6.3 collects the results obtained from the execution of the tests. As observed, for each phase of each proposal, we indicate the total execution time (including network time), as well as the different computational time spent by each individual component. These values are represented as 95% confidence intervals, denoted by *mean* $\pm$ *offset*. The KRB_TGS and KRB_AP exchanges have been measured only once, and their values used for all the proposals, as they are completely identical for all of them. Figure 6.3 shows a graphical representation for each proposal comparing the computational time for each phase, and another one with a summary comparing the total times for each proposal.

Table 6.3: Time measurements for the prototypes.

Phase	Computational time EU	KDC	AppS	RAAA	HAAA	Total	Execution time (incl. network)
(1) Kerberos preauth.	21.56 ±0.13	15.34 ±0.92	— —	4.53 ±0.01	9.71 ±0.07	51,14 ±1.13	1255.78 ±12.83
(2a) TGS exchange	2.87 ±0.02	0.77 ±0.01	— —	— —	— —	3,64 ±0.03	4.18 ±0.02
(2b) AP exchange	1.92 ±0.01	— —	2.05 ±0.03	— —	— —	3.97 ±0.04	4.44 ±0.03
Whole process	**26.35 ±0.16**	**16.11 ±0.93**	**2.05 ±0.03**	**4.53 ±0.01**	**9.71 ±0.07**	**58,75 ±1.20**	**1264.40 ±12.88**

(a) Time measurements for the FedKERB prototype (in milliseconds).

Phase	Computational time EU	KDC	PAA	AppS	RAAA	HAAA	Total	Execution time (incl. network)
(1) PANA auth.	13.03 ±0.13	— —	5.95 ±0.02	— —	5.65 ±0.06	9.66 ±0.03	34.29 ±0.24	1396.81 ±32.07
(2) KRB enforc.	— —	18.56 ±0.26	1.12 ±0.02	— —	— —	— —	19.68 ±0.28	20.60 ±0.24
(3) Kerberos auth.	20.21 ±0.11	0.69 ±0.00	— —	— —	— —	— —	20.90 ±0.11	22.03 ±0.12
(4a) TGS exchange	2.87 ±0.02	0.77 ±0.01	— —	— —	— —	— —	3.64 ±0.03	4.18 ±0.02
(4b) AP exchange	1.92 ±0.01	— —	— —	2.05 ±0.03	— —	— —	3.97 ±0.04	4.44 ±0.03
Whole process	**38.03 ±0.27**	**20.02 ±0.27**	**7.07 ±0.04**	**2.05 ±0.03**	**5.65 ±0.06**	**9.66 ±0.03**	**82.48 ±0.70**	**1448.06 ±32.48**

(b) Time measurements for the PanaKERB prototype (in milliseconds).

Phase	Computational time EU	KDC	AppS	RAAA	HAAA	Total	Execution time (incl. network)
(1) Network auth.	3.01 ±0.01	— —	— —	7.22 ±0.02	11.19 ±0.06	21.42 ±0.09	1425.84 ±2.70
(2) Kerberos preauth.	10.83 ±0.49	42.66 ±0.55	— —	— —	— —	53.49 ±1.04	54.02 ±1.14
(3a) TGS exchange	2.87 ±0.02	0.77 ±0.01	— —	— —	— —	3.64 ±0.03	4.18 ±0.02
(3b) AP exchange	1.92 ±0.01	— —	2.05 ±0.03	— —	— —	3.97 ±0.04	4.44 ±0.03
Whole process	**18,63 ±0.53**	**43.43 ±0.56**	**2.05 ±0.03**	**7.22 ±0.02**	**11.19 ±0.06**	**82.52 ±1.20**	**1488.48 ±3.89**

(c) Time measurements for the EduKERB prototype (in milliseconds).

Although not implemented for these prototypes, its is worth mentioning how much time authorization would require. According to the performance model described in 6.2, this functionality is expected to have a high impact. Indeed, the measurements taken in [181] show that T_{AUTHZ1} may require up to 1000 ms., while T_{AUTHZ2} would require up to 650 ms. to complete. Note that the execution environment of those tests were significantly less powerful than the one used for this thesis. In any case, both of these times would require several hundreds of milliseconds to complete.

6.4.3 Analysis of results

This section analyses the empirical results obtained from the execution of the tests described in Section 6.4.2. The aim of this analysis is to validate the performance model described in Section 6.2. However, while the model is expressed in terms of variables, there is no practical way to measure the time associated to each one of these variables during the execution of the tests. Instead, the measurements have been performed in terms of phases, where each proposal has a different number of phases, with different purposes. Hence, the methodology followed for this analysis consists of inferring the magnitude of the variables of the model using the values in Table 6.3 and the distribution of the variables along the different phases described in Table 6.1.

- $\boldsymbol{T_{EAP}}$. In terms of the time required to complete the whole authentication and the access to the application service (see *Whole process* row, *Execution times* column), all the proposals devote similar times (between 1264 and 1488 ms.). As expected, most of the time is spent during the EAP authentication (included in phase 1 of all proposals). This large time is mostly due to the network transmission time, as the computational time of phase 1 is not higher than 52 ms. for any proposal. It can also be noted how these transmission times fluctuate compared to the stability of the computational times. This fluctuation is a direct consequence of using a real production network such as eduroam. For example, even though phase 1 of PanaKERB requires less computational time than phase 1 of FedKERB (34 ms. vs. 51 ms. respectively), the reality is that, during the tests, PanaKERB found a more unreliable network, resulting into higher overall execution times (1396 vs. 1255 respectively). In the case of EduKERB (1425 ms.), the increment is a consequence of using PEAP instead of TTLS, which requires an additional round-trip.

- $\boldsymbol{T_{LOW}}$. The computational times confirm that T_{LOW} differences are indeed as foreseen in the analytical model. The PAA (EAP authenticator in PanaKERB) requires significantly less computational time ($\approx$3 times lower) than the KDC (EAP authenticator in FedKERB) to perform the forwarding of the EAP packets between the EU and the RAAA. This comes from the stateless behaviour of the KDC, as explained in the analytical model, which generates an overload in both computational time and amount of data transmitted in the network. This difference is also noticeable on the clients. As expected, the EduKERB client requires less time (between 4 and 7 times) than the rest of the proposals. The difference between the PanaKERB and FedKERB clients can also be imputable to the stateless nature of the Kerberos

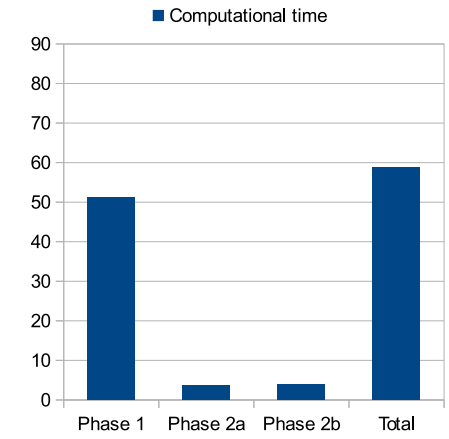
protocol. This makes the EU try to find the best available KDC on each iteration of the process, including the parsing of a list of available KDCs, and their name resolution.

- $\boldsymbol{T_{ENF}}$ & $\boldsymbol{T_{DER}}$. Phase 1 of FedKERB includes more computation (i.e. T_{ENF} and T_{DER}) than phase 1 of PanaKERB or EduKERB, contributing to its increased computational time (51 ms. vs. 34 ms. and 21 ms., respectively). Conversely, PanaKERB and FedKERB require these enforcement and derivation tasks to be performed in additional phases. On the one hand, PanaKERB spends ≈12 ms. executing the *string-to-key* function [190], which derives a binary key from the textual password, as part of its T_{DER} (phases 2 and 3). The bottleneck problem with the string-to-key function situation was already detected in [180]. This call is not performed by neither the FedKERB nor the EduKERB proposals, as the binary key is derived directly from the MSK. This situation would be mitigated if the implementation of the KADM interface was optimized in such a manner that the use of binary keys is allowed instead of textual ones. In this way, the call to the *string-to-key* function would be avoided, so speeding up these phases and, consequently making the overall process faster.

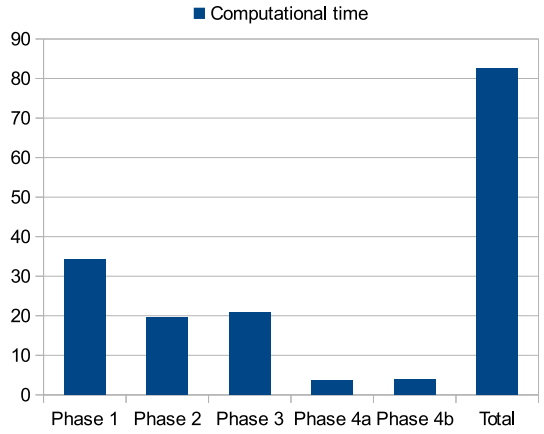
  On the other side, EduKERB spends a considerable amount of computational time on its phase 2 (53 ms.), specially on the KDC (42 ms.), as par of its T_{ENF}. The main reason for this is that it is not an optimised implementation. Indeed, to create the RADIUS *Access-Request* packet and to process the response, the prototype makes use of an external program (*radclient*). This option was taken for convenience, as the resulting code is much simpler. However, this requires the creation of a new process, spending $\approx 30ms$. This situation would be mitigated if a RADIUS library were used from the prototype instead of using an external program.

- $\boldsymbol{T_{AS}}$. Finally, as described in the model, EduKERB is expected to present high values for T_{AS} due to the use of FAST. This is shown on the computational times of the EU and the KDC of phase 2 (10 ms. and 42 ms. respectively), that would have been lower otherwise.

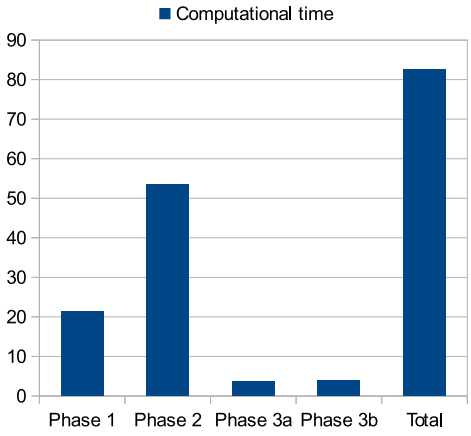
Therefore, the empirical results obtained after the execution of the three prototypes confirm that the three proposals perform similarly, confirming the validity of the model presented in Section 6.2. Besides, the times required to perform their functionality seem quite reasonable, as they do not incur in any substantial overhead when compared to the actual time required by the eduroam EAP authentication (which, in this scenario, coincides



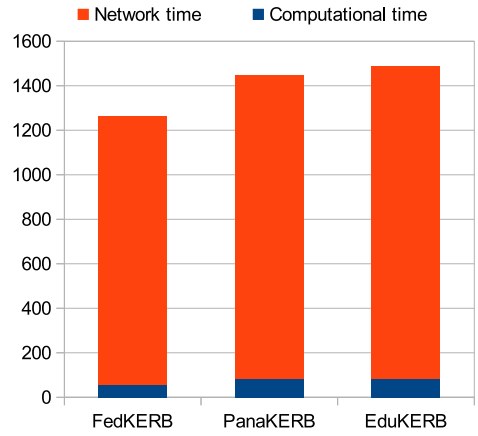
(a) FedKERB.



(b) PanaKERB.



(c) EduKERB.



(d) Comparative.

Figure 6.3: Time graphs for the prototypes.

with the time of *EduKERB's* phase 1). It is worth noting that the EU will only require this amount of time during the first access to an application service, in order to bootstrap a security association with the service provider's KDC. Thanks to the Kerberos SSO capabilities, subsequent accesses to application services deployed on that service provider within the session lifetime will be based on the standard Kerberos operation (i.e. TGS and AP exchanges requiring only ≈ 4.8 ms.).

6.5 Considerations on the use of an already existing AAA federation

One important consideration that must be taken into account before starting to deploy any of the proposed contributions of this thesis (or any other identity federation beyond the web proposal) over an existing AAA federation, is whether that federation will administratively allow its utilization for that purpose. While from a technical standpoint, these contributions should work over any existing AAA federation (including the *eduroam*'s RADIUS infrastructure), that does not necessarily mean that this kind of traffic will be allowed from an administrative point of view. For instance, *eduroam* is an AAA-based federation which purpose is the provision of the network access service. Therefore, its usage for any other purpose is not explicitly allowed and, of course, not supported.

That is something we found out after our tests over the *eduroam* infrastructure (Section 6.4.1). In particular, we realized that certain RADIUS attributes (i.e. *GSS-Acceptor-Service-Name* [107]) were being sent from the AppS to the H-AAA, but they never arrived to their destination. The reason was that the eduroam's top level RADIUS proxies were transcoding that specific attribute code (i.e. 164) into a different one. They did this because that attribute code was illegally allocated by Ascend in the past, and thus they considered its usage as illegal. However, that attribute has been recently allocated by the IANA to RFC 7055 (GSS-EAP). When we asked them to disable that "outdated" transcoding, the request was kindly rejected based on the fact that our traffic was out of the scope of the eduroam's purpose and, hence, unsupported.

This transcoding did not affect the execution of the tests, since at the time they were performed, RFC 7055 had not been published yet, and a *vendor specific* attribute was being used instead for the *GSS-Acceptor-Service-Name* attribute. Once the document was published, and the GSS-EAP implementation was updated accordingly, the issue was revealed.

