Design and implementation of a trusted third-party based cross-realm AAA system

By Saber Zrelli

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Yoichi Shinoda

March, 2006

Design and implementation of a trusted third-party based cross-realm AAA system

By Saber Zrelli (410067)

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Yoichi Shinoda

and approved by
Professor Yoichi Shinoda
Associate Professor Yasuo Tan
Professor Hong Shen

February, 2006 (Submitted)

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Inter-realm agreements : perpetual expansion

In the last few years we have been witnesses of the increasing popularity of the new generation Internet, referred to as IPv6, as well as the expansion of home networks combined with the advances in networking capable portable devices such as PDAs and 3G cellular phones. In this new area of technological novelties, several applications and new services are expected to be developed for personal devices and home appliances. Such services would be innovative thanks to the features of the new IPv6 protocol, which include: unlimited address range, seamless node mobility [1] and network mobility support [2]. These facilities allow the deployment of new categories of services. In the other hand, they impose more strict management approach and a clear definition of the business model. For this reason, more and more administrative domains or realms would be defined. The multi-realm separation of the global Internet allows the delegation of the authority amongst several entities. Inter-realm agreements and protocols come then in action, to provide interconnection and distributed management of the services.

When considering home networking for instance, each home network would be equivalent to a separate realm. Each householder can control all the appliances in his realm. Some friends of the house holder can be allowed to access the services in the home network, while access control and protection of the resources is maintained in order to avoid malicious users.

In another context, Internet Service Providers (ISP)s are divided into different administrative domains. As for cellular telephony providers, the Internet service providers would need to maintain inter-realm agreements, these agreements would allow users of a certain ISP, to obtain network access in an area covered by another ISP.

1.2 Motivations and target application domain

The readiness of the IPv6 protocol and the advances in personal computing and mobile devices will cause new services and markets to flourish and to attract many operators worldwide. We explain in the following paragraph, the motivation of this work and its target application domain

considering these emerging markets.

IPv6 is a new version of the Internet Protocol that is designed to be an evolutionary step from IPv4. It builds upon the architecture that made IPv4 successful in the Internet. It is designed to solve the growth problem that the Internet is encountering. The new generation Internet will produce new types of applications and markets. And reciprocally, the growth of the new generation Internet is expected to be driven by these emerging markets.

Nomadic personal computing devices seem certain to become the actors of an emerging market as their prices drop and their capabilities increase. A Key capability is that they will be networked. These types of devices will become consumer devices and will replace current generation of cellular phones, pagers and personal digital assistants.

Another type of applications that could benefit from the next Internet generation, consist on device control systems. These systems are implemented as control networks in households or in the industry. Such applications are used for example to control everyday devices such as lighting, heating and cooling equipments.

The mobile personal computing offer attractive market for the network operators and service providers since it is a completely new domain of application that was not possible to exploit, due to the shortcomings of the actual Internet protocols. In the other hand, the networked control of small devices can result on extremely large cost savings.

The main goal of this thesis is to design a AAA framework that meets the requirements of these new markets.

1.3 Need for AAA with cross-realm support

In addition to the obvious requirements of an Internet protocol that can support large scale routing and addressing, the emerging markets will need deployment solutions that offer security and AAA functionalities while maintaining a low overhead and ease of use. The markets of personal computing and control networks requires easy deployment of the services with minimal configurations and maintenance costs. Furthermore, the client side requirements in term of computational power must be reasonable enough to support the average hardware capabilities of small devices.

Authentication and authorization are vital actors in the scene. AAA (Authentication Authorization and Accounting) operations allow service providers to control access to their networks and application services. AAA frameworks are required to support cross-realm service access in order to allows clients to obtain credentials to access services deployed by foreign realms.

The authentication protocols are the mean by which users and/or devices can prove their identity. Based on a verified identity, the authorization system distributes the privileges to the clients. The clients can then access the services and accounting protocols can be used to count service units and get statistics of service usage.

In parallel to the Internet and network services advances, the research in the field of Authentication and authorization for access control and network security was also following noticeable improvements. Such protocols are being designed and improved to fulfill the new requirements and the new constraints imposed by modern business models and service frameworks.