6.6 Conclusions

Having a tangible demonstration of a particular proposal is an excellent way to prove its feasibility. Community has shown its interest on having running code, something that can be tested. Therefore, three prototypes have been developed for the three proposals of this thesis, based on existing open-source solutions of the protocols that take part of the designed architectures. These prototypes implement the functionality described

in Chapters 3, 4, and 5. That is, they provide different means to allow an end user retrieving a valid TGT from a KDC, where the eduroam RADIUS infrastructure is used to convey authentication and authorization information from/to the home organization to/from the service provider. This chapter has provided several implementation details of these prototypes, indicating how they have been developed and the open-source solutions they are based on.

Afterwards, this chapter has performed an extensive performance analysis of the proposals. Firstly, from an analytical point of view, breaking down the time required to execute each proposal into a set of variables comprising the most relevant operations that are performed. Secondly, from an empirical point of view, deploying the developed prototypes into a real production environment (i.e. connected to the eduroam infrastructure), to measure the actual overall execution times required to complete the federated authentication process.

Finally, this chapter has analysed the empirical results obtained from the prototypes, comparing them with the specified model.

The results form this performance analysis have confirmed that the three proposals described in this thesis can be executed in a reasonable amount of time. That is, they present similar execution times to the ones required for the network access service (which is used as a reference since it is the most extended use of AAA infrastructures). Moreover, as they have been implemented and executed in a real environment, their feasibility has also been demonstrated. In this sense, it must be noted that that, even though FedKERB has shown the bests results, the different is not significant enough to use it as a selection criteria. Instead, that decision should be merely based on operational premises, such as which one fits better with the actual requirements of the target scenario, or which one has a minor impact on the existing infrastructure. For instance, if an organization does not want to deploy any additional protocol, *FedKERB* would most probably be the best choice. However, if no modifications to the KDC are desired, *PanaKERB* would be the surest bet. Finally, if one organization wants to leverage network access to provide application service SSO, *EduKERB* should be the selected option.

Chapter 7

Conclusions and future work

This chapter provides a summary of the thesis, its main contributions and conclusions, and discusses the envisaged future work.

7.1 Summary and main contributions

The problem of controlling the access to network service applications is a topic that has been gaining interest as the Internet has become more popular and pervasive. A result of this increasing interest has been the research and design of new and more powerful access control architectures that allow, on the one hand, enforcing access control policies with a high level of detail and, on the other hand, increasing the usability and reduce the complexity of the processes to be performed by end users. In this sense, access control architectures have evolved from the more archaic approaches based on static access lists, to the most modern identity federations, which allow end users to access a wide range of application services deployed by different service providers. In particular, each federation comprises a number of service providers, which establish trust relationships with their respective identity providers, agreeing on a set of technologies and protocols to exchange authentication and authorization information.

As analysed in the PhD work, nowadays there are two types of well-established identity federations: *Web-based identity federations* and *Authentication Authorization and Accounting (AAA)-based identity federations*. On the one hand, the former focuses on providing federated access to web-based services. Several companies, such as Google, Amazon or Flickr, already support it. On the other hand, AAA is a generic framework which is nowadays used to control the network access service. The roaming in cellular networks, companies such as IPass, Mach, and Syniverse, as well as the *eduroam* network

are good examples of such AAA-based federations.

In this context, one of the more interesting challenges in current identity management research has been to define how federated access can be provided to other types of application services not supporting the integration neither with AAA-based nor web-based federations, such as file transfer, terminal access, etc. This type of federation is known as *Identity federation beyond the web*. This thesis has analysed the current state of the art on this type of federation and, in particular, it has focused on Kerberos and Moonshot, as the only two currently existing approaches to support this kind of federation. For them, this thesis has pointed out the different functionality and deployability gaps that they have, when compared to the more deployed and consolidated identity federations (i.e. web-based and AAA-based). Specifically, Kerberos requires the deployment of a complete independent federation for its *cross-realm* operation, and it lacks of fine-grained authorization support. On the other side, Moonshot lacks of support for SSO, and requirements the modification of every application service.

Keeping these gaps in mind, this thesis has aimed to design solutions for *Identity federations beyond the web* that do not suffer from them. In particular, after having analysed the most important current access control technologies, this thesis has concluded that, although Kerberos possesses great features for intra-domain access control (e.g. secure, lightweight, integrated SSO, etc.), and it is supported by most of current application services, it has some weak points related to federation and authorization support. In particular, the standard federated operation for Kerberos (i.e. *cross-realm*) has several issues, and it is not widely deployed. Besides, Kerberos authorization capabilities are very limited and not suitable for federated environments. Conversely, AAA-based federations have been used for decades to provide federated access to the network access service with great success and presence, due to their robustness and reliability. Therefore, the integration of these two technologies (i.e. Kerberos and AAA) offers an approach that supersedes the *Kerberos cross-realm* operation with a more widely adopted federation technology (i.e. AAA), allowing end users within the federation to authenticate without being previously registered on the Kerberos user database.

Hence, this thesis has looked for approaches that enable the interconnection of Kerberos with AAA infrastructures to provide federated authentication. Moreover, its has incorporated SAML processing capabilities to Kerberos to provide federated authorization. Specifically, this work has analysed, designed and validated three different approaches to perform this interconnection, each one solving different deployment requirements. Moreover, its has also designed a common authorization model that can be applied to all of them. These three approaches have been called *FedKERB*, *PanaKERB*, and *EduKERB*,

and they constitute the main contributions of this work. Each one has its advantages and disadvantages, making them more suitable for a specific set of scenarios than others. Table 7.1 summarises the main features of each contribution, focusing on their main aspects.

Table 7.1: Summary of contributions and features.

Feature	**FedKERB**	**PanaKERB**	**EduKERB**
Applicability scenarios	Any AAA federation	Any AAA federation	Eduroam RADIUS infrastructure
Standards used	Kerberos, AAA, EAP, SAML, GSS-API	Kerberos, AAA, EAP, SAML, PANA	Kerberos, AAA, EAP, SAML, 802.11
Federated AAA-based authentication	Yes	Yes	Yes
Advanced authorization	Yes	Yes	Yes
SSO	Yes (intra-organization)	Yes (intra-organization)	Yes (cross-layer intra-organization)
EAP lower-layer	Kerberos	PANA	802.11
Modifies AppS	No	No	No
Modifies Kerberos	Yes	No (*)	Yes
Modifies EAP methods	No	No	Yes
Modifies AAA	No (*)	No (*)	Yes

(*) Only when advanced authorization is not used.

FedKERB provides an architecture that integrates the Kerberos and AAA infrastructures by defining a new pre-authentication mechanism for Kerberos based on GSS-API and EAP. Besides, the architecture includes the possibility of performing advanced authorization after the authentication process, based on the transport of a SAML assertion containing identity information from the home organization's IdP to the service provider's Kerberos infrastructure. FedKERB provides a generic solution that accommodates all the requirements defined in Section 1.2 for an *identity federation beyond web* solution, including immediate support for SSO within the boundaries of the service provider (i.e. intra-organization). All of this makes FedKERB adaptable to most of the foreseen *identity federation beyond web* scenarios [31]. However, although FedKERB does not require application services to be modified, it does require the modification of the Kerberos pre-authentication process. While this would only imply to modify a single entity (i.e. the KDC), this requirement might still not be acceptable for some service providers and scenarios, where the deployed Kerberos infrastructure should be left intact. Moreover, if advanced authorization is implemented, the AAA infrastructure also requires slight modifications to obtain or generate the SAML statement and be transported to the

KDC.

PanaKERB defines an optimized alternative to FedKERB that does not require any modification to the existing Kerberos pre-authentication mechanisms. For achieving this objective, this contribution defines an architecture that uses an *out-of-band* protocol with native support for AAA infrastructures (in this case, PANA), to perform the federated authentication of the end user. As FedKERB, PanaKERB addresses the requirements defined in Section 1.2, and also provides intra-organization SSO. Therefore, PanaKERB would be applicable to the same scenarios as FedKERB does. The strength of PanaKERB resides on its simplicity, since it requires no modifications to the Kerberos protocol. However, this simplicity comes at the expense of introducing an additional protocol (i.e. PANA) and an entity (i.e. PAA) into the architecture. It is expected that this simplicity will make its adoption and deployment easier than for FedKERB, making of it a more suitable solution for those organizations that already have a Kerberos infrastructure deployed, and are reticent to modify it to incorporate any of the changes required by FedKERB. Additionally, PanaKERB also allows the use of the advanced authorization model defined for FedKERB. It is important to note that, in such a case, the Kerberos infrastructure would also need modifications, in order to provide the KDC with SAML processing capabilities.

Finally, EduKERB provides an architecture that integrates the Kerberos and AAA infrastructures by defining a new pre-authentication mechanism for Kerberos based on the results obtained after a successful federated authentication to the network service. As these two authentication processes happen at different layers of the *Open Systems Interconnection model* (OSI) [191] model (i.e. link and application layers), it has been called *cross-layer SSO*. EduKERB reduces the number of federated AAA-based authentication processes required to access application services, improving efficiency and optimizing the resource utilisation. Another relevant aspect of EduKERB when compared to FedKERB and PanaKERB is its applicability. Whereas FedKERB and PanaKERB have a generic applicability, EduKERB has been designed to take advantage of the specific network access authentication process defined for the *eduroam's* RADIUS infrastructure. Nevertheless, that does not preclude its utilisation in other scenarios if so is desired by the involved parties, and they commit to implement the required architecture. EduKERB is the contribution that requires more modifications to the existing infrastructures. In particular, it requires the modification of the Kerberos pre-authentication process, the EAP method (to transport the *edutoken* to the end user), and the AAA infrastructure (to implement the key distribution defined in [66]). Nevertheless, these modifications allow reducing the number of EAP authentications and the related overload in terms of network messages and

computational time. EduKERB also allows the use of the advanced authorization model defined for FedKERB and PanaKERB.

This thesis has also provided a performance and functional analysis of these three contributions, presented on Chapter 6. Specifically, this analysis has consisted on the development of a performance model, describing the different time-consuming operations performed on each proposal, the implementation of three prototypes to demonstrate their functional viability and feasibility, and the actual measurement of their performance based on these prototypes. Two major conclusions have been extracted from the results of this validation. On the one hand, it is demonstrated that the three proposals described in this thesis present similar execution times to the ones required for the network access service (which is used as a reference since it is the most extended use of AAA infrastructures). Moreover, as they have been implemented and executed in a real environment, their feasibility has also been demonstrated beyond a theoretical stand point. Finally, it has been confirmed that the three proposals perform in a very similar way. In fact, even though FedKERB has shown the bests results, the difference is not significant enough to use it as a selection criteria. Instead, that decision should be merely based on operational premises, such as which one fits better with the actual requirements of the target scenario, or which one has a minor impact on the existing infrastructure, if so is a requirement.

Finally, in order to highlight the increasing interest that *identity federations beyond the web* are receiving nowadays, it is important to mention that there is a current movement within the GÉANT community, and in particular within GN3Plus project [192], promoting the deployment of pilots of the Moonshot technology. This technology is being evaluated as an alternative to integrate existing and future application services within the GÉANT community. Whenever Moonshot support becomes more widespread, the contributions proposed in this thesis will be more likely to be deployed. These and other future aspects are dealt in the following section.

7.2 Future work

As commented throughout this work, this thesis provides several proposals for the *identity federations beyond the web* problem, enabling federated access and SSO to application services by means of Kerberos and AAA infrastructures. Without undermining its applicability or validity, during the realisation of this thesis some interesting research topics have been found worth being explored as future work, in order to provide further improvements in this area.

The following subsections provide a brief introduction on these topics, along with some preliminary ideas on how they could be addressed. They have been ordered according to their expected addressing term, from those to be done in a short-term to those envisioned in the long-term.

7.2.1 Deployment of the solutions in real scenarios

One of the more immediate questions one might ask after reading this thesis is: *when are these solutions envisioned to be deployed in real scenarios?* As commented above, there is an evident interest within the GÉANT community on the *identity federation beyond the web* area. Indeed, the GN3 and GN3Plus projects have put a lot of effort designing, testing, and deploying Moonshot. Moreover, the upcoming GN4 project, scoped within the H2020 programme [193], will aim to move Moonshot from the current status of *technology solution*, to a more ambitious status of *service*. That is, focusing on infrastructure deployment at NRENs/federations, to be able to offer Moonshot as a solution for non-web use cases under the eduGAIN brand.

Whenever Moonshot support becomes more widespread, the contributions proposed in this thesis will be more likely to be deployed. In particular, FedKERB will be the one more directly benefited, since it makes a direct use of the GSS-EAP mechanism to perform federated authentication with the Kerberos infrastructure. The rest of the contributions will be benefited as well, since the home AAA servers will be adapted to distribute SAML assertions to the service providers.

One interesting aspect has come up when working on the current Moonshot pilots within the GÉANT scope. The existing RADIUS infrastructure for eduroam is so far quite focused on the access to the network (Section 6.5). That is, many organizations and *National Research and Educational Network* (NRENs) implement filtering rules on their RADIUS proxies forbidding any RADIUS attribute that is not specifically listed to be passed-through. This is particularly relevant, as Moonshot/ABFAB defines some new RADIUS attributes that need to be conveyed from the service provider to the IdP, and vice-versa. When these attributes are not able to reach their destination, the federated authentication might fail due to the lack of information. For solving this issue two approaches can be followed: a) evolve the eduroam's RADIUS infrastructure to a more generic federation, allowing a larger number of attributes; or b) use an alternative and specific AAA-based federation, such as the one based on the *Trust Router* [149] technology (Section 7.2.4), also being currently tested in the GÉANT context. So far, this second alternative is gaining ground to the first one, as it does not requires to modify existing

running RADIUS proxies and federation agreements.

Hence, an interesting short-term future work line would consist of actively working on the deployment of *identity federation beyond the web* technologies, such as the ones presented on this PhD work, or Moonshot, in real environments (either using the *eduroam*'s infrastructure or a different one). This will help assessing the viability of those technologies, as well as discovering new gaps needing to be solved.

Although most application services support either GSS-API or SASL, some of them still do not do it. Hence, another short-term future work line related with real deployments is providing GSS-API support to those application services. This will extend the number of application services that support either Moonshot or Kerberos, as required for any of the proposals of this thesis. An example of this is the work being currently carried out in the *Cloud-ABFAB Federation Services in eduroam* (CLASSe) project [194], where the integration between OpenStack [195], a consolidated cloud server solution, and GSS-API is being defined [196]. In particular, the GSS-EAP mechanism is being tested, although using the GSS-KRB mechanism would be straightforward.

7.2.2 Use of HTTP instead of PANA as out-of-band protocol

PanaKERB proposal defines the use of PANA as its out-of-band authentication protocol. This decision is supported by its lightweight operation, as well as its ability to easily export the keying material derived from the EAP authentication. However, as described in Section 4.4.4, it requires configuring the firewalls of the organizations to allow PANA traffic. This might be seen as a drawback for some organizations that do not want to modify their network configuration rules. Even more, this is likely to happen whichever the out-of-band protocol is, unless that protocol is already allowed by most of the deployed firewalls. One of this widely-accepted protocols is HTTP.

Hence, a short-term future work line would analyse, design and implement of a solution to enable the use of HTTP as out-of-band protocol for Kerberos, instead of PANA. A promising alternative might include the use of the HTTP Negotiate Authentication Scheme [197] in combination with the GSS-EAP mechanism.

7.2.3 Inter-organization SSO

Another topic that has been foreseen as an interesting medium-term future work is the improvement of the SSO support, in order to allow inter-organization (i.e. service provider to service provider) SSO.

The proposals described in this work assume the deployment of a different KDC on each service provider. This means that the tickets obtained as a result of the bootstrapping process will enable the end user to benefit from SSO only within the boundaries of that specific provider. If the end user wants to access to an application service deployed on a different service provider, she needs to perform a new bootstrapping process with the KDC of that provider.

Hence, one interesting research topic would be analysing and designing solutions to extend these SSO boundaries out of a single organization. This would allow the end user to access to any of the application services provided within the federation by means of SSO. The main benefit would be twofold. On the one hand, reducing the amount of data exchanged between the service provider and the home organization. This information often needs to travel through a high number of intermediary proxies, resulting into a waste of both time and network bandwidth. On the other hand, a non-SSO authentication process often implies the use of asymmetric cryptographic, such as digital signature or Diffie-Hellman operations. This might have a noticeable impact on low-power devices, such as mobile phones, tablets, or sensors.

A possible approach for providing inter-organization SSO could be deploying the KDC at the home organization, instead of at the service provider. This KDC would provide the end user with some sort of generic ST that any application service within the federation can process. Another approach would be extending the GSS-EAP mechanism to incorporate support for the *EAP Re-authentication Protocol* (ERP) [198]. ERP defines extensions to EAP and the EAP keying hierarchy to allow re-authentication between the peer (end user) and an re-authentication server (e.g. local AAA server) in a single round-trip. In this way, the end user would be able to perform an ERP bootstrapping process with the KDC, instead of a full-blown EAP authentication, reducing the cost of the process to its minimal.

In particular, this inter-organization SSO topic is being currently investigated as part of CLASSe, were the University of Murcia is involved. Since that project is focused on the access to Cloud services, this concept is called *Cloud-to-cloud SSO*.

7.2.4 Dynamic AAA-based federations

Typically, trust relationships in AAA-federations have a static nature. Administrators establish pre-shared secrets between AAA servers, building a hierarchical structure (see chapter 2). That is, each node needs to be configured with the key (or the certificate) of each one of the next hops in the infrastructure. However, this structure limits the flexibility and scalability of AAA federations. For instance, if one of these AAA servers

fails, the federation might become fractured, and some of the organizations would not be reachable by others. Besides, becoming a member of the federation always requires of manual configuration, which is usually a laborious task.

Hence, another interesting medium-term research topic would be the analysis and design of dynamic establishment and management mechanisms for AAA federations, where trust relationships can be established in an automated fashion between members. For this purpose, current IP routing algorithms, such as *Border Gateway Protocol* (BGP) [199], could be used with the required adaptations.

In this line, the Moonshot project has started working on the *Trust Router* [149] concept, which provides a novel approach to establishing trust relationships between entities, which may significantly improve the flexibility, robustness and scalability of the federation. In particular, the *Trust Router* aims to distribute information about trust relationships across the members of a federation, by using protocols with many similarities to existing routing protocols. The distribution of this information allows a sending entity to discover the IP address of the recipient one, and the dynamic establishment of shared secrets between them, thus avoiding any requirement for technologies such as PKI. That is, the *Trust Router* releases AAA protocols from the routing functionality by establishing a point-to-point trust relationships between the AAA client and server. Then, the transport of AAA information is performed without the need of any intermediary AAA proxy. This avoids the requirement of trusting each single proxy server within the AAA infrastructure, as discussed in the security analysis of each contribution (i.e. sections 3.6, 4.4, and 5.5).

The *Trust Router* is currently in an early stage of specification in the IETF, and more precisely within the ABFAB WG. Besides, the Moonshot project is working on an implementation, which is so far functional, but still in a prototype state. This implementation is being piloted under the umbrella of the GN3Plus project, where the University of Murcia is a partner. The intention of this pilot is to assess its viability as a long-term alternative to the eduroam's RADIUS infrastructure.

The use of this dynamic AAA-based federations would help spreading the *identity federations beyond the web* technologies that are supported by AAA protocols, as it eases its management and usability. This would include Moonshot as well as the contributions that have been described in this thesis.

7.2.5 IDaaS

With the proliferation of cloud services, there has been an increasing interest in a new concept and trend in identity management, known as Identity-as-a-Service (IDaaS) [200].

This concept can be summarized as the outsourcing of identity management, such as authentication, provisioning, and attribute services from an organization to a cloud provider. Among the different services that this new type of service provider may offer we find: registration, identity verification, authentication, attribute-based authorization, SSO, federation, monitoring, roles and entitlement management, provisioning, and reporting.

There are two common models for IDaaS. On the one hand, *Cloud IDaaS*, where the IDaaS provider manages the whole identity service infrastructure and provides these services in a Software-as-a-Service (SaaS) fashion. In this case, there is no need for any kind of software, footprint or backend integration with the organization's IT infrastructure. This may fit better in small and medium organizations willing to avoid, as much as possible, the burden associated with the management of identities in their domain. On the other hand, *co-sourced IDaaS* is a variant where the IDaaS provider interacts directly with the backend IT infrastructure (directories, repositories, databases, etc.) managed and operated by the organization. This may fit better in medium to large organizations that do not want to lose control over its identity data.

The most important benefit of IDaaS [201, 202] is that it allows organizations to relay in experts the management and operations related with the usually complex identity management administration. Thus, it eases the life of the organizations by reducing the workload related with identity access management. Conversely, the most relevant drawback is that the organization is outsourcing critical functions and information to a third party. This may create certain level of reluctance in the usage of the IDaaS services.

Hence, a long-term future work line would consist of analysing, designing, and implementing approaches for integrating the IDaaS concept with the *identity federations beyond the web* technologies discussed in this thesis, as a way to simplify their deployment, and spread their usage.

Bibliography

[1] J. Postel and J. Reynolds. **File Transfer Protocol (FTP)**. IETF RFC 959, Oct 1985.

[2] B. Thomas. **On the Problem of Signature Authentication for Network Mail**. IETF RFC 644, July 1974.

[3] B. Goode. **Voice over Internet protocol (VoIP)**. *Proceedings of the IEEE*, 90(9):1495–1517, Sep 2002.

[4] R. Sandhu and P. Samarati. **Authentication, Access Control, and Audit**. *ACM Comput. Surv.*, 28(1):241–243, March 1996.

[5] T. Moses (Ed.). **eXtensible Access Control Markup Language (XACML) Version 2.0**. OASIS, February 2005.

[6] S. Cantor, J. Kemp, R. Philpott, and E. Maler (Eds.). **Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v2.0**. OASIS, March 2005.

[7] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. **The Kerberos Network Authentication Service (V5)**. IETF RFC 4120, July 2005.

[8] **OpenID Web Site**. Available from: `http://openid.net/`. Accessed 7 July 2014.

[9] E. Hammer-Lahav, D. Recordon, and D. Hardt. **The OAuth 2.0 Authorization Protocol**. IETF RFC 6749, October 2012.

[10] C. Rigney, S. Willens, A. Rubens, and W. Simpson. **Remote Authentication Dial In User Service (RADIUS)**. IETF RFC 2865, June 2000.

[11] P. Calhoun and J. Loughney. **Diameter Base Protocol**. IETF RFC 6733, October 2012.

[12] C. Shaer. **Single sign-on**. *Network Security*, 1995(8):11–15, 1995.

[13] A. Barth. **HTTP State Management Mechanism**. IETF RFC 6265, April 2011.

[14] **Google Accounts**. Available from: `https://accounts.google.com`. Accessed 7 July 2014.

[15] J. Torres, M. Nogueira, and G. Pujolle. **A Survey on Identity Management for the Future Network**. *Communications Surveys Tutorials, IEEE*, 15(2):787–802, February 2013.

[16] **SAML Single Sign-On (SSO) Service for Google Apps**. Available from: `https://developers.google.com/google-apps/sso/saml_reference_implementation`. Accessed 7 July 2014.

[17] **Amazon Help: About Single Sign On**. Available from: `http://www.amazon.com/gp/help/customer/display.html?nodeId=201221890`. Accessed 7 July 2014.

[18] **Sign up for Flickr with your Google Account!** Available from: `http://blog.flickr.net/en/2010/10/28/sign-up-for-flickr-with-your-google-account`. Accessed 7 July 2014.

[19] **Single Sign-on Services for Microsoft Enterprise Application Integration Solutions**. Available from: `http://download.microsoft.com/download/C/6/5/C65FF9FD-0ED7-47F6-91AB-000E6265EA5B/Enterprise_SSO_Whitepaper.doc`. Accessed 7 July 2014.

[20] **Authentication Authorisation Accounting Architecture Research Group**. Available from: `http://irtf.org/concluded/aaaarch`. Accessed 7 July 2014.

[21] **IEEE 802.11 Std., Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Network – Specific Requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications**. IEEE Standards, June 2007.

[22] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. **Extensible Authentication Protocol (EAP)**. IETF RFC 3748, June 2004.

[23] **Cx and Dx interfaces based on the Diameter protocol; Protocol details**. 3GPP TS 29.229 v12.0.0, 3rd Generation Partnership Project, June 2013.

[24] K. Wierenga and others. **DJ5.1.4: Inter-NREN Roaming Architecture.Description and Development Items**. Project Deliverable, September 2006.

[25] **iPass - Netserver configuration**. Available from: `http://help.ipass.com/doku.php?id=netserver_configuration`. Accessed 7 July 2014.

[26] **Mach - Uniquely Integrated Plus Portfolio**. Available from: `http://www.starhomemach.com/roaming/`. Accessed 7 July 2014.

[27] **Syniverse - Global Interstandard Roaming Solution**. Available from: `http://www.syniverse.com/products-services/product/global-interstandard-roaming-solution-uniroam`. Accessed 7 July 2014.

[28] **SIR - The RedIRIS Identity Service** . Available from: `http://www.rediris.es/sir/index.html.en`. Accessed 7 July 2014.

[29] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G- Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. **A View of Cloud Computing**. *Commun. ACM*, 53(4):50–58, April 2010.

[30] F. Berman, G. Fox, and A. JG. Hey. ***Grid computing: making the global infrastructure a reality***, volume 2. John Wiley and sons, 2003.

[31] R. Smith. **Application Bridging for Federated Access Beyond web (ABFAB) Use Cases**. IETF Internet Draft, draft-ietf-abfab-usecases-05, September 2012.

[32] **Application Bridging for Federated Access Beyond web (abfab) IETF Working Group**. Available from: `http://datatracker.ietf.org/wg/abfab/charter/`. Accessed 7 July 2014.

[33] J. Howlett, V. Nordh, and W. Singer. **Deliverable DS3.1.1: eduGAIN service definition and policy (Initial Draft)**. Project Deliverable, May 2010.

[34] T. Ylonen and C. Lonvick. **The Secure Shell (SSH) Protocol Architecture**. IETF RFC 4251, January 2006.

[35] **Public-Key Infrastructure (X.509)**. Available from: `http://www.ietf.org/html.charters/pkix-charter.html`. Accessed 7 July 2014.

[36] T. Dierks and C. Allen. **The TLS Protocol Version 1.0**. IETF RFC 2246, January 1999.

[37] J. Howlett and S. Hartman. **Project Moonshot**. Avalabile from: `https://community.ja.net/groups/moonshot`. Accessed 7 July 2014.

[38] S. Sakane, K. Kamada, S. Zrelli, and M. Ishiyama. **Problem Statement on the Cross-Realm Operation of Kerberos**. IETF RFC 5868, May 2010.

[39] **Janet home page**. Available from: `https://www.ja.net/`. Accessed 7 July 2014.

[40] J. Linn. **Generic Security Service Application Program Interface Version 2**. IETF RFC 2743, January 2000.

[41] **The Internet Engineering Task Force (IETF)**. Available from: `http://www.ietf.org/`. Accessed 7 July 2014.

[42] **GEANT Project** . Available from: `http://www.geant.net/pages/home.aspx`. Accessed 7 July 2014.

[43] D. Forsberg, Y. Ohba, B. Patil, H. Tschofenig, and A. Yegin. **Protocol for Carrying Authentication for Network Access (PANA)**. IETF RFC 5191, May 2008.