1.4 Purpose of this thesis

We noticed that some of the existing AAA frameworks are not suitable when considering client devices with low computational capabilities and limited memory. This is due to the large footprint[1] of the code required to run on the client side. In addition, AAA protocols using RSA [4] asymmetric cryptography in the client side, introduce an extremely large overhead when considering small devices. Low end CPUs (8-bit or 16-bit CPUs), which are common in small devices, have limited computational capabilities that are not adapted for use with certain AAA protocols using RSA cryptography. As an example, the H8/3048 CPU [5](16-bit, 16MHz, 512KB) takes 600 seconds to decrypt a message using RSA asymmetric cryptography with 2024-bits key size [6].

In this paper, we explore the Kerberos [7] protocol which offers third-party based authentication and key sharing framework. We suggest the use of Kerberos as the basis of a AAA framework that responds to the requirements of the target domain of application presented in Section 1.2. We study the Kerberos protocol operations in detail and we compare it against standard AAA protocols. We analyze the cross-realm operations in Kerberos and we show that despite some interesting proposals and extensions to the standard specification , the performance and the general model of cross-realm operations can be improved.

We propose a new model for cross-realm operations in Kerberos. Our approach has the advantage of making cross-realm operations transparent to the client and requiring minimal processing in the client device. This is performed by delegating the cross-realm operations (which are normally client-centric) to the Kerberos servers. Furthermore, our protocol allows the use of Kerberos in access networks for authentication of clients in roaming situations.

[1] The footprint of a software is the portion of computing resources, typically RAM, CPU time and disc space, that it requires in order to operate [3].

Chapter 2

Assumptions and AAA Requirements

In this chapter, we describe the realm reference model that we assume in our study and we introduce the AAA requirements for different service scenarios.

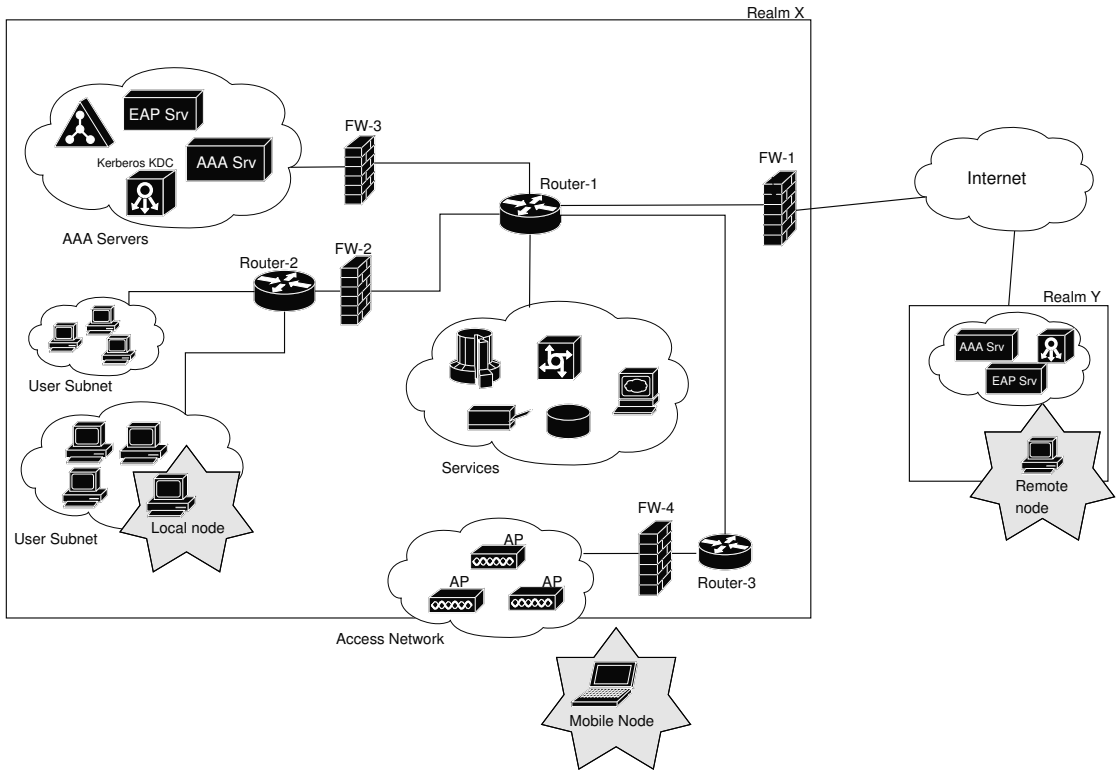


Figure 2.1: A Realm model

2.1 Realm model

Referring to Figure 2.1, we can distinguish three areas or subnetworks: Services subnet, AAA servers subnet, Users subnet and an access network.