[44] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence. **Generic AAA Architecture**. IETF RFC 2903, August 2000.

[45] D. Verma. **Service Level Agreement in IP Networks**. In ***Proceedings of IEEE***, volume 92, pages 1382–1388, September 2004.

[46] C. Rigney. **RADIUS Accounting**. IETF RFC 2866, June 2000.

[47] B. Aboba and P. Calhoun. **RADIUS support for EAP**. IETF RFC 3579, June 2003.

[48] A. DeKok and G. Weber. **RADIUS Design Guidelines**. IETF RFC 6158, March 2011.

[49] A. DeKok. **Remote Authentication Dial-In User Service (RADIUS) Protocol Extensions**. IETF RFC 6929, April 2013.

[50] **RADIUS EXTensions (RADEXT) IETF Working Group**. Available from: `http://datatracker.ietf.org/wg/radext/charter/`. Accessed 7 July 2014.

[51] A. Perez-Mendez, R. Marin-Lopez, F. Pereniguez-Garcia, G. Lopez-Millan, A. DeKok, and D. Lopez. **Support of fragmentation of RADIUS packets**. IETF Internet Draft, draft-perez-radext-radius-fragmentation-07, July 2014.

[52] S. Winter, M. McCauley, S. Venaas, and K. Wierenga. **Transport Layer Security (TLS) Encryption for RADIUS**. IETF RFC 6614, May 2012.

[53] R. Stewart. **Stream Control Transmission Protocol**. IETF RFC 4960, September 2007.

[54] M. Tuexen, R. Seggelmann, and E. Rescorla. **Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)**. IETF RFC 6083, January 2011.

[55] S. Kent and K. Seo. **Security Architecture for the Internet Protocol**. IETF RFC 4301, December 2005.

[56] P. Calhoun, G. Zorn, D. Spence, and D. Mitton. **Diameter Network Access Server Application**. IETF RFC 4005, August 2005.

[57] P. Eronen, T. Hiller, and G. Zorn. **Diameter Extensible Authentication Protocol (EAP) Application**. IETF RFC 4072, August 2005.

[58] Il-Gon Kim and Jin-Young Choi. **Formal verification of PAP and EAP-MD5 protocols in wireless networks: FDR model checking**. In *Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on*, volume 2, pages 264–269 Vol.2, March 2004.

[59] D. Simon, B. Aboba, and R. Hurst. **The EAP-TLS Authentication Protocol**. IETF RFC 5216, March 2008.

[60] P. Funk and S. Blake-Wilson. **EAP Tunneled TLS Authentication Protocol (EAP-TTLS)**. IETF Internet Draft, draft-ietf-pppext-eap-ttls-05, July 2004.

[61] A. Palekar, D. Simon, J. Salowey, H. Zhou, G. Zorn, and S. Josefsson. **Protected EAP Protocol (PEAP) Version 2**. IETF Internet Draft, draft-josefsson-pppext-eap-10, October 2004.

[62] J. Arkko and H. Haverinen. **Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA)**. IETF RFC 4187, Jan. 2006.

[63] H. Haverinen and J. Salowey. **Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM)**. IETF RFC 4186, January 2006.

[64] H. Tschofenig, D. Kroeselberg, A. Pashalidis, Y. Ohba, and F. Bersani. **The Extensible Authentication Protocol-Internet Key Exchange Protocol version 2 (EAP-IKEv2) Method**. IETF RFC 5106, February 2008.

[65] B. Aboba, D. Simon, and P. Eronen. **Extensible Authentication Protocol Key Management Framework**. IETF RFC 5247, August 2008.

[66] J. Salowey, L. Dondeti, V. Narayanan, and M. Nakhjiri (2008). **Specification for the Derivation of Root Keys from an Extended Master Session Key (EMSK)**. IETF RFC 5295, August 2008.

[67] S. Winter and J. Salowey. **Update to the EAP Applicability Statement**. IETF Internet Draft, draft-winter-abfab-eapapplicability-02, October 2011.

[68] **IEEE 802.1X Std., Standards for Local and Metropolitan Area Networks: Port based Network Access Control**. IEEE Standards, 2004.

[69] W. A. Arbaugh, N. Shankar, and Y. Wan. **Your 802.11 Wireless Network has No Clothes**. *IEEE Wireless Communications*, vol. 9(1):pp. 44–51, November 2006.

[70] **IEEE 802.11i Std., Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Specification for Enhanced Security**. IEEE Standards, July 2005.

[71] Bruce Schneier. ***Applied cryptography: protocols, algorithms, and source code in C***. john wiley & sons, 2007.

[72] J. Katoen. **NIST FIPS PUB 197, Advanced Encryption Standard (AES)**. Available from: `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`. Accessed 7 July 2014, November 2001.

[73] **IEEE 802.16: Broadband Wireless Metropolitan Area Networks (MANs)**. IEEE Standard, 2012.

[74] **IEEE 802.16e Standard: Air Interface for Fixed and Mobile Broandband Wireless Access System**. IEEE Standard, February 2006.

[75] **IEEE 802.21: Draft IEEE Standard for Local and Metropolitan Area Networks: Media Independent Handover Services**. IEEE Standard, 2008.

[76] **IEEE 802.21a. Local and Metropolitan Area Networks: Media Independent Handover Services - Amendment for Security Extensions to Media Independent Handover Services and Protocol**. IEEE Standard, 2012.

[77] C. Kauffman. **Internet Key Exchange (IKEv2) Protocol**. IETF RFC 4306, December 2005.

[78] W. Stallings. SNMPv3: A security enhancement for SNMP. *Communications Surveys Tutorials, IEEE*, 1(1):2–17, January 1998.

[79] Tim Berners-Lee and Mark Fischetti. ***Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor***. HarperInformation, 2000.

[80] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. **Hypertext Transfer Protocol – HTTP/1.1**. IETF RFC 2616, June 1999.

[81] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. **HTTP Authentication: Basic and Digest Access Authentication**. IETF RFC 2617, June 1999.

[82] **OASIS Security Services Technical Committe**. Available from: `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`. Accessed 7 July 2014.

[83] S. Cantor, F. Hirsch, J. Kemp, R. Philpott, and E. Maler (Eds.). **Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0**. OASIS, March 2005.

[84] J. Hughes, S. Cantor, J. Hodges, F. Hirsch, P. Mishra, R. Philpott, and E. Maler (Eds.). **Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0**. OASIS, March 2005.

[85] **XML Security Working Group**. Available from: `http://www.w3.org/2008/xmlsec/`. Accessed 7 July 2014.

[86] S. Cantor, J. Moreh, R. Philpott, and E. Maler (Eds.). **Metadata for the OASIS Security Assertion Markup Language (SAML) v2.0**. OASIS, March 2005.

[87] **Internet2 - Shiboleth**. Available from: `http://shibboleth.internet2.edu`. Accessed 7 July 2014.

[88] **SimpleSAMLphp**. Available from: `simplesamlphp.org`. Accessed 7 July 2014.

[89] **Windows CardSpace**. Available from: `http://www.microsoft.com/windows/products/winfamily/cardspace`. Accessed 7 July 2014.

[90] D.J. Lutz and B. Stiller. **A Survey of Payment Approaches for Identity Federations in Focus of the SAML Technology**. *Communications Surveys Tutorials, IEEE*, 15(4):1979–1999, 2013.

[91] D. Reed and D. McAlpin (Eds.). **Extensible Resource Identifier (XRI) Syntax V2.0**. OASIS, November 2005.

[92] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. **Web Services Description Language (WSDL) 1.1**. Available from: `http://www.w3.org/TR/wsdl`. Accessed 7 July 2014, March 2001.

[93] M. Gudgin, M. Hadley, N. Mendelsohn, and J. Moreau. **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**. Available from: `http://www.w3.org/TR/soap12-part1/`. Accessed 7 July 2014, April 2007.

[94] K. Lawrence, C. Kaler, and A. Nadalin. **Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)**. Available from: `http://docs.oasis-open.org/wss/v1.1/wss-v1.`

1-spec-errata-os-SOAPMessageSecurity.pdf. Accessed 7 July 2014, November 2006.

[95] A. Nadalin et at. **WS-Trust 1.4**. Available from: http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/errata01/os/ws-trust-1.4-errata01-os-complete.pdf. Accessed 7 July 2014, April 2012.

[96] M. Goodner and A. Nadalin. **WS-Federation 1.2**. Available from: http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.pdf. Accessed 7 July 2014, May 2009.

[97] A. Nadalin et al. **WS-SecurityPolicy 1.3**. Available from: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/errata01/os/ws-securitypolicy-1.3-errata01-os-complete.pdf. Accessed 7 July 2014, April 2012.

[98] M. Jones, D. Balfanz, J. Bradley, Y. Goland, J. Panzer, N. Sakimura, and P. Tarjan. **JSON Web Token (JWT)**. IETF Internet Draft, draft-ietf-oauth-json-web-token-25, July 2014.

[99] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. **OpenID Connect Basic Client Profile 1.0 - draft 24**. Available from: http://openid.net/specs/openid-connect-basic-1_0.html. Accessed 7 July 2014, March 2013.

[100] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. **Network File System (NFS) version 4 Protocol**. IETF RFC 3530, April 2003.

[101] J. Klensin. **Simple Mail Transfer Protocol**. IETF RFC 5321, October 2008.

[102] M. Crispin. **INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1**. IETF RFC 3501, March 2003.

[103] S. Hartman and L. Zhu. **A Generalized Framework for Kerberos Pre-Authentication**. IETF RFC 6113, April 2011.

[104] **Common Authentication Technology Next Generation (kitten) IETF Working Group**. Available from: http://datatracker.ietf.org/wg/kitten/charter/. Accessed 7 July 2014.

[105] **SQL Server Protocols**. Available from: http://msdn.microsoft.com/en-us/library/ee210043(v=sql.105).aspx. Accessed 7 July 2014.

[106] L. Zhu, K. Jaganathan, and S. Hartman. **The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2**. IETF RFC 4121, July 2005.

[107] S. Hartman and J. Howlett. **A GSS-API Mechanism for the Extensible Authentication Protocol**. IETF RFC 7055, August 2012.

[108] S. Josefsson and N. Williams. **Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family**. IETF RFC 5801, July 2010.

[109] P. Saint-Andre. **Extensible Messaging and Presence Protocol (XMPP): Core**. IETF RFC 3920, October 2004.

[110] K. Zeilenga. **Lightweight Directory Access Protocol (LDAP): Technical Specification Road MapAn Information Model for Kerberos Version 5**. IETF RFC 4510, June 2006.

[111] P. Leach, C. Newman, and A. Melnikov. **Using Digest Authentication as a SASL Mechanism**. IETF RFC 6331, May 2000.

[112] L. Nerenberg. **The CRAM-MD5 SASL Mechanism**. IETF Internet Draft, draft-ietf-sasl-crammd5-10, July 2008.

[113] **Trans-European Research and Education Networking Association**. Available from: http://www.terena.org/. Accessed 7 July 2014.

[114] **eduroam: instant wireless access for visitors and Princeton travelers**. Available from: http://www.princeton.edu/oit/news/archive/?id=8799. Accessed 7 July 2014.

[115] M. Sánchez, G. López, O. Cánovas, and A.F. Gómez-Skarmeta. **Performance analysis of a cross-layer SSO mechanism for a roaming infrastructure**. *J. Netw. Comput. Appl.*, 32:808–823, July 2009.

[116] **eduPKI**. Available from: http://www.edupki.org. Accessed 7 July 2014.

[117] S. Hartman, T. Clancy, and K. Hoeper. **Channel Binding Support for EAP Methods**. IETF RFC 6677, July 2012.

[118] N. Williams, L. Johansson, S. Hartman, and S. Josefsson. **Generic Security Service Application Programming Interface (GSS-API) Naming Extensions**. IETF RFC 6680, August 2012.

[119] S. Hartman and J. Howlett. **Name Attributes for the GSS-API Extensible Authentication Protocol (EAP) Mechanism**. IETF RFC 7056, December 2013.

[120] B. Aboba, M. Beadles, J. Arkko, and P. Eronen. **The Network Access Identifier**. IETF RFC 4282, December 2005.

[121] **The MIT Kerberos Consortium**. Available from: `http://www.kerberos.org`. Accessed 7 July 2014.

[122] H. Tschofenig. **Bootstrapping Kerberos**. IETF Internet Draft, draft-tschofenig-pana-bootstrap-kerberos-00, July 2004.

[123] P.L. Hellwell, T.W. van der Horst, and K.E. Seamons. **Extensible Pre-Authentication in Kerberos**. In ***Proc. of the Twenty-Third Annual Conference on Computer Security Applications, 2007***, Miami Beach, FL, December 2007.

[124] **Microsoft MS-PAC: Privilege Attribute Certificate Data Structure (v20100711)**.

[125] **Introduction to Active Directory**. Available from: `http://technet.microsoft.com/en-us/library/cc758535(v=ws.10).aspx`. Accessed 7 July 2014.

[126] **Kerberos WG**. Available from: `http://www.ietf.org/html.charters/krb-wg-charter.html`. Accessed 7 July 2014.

[127] S. Sorce, T. Yu, and T. Hardjono. **A Generalized PAC for Kerberos V5**. IETF Internet Draft, draft-ietf-krb-wg-general-pac-01, Oct 2011.

[128] J. Hodges, J. Howlett, L. Johansson, and RL. Morgan. **Towards Kerberizing Web Identity and Services**. Kerberos consortium, December 2008.

[129] J. Howlett and T. Hardjono. **SAML V2.0 Kerberos Subject Confirmation Method Version 1.0**. Committee Draft 01, December 2009.

[130] J. Howlett and T. Hardjono. **SAML V2.0 Kerberos Attribute Profile Version 1.0**. Committee Draft 01, December 2009.

[131] N. Klingenstein, T. Scavo, J. Howlett, and T. Hardjono. **SAML V2.0 Kerberos Web Browser SSO Profile Version 1.0**. October 2009.

[132] J. Hodges and T. Wason (Eds.). **Liberty Architecture Overview. Version 1.1**. January 2003.

[133] R. Marín-López, F. Pereñíguez, G. López, and A. Pérez-Méndez. **Providing EAP-based Kerberos pre-authentication and advanced authorization for network federations**. *Computer Standards & Interfaces*, 33(5):494–504, 2011.

[134] J. Vollbrecht, P. Eronen, N. Petroni, and Y. Ohba. **State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator**. IETF RFC 4137, August 2005.

[135] A. Pérez-Méndez, R. Marin-Lopez, F. Pereniguez-Garcia, and G. Lopez-Millan. **GSS-API pre-authentication for Kerberos**. IETF Internet Draft, draft-perez-krb-wg-gss-preauth-02, September 2012.

[136] **MIT Kerberos**. Available from: `http://web.mit.edu/kerberos/`. Accessed 7 July 2014.

[137] **GSS preauth plugin**. Available from: `https://github.com/alejandro-perez/krb5.git`. Accessed 7 July 2014.

[138] G. Zorn. **Microsoft Vendor-specific RADIUS Attributes**. IETF RFC 2548, March 1999.

[139] J. Howlett and S. Hartman. **A RADIUS Attribute, Binding, Profiles, Name Identifier Format, and Confirmation Methods for SAML**. IETF Internet Draft, draft-ietf-abfab-aaa-saml-09, February 2014.

[140] W. Hommel. **Using XACML for Privacy Control in SAML-Based Identity Federations**. In Jana Dittmann, Stefan Katzenbeisser, and Andreas Uhl, editors, *Communications and Multimedia Security*, volume 3677 of *Lecture Notes in Computer Science*, pages 160–169. Springer Berlin Heidelberg, 2005.

[141] L. Zhu. **Additional Kerberos Naming Constraints**. IETF RFC 6111, February 2011.

[142] L. Zhu, P. Leach, and S. Hartman. **Anonymity Support for Kerberos**. IETF RFC 6112, April 2011.

[143] G.C. Kessler and D.E. Levine. **DENIAL-OF-SERVICE ATTACKS**. *Computer Security Handbook, Set*, 18:27, 2012.

[144] **RADIUS fragmentation Proof of Concept implementation**. Available from: `https://libra.inf.um.es/~alex/freeradius-server-2.1.12-fragmentation-support.tar.gz`. Accessed 7 July 2014.

[145] **FreeRadius**. Available from: `http://www.freeradius.org`. Accessed 7 July 2014.

[146] S. Hartman. **Larger Packets for RADIUS over TCP**. IETF Internet Draft, draft-ietf-radext-bigger-packets-01, July 2014.

[147] R. Housley and B. Aboba. **Guidance for Authentication, Authorization, and Accounting (AAA) Key Management**. IETF RFC 4962, July 2007.

[148] D.R. Lopez et al. **Deliverable DJ5.2.2,2: GÉANT2 Authorisation and Authentication Infrastructure (AAI) Architecture - second edition**. Project Deliverable, April 2007.

[149] M. Wasserman and S. Hartman. **Application Bridging for Federation Beyond the Web (ABFAB) Trust Router Protocol**. IETF Internet Draft, draft-mrw-abfab-trust-router-02, February 2014.

[150] A. Pérez-Méndez, R. Marín-López, F. Pereniguez-Garcia, and G. Lopez-Millan. **GSS-EAP pre-authentication for Kerberos**. IETF Internet Draft, draft-perez-abfab-eap-gss-preauth-01, March 2012.

[151] R. Marin-Lopez, F. Pereniguez-Garcia, Y. Ohba, and A.F. Skarmeta. **Network access security for the internet: protocol for carrying authentication for network access**. *IEEE Communications Magazine*, vol. 50(3):pp. 84–92, March 2012.

[152] G. Giaretta, R. Lopez, Y. Ohba, S. Thomson, and H. Tschofenig. **Usage Scenarios and Requirements for Multi-hop EAP Lower Layer**. IETF Internet Draft, draft-ohba-multihop-eap-00, February 2005.

[153] H. Krawczyk and M. Bellare. **HMAC: Keyed-Hashing for Message Authentication**. IETF RFC 2104, February 1997.

[154] W. Diffie and M.E. Hellman. **New directions in cryptography**. *Information Theory, IEEE Transactions on*, 22(6):644–654, Nov 1976.

[155] Y. Ohba and A. Yegin. **Definition of Master Key between PANA Client and Enforcement Point**. IETF RFC 5807, March 2010.

[156] L. Johansson. **An Information Model for Kerberos Version 5**. IETF RFC 6880, March 2013.

[157] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. **Network Configuration Protocol (NETCONF)**. IETF RFC 6241, June 2011.

[158] S. Josefsson. **The Base16, Base32, and Base 64 Data Encodings**. IETF RFC 4648, October 2006.

[159] D. Stanley, B. Aboba, and J. Walker. **Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs**. IETF RFC 4017, March 2005.

[160] J. Schaad, L. Zhu, and J. Altman. **Initial and Pass Through Authentication Using Kerberos V5 and the GSS-API (IAKERB)**. IETF Internet Draft, draft-ietf-kitten-iakerb-01, February 2014.

[161] Y. Wei. **Federated Cross-Layer Access**. IETF Internet Draft, draft-wei-abfab-fcla-02, March 2012.

[162] William Stallings. ***Network and internetwork security: principles and practice***, volume 1. Prentice Hall Upper Saddle River, NJ, 1995.

[163] G. Zorn, T. Zhang, J. Walker, and J. Salowey. **Cisco Vendor-Specific RADIUS Attributes for the Delivery of Keying Material**. April 2011.

[164] S. Sakane and M. Ishiyama. **Kerberos Options for DHCPv6**. IETF Internet Draft, draft-sakane-dhc-dhcpv6-kdc-option-10, November 2010.

[165] L. Zhu and B.Tung. **Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)**. IETF RFC 4556, June 2006.

[166] K. Hoeper and L. Chen. **Recommendation for EAP Methods Used in Wireless Network Access Authentication**. Standard document, September 2009.

[167] Y. Desmedt. Man-in-the-Middle Attack. In HenkC.A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 759–759. Springer US, 2011.

[168] J. Katoen. **NIST FIPS 180-2, Secure Hash Standard**. With Change Notice 1 dated Feb. 2004, August 2002.

[169] M. Abdalla and M. Bellare. **Increasing the Lifetime of a Key: A Comparative Analysis of the Security of Re-keying Techniques**. In ***Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology***, ASIACRYPT '00, pages 546–559, London, UK, 2000. Springer-Verlag.

[170] K. Hoeper, M. Nakhjiri, and Y. Ohba. **Distribution of EAP-Based Keys for Handover and Re-Authentication**. IETF RFC 5749, March 2010.

[171] **Automated Validation of Internet Security Protocols and Applications (AVISPA)**. Available from: `http://www.avispa-project.org/`. Accessed 7 July 2014.

[172] A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. Tobarra. **Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps**. In ***FMSE '08: Proceedings of the 6th ACM workshop on Formal methods in security engineering***, pages 1–10, New York, NY, USA, 2008.

[173] A. Ruiz-Martínez, C.I. Marín-López, L. Baño-López, and A.F. Gómez-Skarmeta. **A New Fair Non-repudiation Protocol for Secure Negotiation and Contract Signing**. *Journal of Universal Computer Science*, 15(3):555–584, 2009.

[174] **Deliverable D2.1: The High Level Protocol Specication Language**. AVISPA IST-2001-39252 Deliverable, August 2003.

[175] D. Dolev and A. Yao. **On the security of public key protocols**. *IEEE Transactions on Information Theory*, vol. 29(2):pp. 198–208, March 1983.

[176] L. Vigan. **Automated Security Protocol Analysis With the AVISPA Tool**. *Elsevier Electric Notes in Theoretical Computer Science*, 155:61–86, 2006.

[177] I. Cervesato. **The Dolev-Yao Intruder is the Most Powerful Attacker**. In ***Proc. of the Sixteenth Annual Symposium on Logic in Computer Science LICS'01***, pages 16–19. IEEE Computer Society Press. Short, 2001.

[178] A. Perez Mendez, P.J. Fernandez Ruiz, R. Marin Lopez, G. Martinez Perez, A.F. Gomez Skarmeta, and K. Taniuchi. **OpenIKEv2: Design and Implementation of an IKEv2 Solution**. *IEICE - Trans. Inf. Syst.*, E91-D(5):1319–1329, May 2008.

[179] **The TAO of IETF: A Novice's Guide to the Internet Engineering Task Force**. Available from: `https://www.ietf.org/tao.html`. Accessed 7 July 2014.

[180] F. Pereniguez, R. Marin-Lopez, G. Kambourakis, S. Gritzalis, and A.F. Gomez. **PrivaKERB: A user privacy framework for Kerberos**. *Elsevier Computers & Security*, 30(6-7):446–463, 2011.

[181] A. Pérez-Méndez, F. Pereñíguez-García, R. Marín-López, and G. López-Millán. **A cross-layer SSO solution for federating access to kerberized services in the eduroam/DAMe network**. *International Journal of Information Security*, 11(6):365–388, 2012.

[182] **OpenSSH**. Available from: `http://www.openssh.com`. Accessed 7 July 2014.

[183] **OpenPANA**. Available from: `https://sourceforge.net/projects/openpana/`. Accessed 7 July 2014.

[184] **WPA Supplicant**. Available from: `http://hostap.epitest.fi/wpa_supplicant/`. Accessed 7 July 2014.

[185] **Cardiff University**. Available from: `http://www.cardiff.ac.uk/`. Accessed 7 July 2014.

[186] P. Martinez-Julia, A.J. Jara, and A.F. Skarmeta. **GAIA Extended Research Infrastructure: Sensing, Connecting, and Processing the Real World**. In ***Proceedings of the TridentCom 2012***, pages 3–4. Springer, 2012.

[187] **XEN project**. Available from: `http://www.xen.org/`. Accessed 7 July 2014.

[188] **Debian**. Available from: http://www.debian.org/. Accessed 7 July 2014.

[189] **Wireshark**. Available from: http://www.wireshark.org. Accessed 7 July 2014.

[190] K. Raeburn. **Encryption and Checksum Specifications for Kerberos 5**. IETF RFC 3961, February 2005.

[191] H. Zimmermann. **OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection**. *Communications, IEEE Transactions on*, 28(4):425–432, April 1980.

[192] **GN3plus Open Call**. Available from: http://geant3.archive.geant.net/opencalls/overview/Documents/Open%20Call%20detailed%20text%20FINAL.pdf. Accessed 7 July 2014.

[193] **Horizon 2020 European Research and Innovation programme**. Available from: http://ec.europa.eu/programmes/horizon2020/. Accessed 7 July 2014.

[194] **Cloud-ABFAB Federation Services in eduroam**. Available from: http://www.um.es/CLASSe. Accessed 7 July 2014.

[195] **OpenStack Open Source Cloud Computing Software**. Available from: https://www.openstack.org/. Accessed 7 July 2014.

[196] **A Federated Keystone Identity Server**. Available from: http://sec.cs.kent.ac.uk/demos/keystone.html. Accessed 7 July 2014.

[197] K. Jaganathan and L. Zhu. **SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows**. IETF RFC 4559, June 2006.

[198] Z. Cao, B. He, Y. Shi, Q. Wu, and G. Zorn. **EAP Extensions for EAP Re-authentication Protocol (ERP)**. IETF RFC 6696, July 2012.

[199] Y. Rekhter, T. Li, and S. Hares. **A Border Gateway Protocol 4 (BGP-4)**. IETF RFC 4271, January 2006.

[200] F. Villavicencio. **Defining Identity as a Service**. Available from: http://blog.identropy.com/IAM-blog/bid/29162/Defining-Identity-as-a-Service. Accessed 7 July 2014.

[201] S. Deuby. **Outsourcing Your Identity with IDaaS**. Available from: http://windowsitpro.com/identity-management/outsourcing-your-identity-idaas. Accessed 7 July 2014.

[202] C. Bedell. **Understanding IDaaS: The benefits and risks of Identity as a Service**. Available from: http://goo.gl/QAyS7i. Accessed 7 July 2014.