Services subnet The Services subnet is deployed as a "*demilitarized zone*" (DMZ). A DMZ is a subnet that is located between the Internet and the Intranet trusted network of the realm. It contains services accessible from the Internet such as web server, ftp server and email servers. The services in the DMZ are also accessible from inside the realm. The DMZ is protected from Internet threats by deploying a firewall ("FW-1" in Figure 2.1) that blocks malicious traffic coming from the Internet. Since the DMZ is an area with high risk, a security perimeter is setup by deploying firewalls ("FW-2" and "FW-3") between the DMZ and the other internal subnets of the realm. The role of these firewalls is to protect the internal network. In case one of the services in the DMZ is corrupted by an attacker, "FW-2" and "FW-3" will form an obstacle that might stop the intrusion. Furthermore, the firewall "FW-3" protects the services from malicious users acting from inside the realm.

AAA servers subnet We use this term to refer to the subnet that contains any security related services. This includes AAA servers such as RADIUS [8] and Diameter [9] servers. Authentication servers such as EAP [10] and Kerberos [7] key distribution centers, naming services. etc... The safety of these services is very critical and requires highly secured deployment. If one of these services is corrupted, the whole realm is corrupted. These services are deployed after the DMZ, in a separate subnet. This allows the control of the traffic and permits better monitoring of the network activities related to the security protocols in the realm. The AAA subnet is accessible from hosts located inside the realm and it must be reachable from the Internet as well. The firewall "FW-3" is deployed to control the access to the AAA servers subnet. The administrators define the specifications of the traffic that is allowed to enter the AAA servers subnet. Hence, the implementation of the security policies can be implemented using this firewall. For example the firewall can be used to allow users from inside the realm to access the EAP server, while refusing this service for users located in remote realms.

Users Subnet The realm includes subnets where client hosts are deployed. Such clients could be the PC of the administration officers, end hosts in laboratories (in case of university or research center). Hosts in the users subnet are allowed to access several services deployed in the Services subnet. In order to access a certain service, the user have to obtain valid credentials, then perform authentication and authorization with the application service. All hosts in the users subnet are assumed to have network access and are able to communicate with the services from the AAA subnet as well as a certain number of application servers.

Access network The realm can offer wired or wireless access to the services subnet as well as traffic forwarding to the Internet. Mobile nodes located in the fringe of the realm can use the access points, deployed in the wireless access network, to gain network access. The users of the mobile nodes need to have necessary credentials and privileges in order to use the wireless access network infrastructure. After gaining network access, the mobile nodes can access the services subnet and the Internet by having the traffic forwarded by the main gateway. The firewall "FW-4" is used to isolate the wireless network access from the rest of the realm. This security

perimeter is used to control the traffic generated by the mobile nodes. Actually, several firewalls might need to be deployed in the access networks. We will refer to these firewalls as *Enforcement Points* (EP). The EPs are the mean by which the access network controls which mobile node can access the realm and which can not, according to the authentication results. Depending on the AAA framework model, the access points "AP" may have to deploy authentication agents or authentication proxies to relay the authentication protocol messages between the mobile node and the AAA servers. The EP can be deployed in the AP its self or in an independent hardware unit. The EP is controlled by the access control system to implement filtering rules.

2.1.1 Terminology

In the following section, we define some of the key terms encountered in this document, the definitions below will be used in many parts of the document when describing the different AAA frameworks and cross-realm operations.

Home realm

The home realm is the realm to which the user belongs. the user is the client that wishes to use resources deployed in his home realm or in another realm.

Remote realm

The remote realm term is used when a service is deployed in a realm that is different from the Home realm of the user. From the point of view of all the entities in the user's home realm (AAA servers, users, administrators) the realm offering the service is referred to as a remote realm.