Appendix A

List of Acronyms

AAA	Authentication, Authorization and Accounting
ABFAB	Application Bridging for Federated Access Beyond web
ADE	Authorization data elements
AP	Access Point
API	Application Programming Interface
AppS	Application Server
AS	Authentication Server
AVP	Attribute Value Pair
BGP	Border Gateway Protocol
BS	Base Station
CLASSe	Cloud-ABFAB Federation Services in eduroam
CNP	Configuration Network Protocol
DAMe	Deploying Authorization Mechanisms for federated services in eduroam.
DoS	Denial of Service
DSRK	Domain Specific Root Key
DSUSRK	Domain Specific and Usage Specific Root Key
EAP	Extensible Authentication Protocol
EAPOL	EAP over LAN
EMSK	Extended Master Session Key
EP	Enforcement Point
ERP	EAP Re-authentication Protocol
FAST	Flexible Authentication Secure Tunnelling
GSS-API	Generic Security Service Application Program Interface
GSS-EAP	A GSS-API Mechanism for EAP

GSS-KRB	Kerberos GSS-API Mechanism
GTK	Group Transient Key
H-AAA	Home organization's AAA server
HI	Home Institution
HLPSL	High Level Protocol Specification Language
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technologies
IdP	Identity provider
IDaaS	Identity-as-a-Service
IETF	Internet Engineering Task Force
EU	End User
IKE	Internet Key Exchange
IMAP	Internet Message Access Protocol
KDC	Key Distribution Center
KDF	Key Derivation Function
KMF	Key Management Framework
LAN	Local Area Network
MDS	Metadata Service
MICS	Media Independent Command Service
MIES	Media Independent Event Service
MIH	Media Independent Handover
MIIS	Media Independent Information Service
MSK	Master Session Key
NAI	Network Address Identifier
NAS	Network Access Server
NASREQ	Network Access Server Requirements Application
NREN	National Research and Educational Network
OSI	Open Systems Interconnection model
PAA	PANA Authentication Agent
PaC	PANA Client
PAC	Privilege Attribute Certificate
PAD	Principal Authorization Data
PAN	PANA-Auth-Answer
PANA	Protocol for Carrying Authentication for Network Access
PAR	PANA-Auth-Request

PDP	Policy Decision Point
PEP	Policy Enforcement Point
PEMK	PaC-EP Master Key
PKI	Public Key Infrastructure
PKINIT	Public Key Cryptography for Initial Authentication in Kerberos
PMK	Pairwise Master Key
PoA	Point of Attachment
PoS	Point of Service
PRF	Pseudo Random Function
PTK	Pairwise Transient Key
RFC	Request for Comments
RP	Relaying Party
RPC	Remote Procedure Call
SA	Security Association
SAML	Security Assertion Markup Language
SASL	Simple Authentication and Security Layer
SLA	Service Level Agreement
SP	Service Provider
SP-AAA	Service provider's AAA server
SS	Subscriber Station
SSH	Secure SHell
SSO	Single Sign-On
ST	Service Ticket
TEK	Traffic Encryption Key
TERENA	Trans-European Research and Education Networking Association
TGS	Ticket Granting Server
TGT	Ticket Granting Ticket
USRK	Usage Specific Root Key
VI	Visited Institution
VM	Virtual Machine
VoIP	Voice over IP
WEP	Wired Equivalent Privacy
WG	Working Group
WLAN	Wireless LAN
WPA	Wi-Fi Protected Access

A. List of Acronyms

WWW	World Wide Web
XACML	Extensible Access Control Markup Language

Appendix B

Example SAML assertion for the authorization model

This authorization model described in this document requires the generation and delivery of a SAMLv2 assertion from the home organization to the service provider right after the EAP authentication has been completed (Section 3.4.1). This SAML assertion contains a SAML *AuthnStatement*, which points out that the EU has been successfully authenticated, and provides a transient pseudonym in the *Subject* element. It may optionally contain some end user attributes, as discussed in Section 3.4.1, step 9.

This appendix provides an example of such an assertion, generated by the *University of Murcia*, for one of its end users. The assigned pseudonym is *pseudonym12345@um.es*. The assertion contains a single attribute called *studentcard* (not standardized), specifying she is a student.

```
<saml:Assertion xmlns:saml='urn:oasis:names:tc:SAML:2.0:assertion'
            IssueInstant='2014-07-17T18:45:10.738Z'
            ID='_3c39bc0fe7b13769cab2f6f45eba801b1245264310738'
            Version='2.0'>
   <saml:Issuer Format='urn:oasis:names:tc:SAML:2.0:nameid-format:entity'>
      https://www.um.es
   </saml:Issuer>

   <saml:Subject>
      <saml:NameID Format='urn:oasis:names:tc:SAML:1.1:nameid-format:transient'>
         pseudonym12345@um.es
      </saml:NameID>
   </saml:Subject>

   <saml:Conditions NotBefore='2014-07-17T18:45:10.738Z'
               NotOnOrAfter='2014-07-17T18:50:10.738Z'>
   </saml:Conditions>
```

```
   <saml:AuthnStatement AuthnInstant='2014-07-17T18:45:10.738Z'>
      <saml:AuthnContext>
         <saml:AuthnContextClassRef>
            urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified
         </saml:AuthnContextClassRef>
      </saml:AuthnContext>
   </saml:AuthnStatement>

   <saml:AttributeStatement>
      <saml:Attribute NameFormat='urn:oasis:names:tc:SAML:2.0:attrname-format:uri'
                  Name='studentcard'>
         <saml:AttributeValue>
            Student
         </saml:AttributeValue>
      </saml:Attribute>
   </saml:AttributeStatement>

</saml:Assertion>
```

Appendix C

Formal description of EduKERB

This appendix provides a detailed description of the message exchanges carried out in the different phases of this proposal, as well as of the processing that is performed by the participating entities. The notation is the same as described in Section 5.4.1.

C.1 Functions

The following list provides a brief description of the functions used in the description of the processing that is performed in the entities after the reception of a message.

check_eduToken(x). Verifies the eduToken "x". Checks validity of the signature, adequacy of the authentication context, etc.

compute_DH(x, y)(x). Obtains the shared secret resulting from a Diffie-Hellman key exchange, where "x" and "y" represent the exchangeable parts of the key pairs.

derive_dsrk(x, y). Derives a DSRK for the domain "y" from the EMSK indicated in "x". RFC 5295.

derive_dsusrk(x, y). Derives a DSUSRK for the application "y" from the DSRK indicated in "x". RFC 5295.

derive_emsk_name(x). Derives the EMSKName from the EMSK indicated in "x", following RFC 5295.

derive_reply_key(x). Derives the reply key from the DSUSRK "x", following the guidelines of RFC 5295.

DiffieHellman(). Generates a new Diffie-Hellman key pair.

EAP Identity Response(x). Generates a new EAP Identity Response packet, using "x" as user's NAI.

EAP Request(x, y). Generates a new EAP Request. "x" is the previously received EAP response. "y" is the EAP method being used.

EAP Response(x). Generates a new EAP Response. "x" is the received EAP request.

EAP Success(). Generates a new EAP Success packet.

generate_armor_key(x). Generates the armor key, using the information contained in "x". "x" can be an enc-part or an armor TGT.

generate_eduToken(x). Generates a new eduToken related with the end user identifier by "x".

generate_reply_key_from_DH(x). Generates the reply key from a Diffie Hellman shared secret "x", as described in RFC 4556, section 3.2.3.1

get_attributes(x, y). Obtains the collection of attributes of the end user identified by "x" that are suitable to be provided to the entity "y".

get_dsrk_by_emsk_name(x). Obtains the DSRK associated to the EMSKName "x".

get_emsk(). Returns the EMSK generated by the EAP method.

get_IDP(x). Obtains the IdP to be used for the end user identified by "x".

get_keying_material_key(x). Obtains the shared secret between the caller and the entity "x" used to generate the Keying-Material RADIUS attribute, as specified in RFC 6218.

get_local_radius_server(x). Obtains the address of the local RADIUS server of the entity "x".

get_mac_key(x). Obtains the shared secret between the caller and the entity "x" used to generate the Message-Authentication-Code RADIUS attribute, as specified in RFC 6218.

get_msk(). Returns the MSK generated by the EAP method.

get_next_radius_server(x). Obtains the next hop in the path where a RADIUS packet must be forwarded. "x" is the user's NAI.

get_tunnel_key(). Returns the tunnelled EAP method's tunnel key.

get_pseudonym(x). Obtains the pseudonym included in the eduToken "x".

get_radius_key(x). Obtains the RADIUS key used to protect the communications between the caller and the entity "x".

get_service_key(x). Obtains the shared key between the KDC and the service identified by "x".

get_timestamp(). Obtains a timestamp corresponding to the current time.

get_username_from_EAP_method(). Returns the NAI of the authenticated end user. If the EAP method is tunnelled, returns the user's NAI of the inner method.

KeyingMaterial(x). Generates a new Keying-Material RADIUS attribute (RFC 6218). "x" indicates the kind of attribute:

- request_dsrk → Request DSRK
- request_usrk → Request USRK/DSUSRK
- dsrk → Transport DSRK
- usrk → Transport USRK/DSUSRK

MAC-Randomizer(). Generates a new MAC-Randomizer RADIUS Attribute (RFC 6218).

Message-Authentication-Code(x). Generates a new Message-Authentication-Code attribute (as described in RFC 6218). "x" is the shared key used to generate the code.

SessionKey(). Generates a new Kerberos session key.

store_dsrk(x, y). Stores the association between the EMSKName "x" and the DSRK "y".

take_authz_decision(x, y). Takes an authorization decision based on the available policies for an user with the attributes "y" wanting to access a service identified by "x".

verify_timestamp(x). Verifies that the timestamp "x" is not too old.

C.2 Exchanges' detailed description

This section describes the message exchanges of the proposal. Specifically, the following exchanges are detailed:

- Network authentication, distribution of the eduToken and keying material.
- Kerberos pre-authentication and TGT acquisition.
- Authorization and ST acquisition.

C.2.1 Network authentication, distribution of the eduToken and keying material

1. System ⇒ AP: Start
   (a) eap_req = EAP Identity Request()
2. AP ⇒ EU: eap_req
   (a) eap_res = EAP Identity Response(anonymous@home)
3. EU ⇒ AP: eap_res
   (a) VR = get_local_radius_server(AP)
   (b) key_ap_vr = get_radius_key(VR)
4. AP ⇒ VR: [Access-Request(anonymous@home, eap_res)]$_{key\_ap\_vr}$
   (a) key_ap_vr = get_radius_key(VR)
   (b) HR = get_next_radius_server(anonymous@home)
   (c) key_vr_hr = get_radius_key(HR)
   (d) mackey_vr_hr = get_mac_key(HR)
   (e) keying_material = KeyingMaterial(request_dsrk)
   (f) randomizer = MAC-Randomizer()
   (g) mac = Message-Authentication-Code(mackey_vr_hr)
5. VR ⇒ HR: [[Access-Request(anonymous@home, eap_res, keying_material, randomizer,mac)]$_{mackey\_vr\_hr}$]$_{key\_vr\_vr}$
   (a) key_vr_hr = get_radius_key(HR)

   (b) mackey_vr_hr = get_mac_key(VR)

   (c) eap_req = EAP_Request(eap_rea, PEAP)

6. HR ⇒ VR: [Access-Challenge(anonymous@home, eap_req)]$_{key\_vr\_hr}$

7. VR ⇒ AP: [Access-Challenge(anonymous@home, eap_req)]$_{key\_ap\_vr}$

8. AP ⇒ EU: eap_req

   (a) eap_res = EAP Response(eap_req)

9. Repetition of the process from step 1 to 7. After several iterations, HR finalizes the authentication of the end user (after step 3).

10. VR ⇒ HR: [[Access-Request(anonymous@home, eap_res, keying_material, randomizer,mac)]$_{mackey\_vr\_hr}$]$_{key\_vr\_hr}$

    (a) username = get_username_from_EAP_method()

    (b) IDP = get_IDP(username)

11. HR ⇒ IDP: {{AuthnRequest(username)}$_{HR^{-1}}$}$_{IDP}$

    (a) eduToken = generate_eduToken(username)

12. IDP ⇒ HR: {{SAML Response(eduToken)}$_{IDP^{-1}}$}$_{HR}$

    (a) tk = get_tunnel_key()

    (b) msk = get_msk()

    (c) emsk = get_emsk()

    (d) emsk_name = get_emsk_name(emsk)

    (e) dsrk = derive_dsrk(emsk, VR)

    (f) keymat_key_vr_hr = get_keying_material_key(VR)

    (g) keying_material = KeyingMaterial(dsrk)

    (h) randomizer = Randomizer()

    (i) mac = Message-Authentication-Code(mackey_vr_hr)

    (j) eap_succ = EAP Success()

    (k) pseudonym = get_pseudonym(eduToken)

13. HR ⇒ EU: {eduToken}$_{tk}$

14. HR ⇒ VR: [[Access-Accept(emsk_name, eap_succ, pseudonym, msk, {keying_material(dsrk)}$_{keymat\_key\_vr\_hr}$, randomizer, mac)]$_{mackey\_vr\_hr}$]$_{key\_vr\_hr}$

    (a) store_dsrk(emsk_name, dsrk)

15. VR ⇒ AP: [Access-Accept(emsk_name, eap_succ, pseudonym, msk)]$_{key\_ap\_vr}$

16. AP ⇒ EU: eap_succ

    (a) msk = get_msk()

    (b) emsk = get_emsk()