Roaming

Roaming is the fact to have a user physically located in a realm different from his home realm.

Visited realm

When a user is in a roaming situation, the realm in which the user is physically located is referred to as visited realm

Remote service access

The remote service access is a scenario where a user, physically located in his home realm is wishing to access a service deployed by a remote realm. The term "Physically" denotes the network topology emplacement of the client. The client could be using a VPN to connect to his home realm and from there access remote services, in this case, we still consider that the scenario is remote service access scenario.

AAA framework

The set of protocols and software components (APIs, Server programs, Client programs) that allows a realm to control the access to the services that it provides. The AAA framework is an abbreviation of Authentication, Authorization and Accounting framework. In this thesis, we mention AAA framework to designate a system that supports at least authentication functions. The authorization and accounting functions can be independent from each others, and can be added to the authentication functions as subsequent features. However, if no authentication function is provided, there is no sense of having authorization and accounting functionalities.

Cross-realm AAA operations

Cross-realm AAA operations allow users to obtain credentials and to be authenticated in realms where they normally have no credentials.

2.2 Cross-realm AAA operations: scenarios and requirements

Cross-realm AAA operations are involved when the client and the service belong to different realms. When we analyze the different possibilities according to the location of the user, we can distinguish two cross-realm service access scenarios : remote service access and roaming.

2.2.1 Remote service access scenario

This situation occurs when the client is located in his home realm, whereas the service is deployed in a remote foreign realm. In this scenario, we assume that the client has already obtained Internet connectivity through the infrastructure of the home realm.

In Figure 2.1 "Remote Node", located in "realm Y", can connect to the services provided by "Realm X". This situation is referred to as remote service access scenario.

Figure 2.2 depicts the interaction between the user and the AAA system on one hand (1), and the interaction between the user and the application services on the other hand (2).

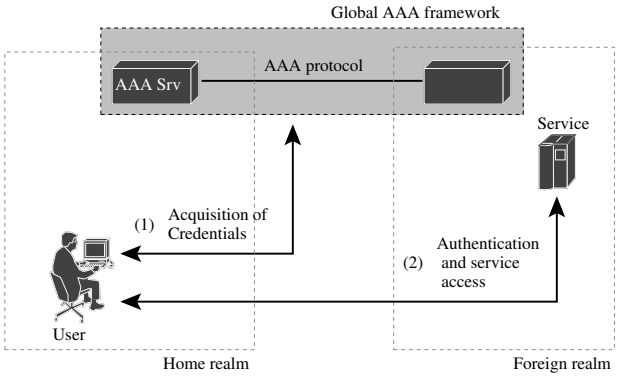


Figure 2.2: Remote Service access.

The "Global AAA framework" refers to the set formed by the AAA servers and any component that can be involved during the AAA operations, such components include authentication agents and proxies and enforcement points [1]. The Global AAA framework includes entities from the home realm as well as the foreign realm.

Depending on the architecture and the protocols used in the AAA framework, the user will have to communicate with one or many entities. The goal of this interaction is to deliver the credentials to the user, allowing him to be authenticated and authorized by the remote service. During this phase, the cross-realm protocols can be used between the different entities from both realms.

From the scenario described here, we can state the first requirement : The AAA framework, deployed between the home realm and the remote realm must be capable of delivering credentials to the user while he/she is located in his home realm.

2.2.2 Roaming scenarios

This situation can occur when the client is physically located in a foreign realm. We will refer to this realm as *visited realm*. In roaming scenarios, the client might want to access services

[1] Where the policy and the access control is actually implemented, such as firewalls and similar tools.

offered in the visited realm. For that, the user needs to obtain the necessary credentials.

Figure 2.3 depicts the interaction between the user and the AAA system on one hand (1), and the interaction between the user and the application services on the other hand (2).