C.2.2 Kerberos pre-authentication and TGT acquisition

1. System ⇒ EU: Start
   (a) dh_eu = DiffieHellman()
2. EU ⇒ KDC: AS_REQ(WELLKNOW:ANONYMOUS, PA_PK_AS_REP(dh_eu))
   (a) dh_kdc = DiffieHellman()
   (b) dh_shared_secret = compute_DH(dh_eu, dh_kdc)
   (c) pkinit_reply_key = generate_reply_key_from_DH(dh_shared_secret)
3. KDC ⇒ EU: AS_REP(WELLKNOW:ANONYMOUS, PA_PK_AS_REP(dh_kdc, $\{signed\_data\}_{KDC^{-1}}$), $\{armor\_TGT\}_{key\_as\_tgs}$, $\{enc\_part\}_{pkinit\_reply\_key}$)
   (a) armor_key = generate_armor_key(enc_part)
   (b) emsk_name = derive_emsk_name(emsk)
   (c) dsrk = derive_dsrk(emsk, VR)
   (d) dsusrk = derive_dsusrk(dsrk, KDC)
   (e) reply_key = derive_reply_key(dsusrk)
   (f) ts = get_timestamp()
4. EU ⇒ KDC: AS_REQ(WELLKNOWN:FEDERATED, PA_FX_FAST_REQUEST($\{armor\_TGT\}_{key\_as\_tgs}$, $\{enc\_fast\_req(PA\_EDUTOKEN(eduToken, emsk\_name, [ts]_{reply\_key}), req\_body)\}_{armor\_key}$))
   (a) armor_key = generate_armor_key(armor_TGT)
   (b) check_eduToken(eduToken)
   (c) VR = get_local_radius_server(KDC)
   (d) key_kdc_vr= get_radius_key(VR)
   (e) mackey_kdc_vr = get_mac_key(VR)
   (f) keying_material = Keying-Material(request_usrk)
   (g) randomizer = MAC-Randomizer()
   (h) mac = Message-Authentication-Code(mackey_kdc_vr)
5. KDC ⇒ VR: $[Access\text{-}Request(emsk\_name, keying\_material, randomizer, mac)]_{mackey\_kdc\_vr}]_{key\_kdc\_vr}$
   (a) dsrk = get_dsrk_by_emsk_name(emsk_name)
   (b) dsusrk = derive_dsusrk(dsrk, KDC)
   (c) keymat_key_kdc_vr = get_keying_material_key(KDC)
   (d) mackey_kdc_vr = get_mac_key(KDC)
   (e) keying-material = Keying-Material(dsusrk)
   (f) randomizer = MAC-Randomizer()
   (g) mac = Message-Authentication-Code(mackey_kdc_vr)
   (h) key_kdc_vr = get_radius_key(KDC)
6. VR ⇒ KDC: $[[Access\text{-}Accept(emsk\_name, \{keying\_material(dsusrk)\}_{keymat\_key\_kdc\_vr}, randomizer, mac)]_{mackey\_kdc\_vr}]_{key\_kdc\_vr}$

   (a) reply_key = derive_reply_key(dsusrk)

   (b) verify_timestamp(ts)

   (c) pseudonym = get_pseudonym(eduToken)

   (d) session_key = SessionKey()

7. KDC ⇒ EU: AS_REP(pseudonym, PA_FX_FAST_RESPONSE({enc_fast_rep}$_{armor\_key}$), {TGT(eduToken, session_key)}$_{key\_as\_tgs}$, {enc-part(session-key)}$_{reply\_key}$)

C.2.3 Authorization and ST acquisition

1. EU ⇒ KDC: TGS_REQ(service, {TGT(eduToken, session_key)}$_{key\_as\_tgs}$, {authenticator}$_{session\_key}$)

   (a) IDP = get_IDP(pseudonym)

   (b) pseudonym = get_pseudonym(eduToken)

2. 2. KDC ⇒ IDP: {{AttributeQuery(pseudonym, service)}$_{KDC^{-1}}$}$_{IDP}$

   (a) attributes = get_attributes(pseudonym, KDC)

3. IDP ⇒ KDC: {{SAML Response(attributes)}$_{IDP^{-1}}$}$_{KDC}$

4. KDC ⇒ PDP: {{Authorization Decision Query(service, attributes)}$_{KDC^{-1}}$}$_{PDP}$

   (a) decision = take_authz_decision(service, attributes)

5. PDP ⇒ KDC: {{Authorization Decision Response(decision, obligations)$_{PDP^{-1}}$}$_{KDC}$

   (a) service_session_key = SessionKey()

   (b) service_key = get_service_key(service)

6. KDC ⇒ EU: TGS_REP(pseudonym, {ST(service_session_key)}$_{service\_key}$, {enc-part(service_session_key)}$_{session\_key}$)

Appendix D

HLPSL specification of EduKERB

This appendix provides the HLPSL specification of EduKERB that has been used for the validation of the proposal.

D.1 Network authentication (phase 1) - Simplified version

```
role peer(P, A, S, D                    : agent,
          Kca, Kidp                     : public_key,
          H : hash_func,
          PRF : hash_func,
          CHAP_PRF : hash_func,
          Tranc : hash_func,
          KeyGen : hash_func,
          SND, RCV                      : channel (dy))
played_by P def=
  local
    UserId       : text,          % should not reveal user
    Version      : text,          % version of TLS protocol, presently v1.0
    SeID         : text,          % session id
    Np           : text,          % nonce from client
    Ns           : text,          % nonce from server
    CipherSuite  : text,          % TLS ciphersuites supplied by the peer
    Cipher       : text,          % TLS ciphersuite selected by server
    Ks           : public_key,    % from server
    Shd          : text,          % server-hello-done
    Ccs          : text,          % change-cipher-spec
    PMS          : text,          % pre-master-secret
    MS : hash(text.text.text),    % master-secret
    Finished : hash(hash(text.text.text).agent.agent.text.text.text),
    ClientK : hash(agent.text.text.hash(text.text.text)),  % client session key for encryption
    ServerK : hash(agent.text.text.hash(text.text.text)),  % server session key for encryption
    Txt          : text,          % string init. with "ttls challenge"
    UName        : text,          % NAI of client e.g. andy@realm
```

```
  ChapRs       : text,          % CHAP response
  Edutoken     : text,          % UMU: Edutoken
  Msk          : hash(hash(text.text.text)),                      % hash(MS)
  Emsk         : hash(hash(hash(text.text.text))),                % hash(MSK)
  Emskname     : hash(hash(hash(hash(text.text.text)))),          % hash(EMSK)
  Dsrk         : hash(hash(hash(hash(text.text.text))).agent),    % hash(EMSK.agent)
  State        : nat

const
  request_id   : text,
  respond_id   : text,
  start_ttls   : text,
  success      : text,
  sec_clientK,
  sec_serverK,
  sec_uname,
  np, ns       : protocol_id

init State := 0

transition
0. State   = 0
     /\ RCV(request_id)
   =|>
   State' := 1
     /\ UserId':= new()
     /\ SND(respond_id.UserId')

1. State   = 1
     /\ RCV(start_ttls)
   =|>
   State' := 2
     /\ Np' := new()
     /\ CipherSuite' := new()
     /\ SeID' := new()
     /\ Version' := new()
     /\ SND(Version'.SeID'.Np'.CipherSuite') % client_hello
     /\ witness(P,S,np,Np')

2. State   = 2
     /\ RCV(Version.SeID'.Ns'.Cipher'.       % server_hello
            {S.Ks'}_inv(Kca).                % server_certificate
            Shd')                            % server_hello_done
   =|>
   State' := 3
     /\ PMS' := new()
     /\ Ccs' := new()
     /\ MS'  := PRF(PMS'.Np.Ns')             % master secret
     /\ Finished' := H(MS'.P.S.Np.Cipher'.SeID)
     /\ ClientK'  := KeyGen(P.Np.Ns'.MS')
     /\ ServerK'  := KeyGen(S.Np.Ns'.MS')
     /\ SND({PMS'}_Ks'.                      % client_key_exchange
               Ccs'.                         % client_change_cipher_spec
               {Finished'}_ClientK')         % finished
```

```
      /\ secret(ClientK',sec_clientK,{P,S})
      /\ secret(ServerK',sec_serverK,{P,S})

  3. State   = 3
      /\ RCV(Ccs.{Finished}_ServerK)
    =|>
    State' := 4
      /\ Txt' := new()
      /\ ChapRs' := new()
      /\ UName' := new()
      /\ SND({UName'.
              Tranc(CHAP_PRF(MS.Txt'.Np.Ns).1.16).
              Tranc(CHAP_PRF(MS.Txt'.Np.Ns).17.17).
              ChapRs'
             }_ClientK)
      /\ secret(UName',sec_uname,{P,S})
      /\ request(P,S,ns,Ns)

  4. State   = 4
      /\ RCV(success)
    =|>
    State' := 5
      /\ Msk'      := KeyGen(MS)           % Derive the three keys, and Emskname
      /\ Emsk'     := KeyGen(Msk')
      /\ Emskname' := KeyGen(Emsk')
      /\ Dsrk'     := KeyGen(Emsk'.A)

  5. State   = 5
      /\ RCV({Edutoken'}_ServerK)
    =|>
    State' := 6
end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role auth (P, A, S                          : agent,
           Kas                              : symmetric_key, % to protect Keying-Material
           SND, RCV                         : channel(dy))
played_by A def=
  local
    Emskname     : hash(hash(hash(hash(text.text.text)))),          % hash(EMSK)
    Dsrk         : hash(hash(hash(hash(text.text.text))).agent),    % hash(EMSK.agent)
    UserId       : text,
    State           : nat

  const
    respond_id   : text,
    dsrk_req     : text,
    success      : text

  init State := 1

  transition
  1. State  = 1
      /\ RCV(respond_id.UserId')
    =|>
```

```
     State' := 2
       /\ SND(respond_id.dsrk_req.UserId') % Visited RADIUS server requests DSRK

  2. State  = 2
       /\ RCV(Emskname'.success.{Dsrk'}_Kas) % Visited RADIUS server receivs EMSKName and DSRK
     =|>
     State':= 3
     /\ SND(success)
end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role server (P, A, S                          : agent,
             Ks, Kca, Kidp                    : public_key,
             Kas                              : symmetric_key,
             H : hash_func,
             PRF : hash_func,
             CHAP_PRF : hash_func,
             Tranc : hash_func,
             KeyGen : hash_func,
             SND, RCV                         : channel (dy))
played_by S def=
  local
    UserId       : text,          % should not reveal user
    Version      : text,          % version of TLS protocol, presently v1.0
    SeID         : text,          % session id
    Np           : text,          % nonce from client
    Ns           : text,          % nonce from server
    CipherSuite  : text,          % TLS ciphersuites supplied by the peer
    Cipher       : text,          % TLS ciphersuite selected by server
    Shd          : text,          % server-hello-done
    Ccs          : text,          % change-cipher-spec
    PMS          : text,          % pre-master-secret
    MS : hash(text.text.text),    % master-secret
    Finished : hash(hash(text.text.text).agent.agent.text.text.text),
    ClientK : hash(agent.text.text.hash(text.text.text)),  % client session key for encryption
    ServerK : hash(agent.text.text.hash(text.text.text)),  % server session key for encryption
    Txt          : text,          % string init. with "ttls challenge"
    UName        : text,          % NAI of client e.g. andy@realm
    ChapRs       : text,          % CHAP response
    Edutoken     : text,                                          % Edutoken
    Msk          : hash(hash(text.text.text)),                    % hash(MS)
    Emsk         : hash(hash(hash(text.text.text))),              % hash(MSK)
    Emskname     : hash(hash(hash(hash(text.text.text)))),        % hash(EMSK)
    Dsrk         : hash(hash(hash(hash(text.text.text))).agent),  % hash(EMSK.agent)
    State        : nat

  const
    request_id   : text,
    respond_id   : text,
    dsrk_req     : text,
    start_ttls   : text,
    success      : text,
    np, ns              : protocol_id,
    sec_emsk, sec_dsrk  : protocol_id
```

```
init State := 0

transition
0. State   = 0
     /\ RCV(start)
   =|>
   State' := 1
     /\ SND(request_id)

1. State   = 1
     /\ RCV(respond_id.dsrk_req.UserId')
   =|>
   State' := 2
     /\ SND(start_ttls)

2. State   = 2
     /\ RCV(Version'.SeID'.Np'.CipherSuite')  % client_hello
   =|>
   State' := 3
     /\ Ns' := new()
     /\ Shd' := new()
     /\ Cipher' := new()
     /\ SND(Version'.SeID'.Ns'.Cipher'.       % server_hello
            {S.Ks}_inv(Kca).                  % server_certificate
            Shd')                             % server_hello_done
     /\ witness(S,P,ns,Ns')

3. State    = 3
     /\ RCV({PMS'}_Ks.                        % client_key_exchange
            Ccs'.                             % client_change_cipher_spec
            {Finished'}_ClientK')             % finished
     /\ MS' = PRF(PMS'.Np.Ns)                 % master secret
     /\ Finished' = H(MS'.P.S.Np.Cipher'.SeID)
     /\ ClientK'  = KeyGen(P.Np.Ns.MS')
   =|>
   State' := 4
     /\ ServerK'  := KeyGen(S.Np.Ns.MS')
     /\ SND(Ccs'.                             % server_change_cipher_spec
            {Finished'}_ServerK')             % finished

4. State  = 4
     /\ RCV({UName'.
             Tranc(CHAP_PRF(MS.Txt'.Np.Ns).1.16).
             Tranc(CHAP_PRF(MS.Txt'.Np.Ns).17.17).
             ChapRs'
            }_ClientK)
   =|>
   State':= 5
     /\ Edutoken'  := new()                   % Edutoken generation simulation
     /\ Msk'       := KeyGen(MS)
     /\ Emsk'       := KeyGen(Msk')
     /\ Dsrk'       := KeyGen(Emsk'.A)
     /\ Emskname'  := H(Emsk')
     /\ SND({Edutoken'}_ServerK)                  % Edutoken (to the EU)
```

```
       /\ SND(Emskname'.success.{Dsrk'}_Kas) % Success + EMSKName + DSRK (to the visited RADIUS)
       /\ secret(Emsk',sec_emsk,{P,S})
       /\ secret(Dsrk',sec_dsrk,{P,S,A})
       /\ request(S,P,np,Np)
       /\ secret(Edutoken',sec_edutoken,{S,P})
end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role session(P, A, S, D                      : agent,
             Ks, Kca, Kidp                   : public_key,
             Kas                             : symmetric_key,
             H : hash_func,
             PRF : hash_func,
             CHAP_PRF : hash_func,
             Tranc : hash_func,
             KeyGen : hash_func)
def=
  local
    SNDP, RCVP, SNDA, RCVA, SNDS, RCVS, SNDI, RCVI  : channel (dy)

  composition
           peer(P,A,S,D,Kca,Kidp,H,PRF,CHAP_PRF,Tranc,KeyGen,SNDP,RCVP)
        /\ auth(P,A,S,Kas,SNDA,RCVA)
        /\ server(P,A,S,Ks,Kca,Kidp,Kas,H,PRF,CHAP_PRF,Tranc,KeyGen,SNDS,RCVS)
end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role environment() def=
   const p, a, s, idp           : agent,
         ks, kca, kidp          : public_key,
         kas, kis               : symmetric_key,              % KeyingMaterial Key
         h, prf, chapprf        : hash_func,
         tranc, keygen          : hash_func

   intruder_knowledge = {p, a, s, idp, ks, kca, kidp,
                         h, prf, chapprf, tranc, keygen,
                         kca, kis
                        }

   composition
        session(p,a,s,idp,ks,kca,kidp,kas,h,prf,chapprf,tranc,keygen)
%     /\ session(p,a,s,idp,ks,kca,kidp,kas,h,prf,chapprf,tranc,keygen)
%     /\ session(i,a,s,idp,ks,kca,kidp,kas,h,prf,chapprf,tranc,keygen)
%     /\ session(p,i,s,idp,ks,kca,kidp,kis,h,prf,chapprf,tranc,keygen)
end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
goal
        secrecy_of sec_clientK, sec_serverK, sec_uname, sec_dsrk, sec_emsk, sec_edutoken
        %Peer authenticates Server on ns
        authentication_on ns
        %Server authenticates Peer on np
        authentication_on np
end goal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
environment()
```