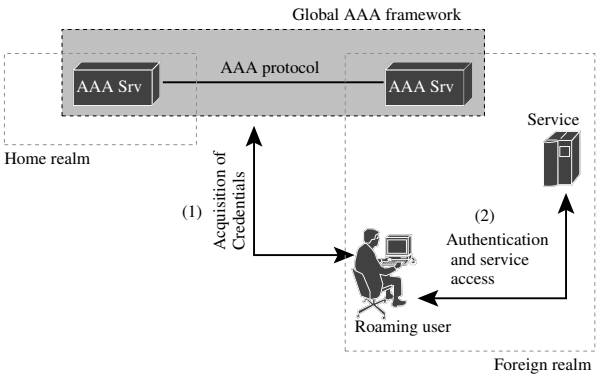


Figure 2.3: Roaming scenarios

It is assumed that, initially, the client does not have full network access to the realm. And no Internet access at all. The Internet connectivity is assumed to be a service that is offered to the authenticated users only.

The scenario described here, requires that the Global AAA framework to be capable of delivering the credentials to the user while he/she is a visited realm, with the constraint of limited network access.

Chapter 3

AAA framework model : Components and operations

In this chapter, we describe the most common model for AAA frameworks, its different components and the different operations between the different entities.

3.1 Generic AAA framework model

In order to deploy AAA frameworks, different authentication, authorization and accounting protocols and different implementations of these protocols can be used. However, all AAA frameworks should follow the same model and architecture specified and recommended by the standardization organizations such as the IETF. The "IETF Network working group" has specified its recommendation for the architecture of AAA frameworks by publishing two RFCs [11] [12]. In each part of the AAA framework, several inter-changeable protocols can be used. It is up to the realm administrators to choose which are the specific protocols they will use according to the specific needs. The AAA model only defines what category of protocols should be used and how the AAA frameworks should be organized. The specification of a standard architecture for AAA frameworks was intended to offer inter-operability and scalability, especially when inter-realm operations are involved. Furthermore, the standard AAA model has as an objective to avoid scalability problems and to allow easy evolution of the AAA frameworks as new protocols are defined.

The main actors in AAA scenarios are the user, the AAA servers and AAA agents. In the following paragraphs we describe the role of each of these entities and the relationship between them.

3.1.1 AAA Clients and agents

In order to access a service, users must first be authenticated and authorized. In a second phase, the accounting framework is used to keep track of the resource utilization. In each phase, the client and/or the application server will need to communicate with the other entities of the AAA framework. For this purpose, the client or the application service implement what is referred to

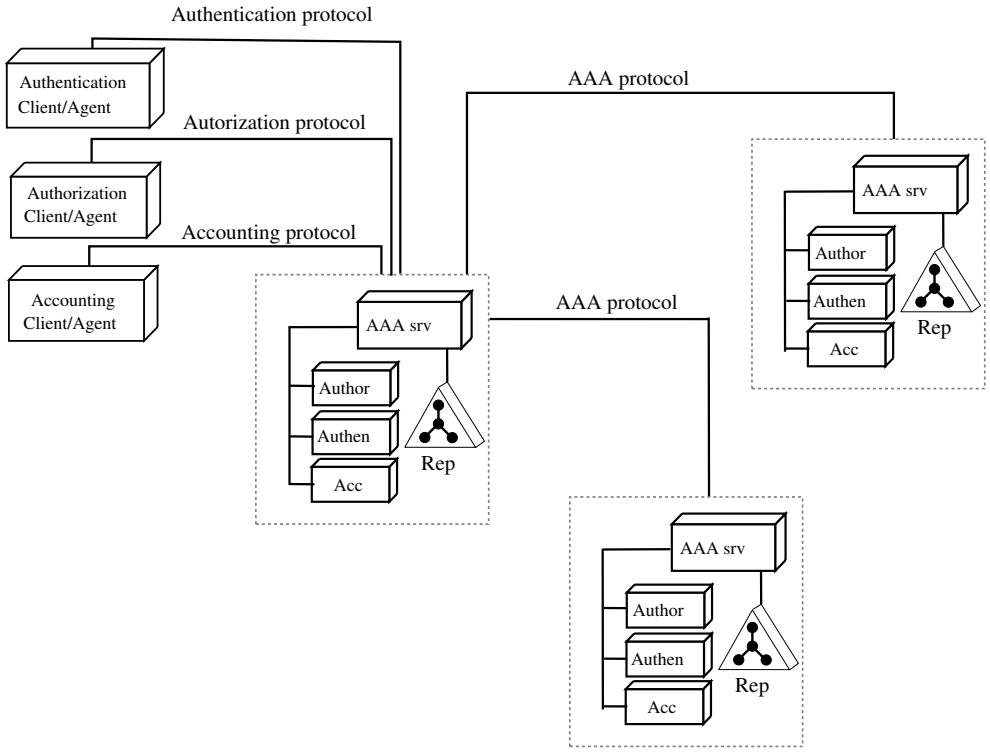


Figure 3.1: Generic AAA framework with multi-domain support

as a **AAA client**. Trough the use of these protocol components, the AAA operations can be initialized by the service or the user. In the next paragraphs, we describe the different types or AAA clients and agents.

Authentication clients and agents The authentication client or agent is a protocol component used by a service or a user to perform operations aiming to prove the identity of the user. The authentication client uses a specific **Authentication Protocol** to communicate with the AAA server. The AAA server hosts the server side of the authentication protocol, and thus is capable of processing the requests. As an example, the Authentication client could consist of a module in a web-server, that authenticates users by verifying their credentials (login and password pairs) registered in the central AAA server of the organization hosting the service. In this scenario, the web-server is said to act as an **Authentication agent**. It gets the credentials from the clients, then acts as a AAA client to query a AAA server. The goal of this AAA exchange is to verify the credentials of the web clients. In other scenarios, the user behaves as an authentication client and performs authentication to prove his identity directly to the AAA server. In such scenarios, the users obtain tickets or certificates to use them as proof of their identity when requesting the service from the application servers. In this situation, the user is said to act as an **Authentication client**. Independently from the authentication model, the AAA server can always prove the identity of users by checking their credentials.

Authorization clients and agents The authorization clients and agents are the client side of the authorization management and policy enforcement system. In order to apply the realm's

policies, the access to any service must be regulated. For this purpose, authorization frameworks are designed and used in different fashions. As an example, a user accessing a multimedia service has a profile that specifies the quality of service that can be granted to the subject. Such authorization information, could be retrieved by the application server from the central AAA server. In such a case, the application server acts as an authorization agent. In other models, the client can obtain authorization data and present it to the application service that will use them to define the user rights and privileges. In this latter case, the user acts as an authorization client since it requests authorization data by its self.

Accounting clients and agents The accounting operation consists on keeping track of the user activity, and his use of the resources. In the commercial and business world, this is a very important part, because the information gathered trough accounting is usually the mean by which bills are calculated. As for authentication and authorization, the accounting can be performed by the client or the service, depending on the needs. If the service side is the initiator or the accounting process, then, it is said to act as accounting agent. In the other side, if the user's device or client is responsible for generating the accounting data then the user is said to act as an accounting client.

3.1.2 AAA Server

The AAA servers are the heart of AAA frameworks. Normally, each realm have at least one AAA server. As shown in Figure 3.1, the AAA server is composed of several application specific modules which implement an authentication, authorization or an accounting protocol. The AAA server uses a generic AAA protocol (see next section) to communicate with other AAA servers and AAA clients.

The AAA server receives AAA messages from clients or other AAA servers and process them as follows: The AAA API, determines if the AAA message can be processed locally. For this purpose, the realm information included in the AAA message is extracted. If the AAA message is destinated to the local realm. , then, the message have chances to be processed locally. In order to actually process the message, the AAA API fetches the identifier of the target application module from the message. This information indicates which module of the AAA server can process the AAA message. If the application identifier is not known to the AAA server then the message is rejected and an error is sent to the originating peer. If however, the application module having the same identifier, extracted from the AAA message, is present. The AAA message is delivered to this module. From there, the processing of the message will take place and resulting reply messages generated from the application module are forwarded to the initiating peer.

3.1.3 The AAA protocol

The AAA protocol is defined as the underlying framework that connects AAA clients, agents and servers to each others. The AAA protocol is used to transport different authentication, authorization and accounting protocols between the different entities.

The AAA protocol, offers transport and hop by hop security. It acts thus, as an underlying framework providing secure transport and routing service for upper layer protocols.