D.2 Kerberos authentication, autorization & services access (phases 2, 3, and 4) - Simplified version

```
role authenticationServer(
            A,C,G,R    : agent,
            Kag,Kar  : symmetric_key,
            Ka               : public_key,
            Keygen     : hash_func,
            DHG        :text,
            SND, RCV : channel(dy),
            L        : text set)
played_by A
def=
  local State    : nat,
        N1       : text,
        U        : agent,
        T0       : text,
        Kcg      : symmetric_key,
        T1start  : text,
        T1expire : text,
        Padata   : text.{agent.text}_symmetric_key.text,
        Emskname : text,           % EMSKName
        Dsusrk   : hash(symmetric_key),
        Replykey : hash(hash(symmetric_key)),
        Edutoken : text,
        DHB      : text,
        KE       : message,
        Karmor   : hash(message)

  const
      dsusrk_req: text,
      sec_a_Kcg : protocol_id

  init  State := 10

  transition
    10. State  = 10 /\ RCV(KE')
        =|>
       State':= 11 /\ DHB' := new()
                   /\ Karmor' := Keygen(exp(KE', DHB'))
                   /\ SND({exp(DHG, DHB')}_inv(Ka))

    11. State  = 11 /\ RCV({U'.G.N1'.Padata'}_Karmor)
        =|>
       State':= 12 /\ SND(A.{dsusrk_req}_Kar)

    12. State  = 12 /\ RCV(R.{Dsusrk'}_Kar)
                       % Check if padata is encrypted with Replykey
                   /\ Padata = Emskname'.{C.T0'}_Keygen(Dsusrk').Edutoken'
                   /\ not(in(T0',L))
      =|>
       State':= 13 /\ Kcg' := new()
                   /\ Replykey' := Keygen(Dsusrk')
```

```
                    /\ T1start' := new()
                    /\ T1expire' := new()
                    /\ SND(U.
                           {U.C.G.Kcg'.T1start'.T1expire'.Edutoken'}_Kag.          % TGT
                           {G.Kcg'.T1start'.T1expire'.N1}_Replykey')               % enc-part
                    /\ L' := cons(T0',L)
                    /\ witness(A,C,n1,Kcg'.N1)
                    /\ request(A,C,t0,T0')
                    /\ secret(Kcg',sec_a_Kcg,{A,C,G})
  end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  role ticketGrantingServer (
              G,S,C,A      : agent,
              Kag,Kgs      : symmetric_key,
              Kidp,Kg,Kpdp      : public_key,
              SND,RCV      : channel(dy),
              L            : text set)
  played_by G
  def=
    local State    : nat,
          N2       : text,
          U        : agent,
          Kcg      : symmetric_key,
          Kcs      : symmetric_key,
          T1start,T1expire  : text,
          T2start, T2expire : text,
          T1       : text,
          Edutoken : text,
          Decision : text,
          Attributes: text

    const
        attribute_req       : text,
        decision_req        : text,
        sec_t_Kcg, sec_t_Kcs : protocol_id

    init  State := 21

    transition
      1. State  = 21 /\ RCV(S.N2'.
                           {U'.C.G.Kcg'.T1start'.T1expire'.Edutoken'}_Kag.
                           {C.T1'}_Kcg')
                    /\ not(in(T1',L))
         =|>
         State':= 22
                    /\ Kcs' := new()
                    /\ T2start' := new()
                    /\ T2expire' := new()
                    /\ SND(U'.
                           {U'.C.S.Kcs'.T2start'.T2expire'}_Kgs.
                           {S.Kcs'.T2start'.T2expire'.N2'}_Kcg')
                    /\ L' := cons(T1',L)
                    /\ request(G,C,t1,T1')
                    /\ secret(Kcg',sec_t_Kcg,{A,C,G})
```

```
                    /\ witness(G,C,n2,Kcs'.N2')
                    /\ secret(Kcs',sec_t_Kcs,{G,C,S})
  end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  role server( S,C,G    : agent,
               Kgs      : symmetric_key,
               SND, RCV : channel(dy),
               L        : text set)
  played_by S
  def=
    local State    : nat,
          U        : agent,
          Kcs      : symmetric_key,
          T2expire : text,
          T2start  : text,
          T2       : text

    const sec_s_Kcs : protocol_id

    init  State := 31

    transition
      1. State  = 31 /\ RCV({U'.C.S.Kcs'.T2start'.T2expire'}_Kgs.
                           {C.T2'}_Kcs')
                     /\ not(in(T2',L)) =|>
         State':= 32 /\ SND({T2'}_Kcs')
                     /\ L' := cons(T2',L)
                     /\ request(S,C,t2a,T2')
                     /\ witness(S,C,t2b,T2')
                     /\ secret(Kcs',sec_s_Kcs,{G,C,S})
  end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  role client( C,G,S,A,R    : agent,
               U            : agent,
               Dsrk         : symmetric_key,
               Ka           : public_key,
               Keygen       : hash_func,
               Edutoken     : text,
               DHG            : text,
               SND,RCV      : channel(dy))
  played_by C
  def=
    local State    : nat,
          Kcs      : symmetric_key,
          T1expire : text,
          T2expire : text,
          T1start  : text,
          T2start  : text,
          Kcg      : symmetric_key,
          Tcg,Tcs  : {agent.agent.agent.symmetric_key.text.text}_symmetric_key,
          T0,T1,T2 : text,
          N1,N2    : text,
          Padata   : text.{agent.text}_symmetric_key.text,
          Emskname : text,           % EMSKName
```

```
          Dsusrk   : hash(symmetric_key),
          Replykey : hash(hash(symmetric_key)),
          DHA      : text,
          KE       : message,
          Karmor   : hash(message)

    const sec_c_Kcg, sec_c_Kcs, sec_Emskname, sec_Dsusrk, sec_Replykey,
          sec_Karmor, sec_Edutoken : protocol_id

    init  State := 0

    transition
      0. State = 0 /\ RCV(start) =|>
         State' := 1 /\ DHA' := new()
                     /\ SND(exp(DHG, DHA'))        % Send DH exchange

      1. State  = 1 /\ RCV({KE'}_inv(Ka))  =|>
         State':= 2 /\ Karmor' := Keygen(exp(KE',DHA))
                    /\ N1' := new()
                    /\ T0' := new()
                    /\ Emskname' := new()
                    /\ Dsusrk' := Keygen(Dsrk)
                    /\ Replykey' := Keygen(Dsusrk')
                    /\ Padata' := Emskname'.{C.T0'}_Replykey'.Edutoken
                    /\ SND({U.G.N1'.Padata'}_Karmor')                    % Karmor == FAST
                    /\ witness(C,A,t0,T0')
                    /\ secret(Emskname', sec_Emskname, {C,A,R})
                    /\ secret(Dsusrk', sec_Dsusrk, {C,A,R})
                    /\ secret(Replykey', sec_Replykey, {C,A})
                    /\ secret(Karmor, sec_Karmor, {C,A})
                    /\ secret(Edutoken, sec_Edutoken, {C,A})


      2. State  = 2 /\ RCV(U.Tcg'.{G.Kcg'.T1start'.T1expire'.N1}_Replykey) =|>
         State':= 3 /\ N2' := new()
                    /\ T1' := new()
                    /\ SND(S.N2'.Tcg'.{C.T1'}_Kcg')
                    /\ witness(C,G,t1,T1')
                    /\ request(C,A,n1,Kcg'.N1)
                    /\ secret(Kcg',sec_c_Kcg,{A,C,G})

      3. State  = 3 /\ RCV(U.Tcs'.{S.Kcs'.T2start'.T2expire'.N2}_Kcg) =|>
         State':= 4 /\ T2' := new()
                    /\ SND(Tcs'.{C.T2'}_Kcs')
                    /\ witness(C,S,t2a,T2')
                    /\ request(C,G,n2,Kcs'.N2)
                    /\ secret(Kcs',sec_c_Kcs,{G,C,S})

      4. State  = 4 /\ RCV({T2}_Kcs) =|>
         State':= 5
                    /\ request(C,S,t2b,T2)
  end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  role radius(A,R           : agent,
```

```
              Kar          : symmetric_key,
              Dsrk         : symmetric_key,
              Keygen       : hash_func,
              SND,RCV      : channel(dy))
  played_by R
  def=

    local State    : nat,
          Emskname : text,           % EMSKName
          Dsusrk   : hash(symmetric_key)

    const
         dsusrk_req: text,
         sec_c_Kcg, sec_c_Kcs : protocol_id

    init  State := 1

    transition
      1. State  = 1 /\ RCV( A.{dsusrk_req}_Kar )  =|>
         State':= 2 /\ Dsusrk' := Keygen(Dsrk)
                    /\ SND(R.{Dsusrk'}_Kar)
  end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role session(A,G,C,S,R,D,P             : agent,
             U                         : agent,
             Kgs,Kag,Kar,Dsrk   : symmetric_key,
             Ka,Kidp,Kg,Kpdp              : public_key,
             LS,LG,LA                  : text set,
             Keygen                    : hash_func,
             Edutoken, Dhg             : text)
def=
        local
             SendC,ReceiveC            : channel (dy),
             SendS,ReceiveS            : channel (dy),
             SendG,ReceiveG            : channel (dy),
             SendA,ReceiveA            : channel (dy),
             SendR,ReceiveR            : channel (dy),
             SendD,ReceiveD            : channel (dy),
             SendP,ReceiveP            : channel (dy)
  composition
           client(C,G,S,A,R,U,Dsrk,Ka,Keygen,Edutoken,Dhg,SendC,ReceiveC)
       /\  server(S,C,G,Kgs,SendS,ReceiveS,LS)
       /\  ticketGrantingServer(G,S,C,A,Kag,Kgs,Kidp,Kg,Kpdp,SendG,ReceiveG,LG)
       /\  authenticationServer(A,C,G,R,Kag,Kar,Ka,Keygen,Dhg,SendA,ReceiveA,LA)
       /\  radius(A,R,Kar,Dsrk,Keygen,SendR,ReceiveR)
end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role environment() def=
  local
     LS, LG, LA : text set

  const a,g,c,s,r,d,p           : agent,
        kgi,kgs,kag,kar,dsrk    : symmetric_key,
        u3,u1,u2                : agent,
```

```
    keygen                  : hash_func,
    ka,kidp, kg, kpdp               : public_key,
        t0,t1,t2a,t2b,n1,n2     : protocol_id,
        edutoken, dhg               : text

  init LS := {} /\ LG := {} /\ LA := {}

  intruder_knowledge = {a,g,c,s,r,u1,u2,keygen,u3,kgi,LA,kidp,kpdp,kg,dhg,ka}

  composition
    % normal session
    session(a,g,c,s,r,d,p,u1,kgs,kag,kar,dsrk,ka,kidp,kg,kpdp,LS,LG,LA,keygen,edutoken,dhg)
% normal session
% /\ session(a,g,c,s,r,d,p,u1,kgs,kag,kar,dsrk,ka,kidp,kg,kpdp,LS,LG,LA,keygen,edutoken,dhg)
% i is Client
% /\ session(a,g,i,s,r,d,p,u2,kgs,kag,kar,dsrk,ka,kidp,kg,kpdp,LS,LG,LA,keygen,edutoken,dhg)
% i is Server
% /\ session(a,g,c,i,r,d,p,u3,kgi,kag,kar,dsrk,ka,kidp,kg,kpdp,LS,LG,LA,keygen,edutoken,dhg)
end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
goal
  secrecy_of sec_a_Kcg,
             sec_t_Kcg, sec_t_Kcs,
             sec_s_Kcs,
             sec_c_Kcg, sec_c_Kcs, % addresses G10
             sec_Emskname, sec_Dsusrk, sec_Replykey, sec_Karmor, sec_Edutoken

  %Client authenticates AuthenticationServer on n1
  authentication_on n1 % addresses G1, G3, G7, and G8
  %Client authenticates TicketGrantingServer on n2
  authentication_on n2 % addresses G1, G3, G7, and G8
  %Client authenticates Server on t2b
  authentication_on t2b % addresses G1, G2, and G3
  %Server authenticates Client on t2a
  authentication_on t2a % addresses G1, G2, and G3
  %TicketGrantingServer authenticates Client on t1
  authentication_on t1 % addresses G1, G2, and G3
  %AuthenticationServer authenticates Client on t0
  authentication_on t0 % addresses G1, G2, and G3
end goal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
environment()
```