The transport service is offered trough an encapsulation of the upper layer protocols within the AAA messages. The encapsulation is implemented trough attribute/value pairs that contains

the protocol specific data. The AAA messages are hence, composed of a set of attribute/value pairs (AVP)s that contains application specific data as well as AVPs transporting routing and security related informations that are only used by the AAA protocol. When an agent or client wishes to communicate with a AAA server, it will decompose the protocol specific message into AVPs, these AVPs are then embedded into AAA messages routable by any AAA server. However, the contents of these AVPs must be processed by the right application specific module on the specific AAA server according to the destination realm information embedded in the AAA message.

3.2 Example of AAA infrastructure used for web authentication

As an example, in order to authenticate some web client, the web service in Figure 3.2 obtains a challenge response (For example, a nonce encrypted using the user's password) from the client and the client's user name. Then it passes these informations to the authentication client module which will take in charge the verification of these credentials. The authentication client module will speak to the authentication server or module in the AAA server of the realm. For this purpose, the authentication client module will use the generic AAA API to encapsulate the user ID and the challenge response. The identifier of the application (the authentication protocol in this case) is included in the AAA message. The AAA API, then takes in charge the transportation of the AAA message to the destination AAA server. Once received by the AAA API of the AAA server, the AAA message will be delivered to the right application specific module, which is in our case, the Authenticator module or service. The AAA API in the AAA server uses the identifier of the application embedded in the AAA message to decide to which upper layer component the AAA message should be delivered. As shown in Figure 3.2 the authentication client module and the authenticator module communicates in an end-to-end manner using the AAA infrastructure as a transport layer.

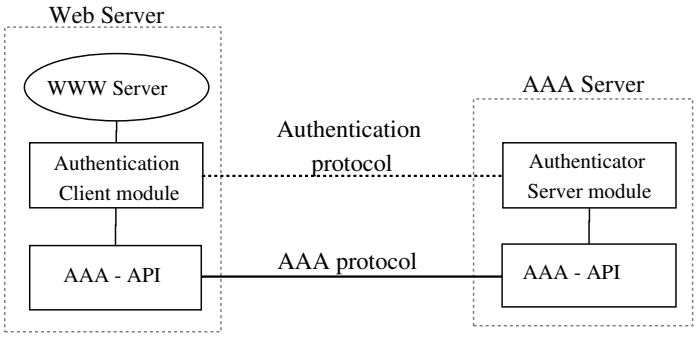


Figure 3.2: Example of AAA infrastructure used for web authentication

In the same manner, an authorization or accounting client can communicate with an authorization or accounting service in the AAA server using the AAA framework.

Cross-realm operations using the AAA framework The cross-realm operations are transparent to the client and agent entities, the AAA API takes in charge the routing of the AAA message to the AAA server that is capable of processing the AAA request. In the same example of web server. Let's suppose that some clients does not belong to the realm of the web

server. In this case, the authenticator module of the local AAA server can not authenticate the client. Thanks to a supplementary information about the client's realm, the AAA infrastructure, can determine the AAA server that knows more about the current client (Remote AAA server in Figure 3.3). This is called **realm-based routing**, in which the AAA infrastructure uses the realm name of the client as information to determine the AAA server that can verify the client's credentials. In the case of the web server example (see Figure 3.3), the local AAA server, can detect that the realm of the user is different from the local realm name. A realm routing table then is used to determine the AAA server that corresponds to the client's realm. The AAA message is then forwarded to the AAA server of the realm to which the user is registered. The remote AAA server will process the request by delivering it to the authenticator module, then send back the result to the local AAA server, which in turn forwards the message to the authentication client module in the web server.

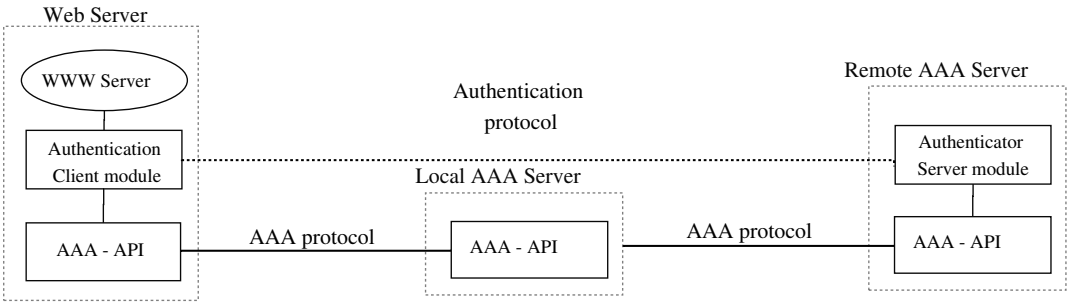


Figure 3.3: Example of cross-realm authentication

3.3 AAA operational modes for roaming

In this section, we describe the different functional modes for performing AAA operations with roaming users. As previously discussed in Section 2.2, the roaming situation occurs when the organization that authorizes and authenticates the user is different from the organization providing the service.

For this purpose several models were specified in [12]. These models describe the interaction between the roaming user, the home AAA infrastructure and the AAA infrastructure of the service provider. The result is to provide credentials and/or check the user's credentials in order to authenticate and authorize the roaming user.

3.3.1 The agent sequences

In the agent sequence (see Figure 3.4), the home AAA Server operates as an agent between the User and the AAA infrastructure of the service provider. The home AAA Server receives a request from the User (1) and forwards authorization and possibly configuration information to the service provider (2). The AAA server of the service provider as well as the service equipment setup whatever is required and become ready for the roaming user. The service provider responds to the home AAA Server acknowledging that it has set up the service for the user (3). Finally, the AAA Server replies to the user telling him that the Service is ready (4).

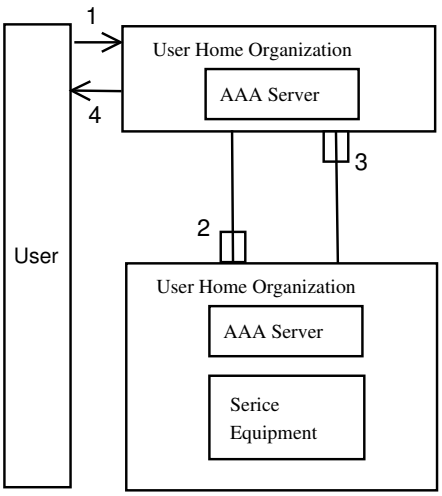


Figure 3.4: Roaming agent sequence

3.3.2 The pull sequence

The pull sequence (see Figure 3.5) is what is typically used in the Dial-in application, in the Mobile-IP proposal, and in some QoS proposals. The AAA server of the visited realm acts as a proxy agent between the user and the AAA server of the user's home organization. The client request (1) is forwarded to the home AAA server (2) which evaluates the request and returns an appropriate response to the AAA server of the service provider (3) which then can provide credentials and grant service access to the user (4). The generic AAA model presented in Section 3.1 follows the pull sequence model, since the AAA server acts as an agent if the realm of the client is different from the realm of the local AAA server.

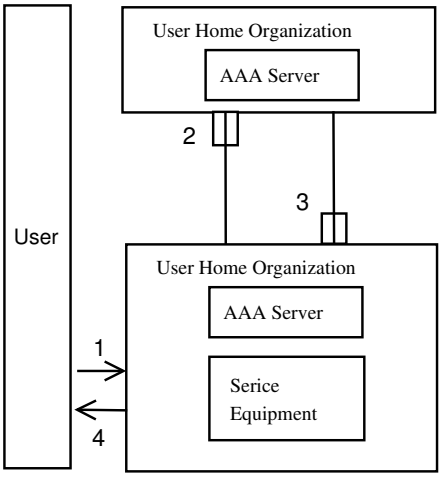


Figure 3.5: Roaming pull sequence

3.3.3 The push sequence

The push sequence (see Figure 3.6) requires that the user obtains a certificate or ticket from the AAA server of the home organization (1) and (2). These credentials are then presented to the AAA server of the service provider or directly to the application server (3). The AAA server or the application server will set up the service for the user then replies to the user telling that the service is up (4). Kerberos based frameworks fall in this category. The pull model is not adequate for use in network access authentication since it introduces a pre-authentication problem. The pre-authentication problem here, consists on the fact that the client can not contact his home AAA server trough the infrastructure of the service provider. Hence, the client find him self in a deadlock situation and can not authenticate to the AAA server of the service provider.

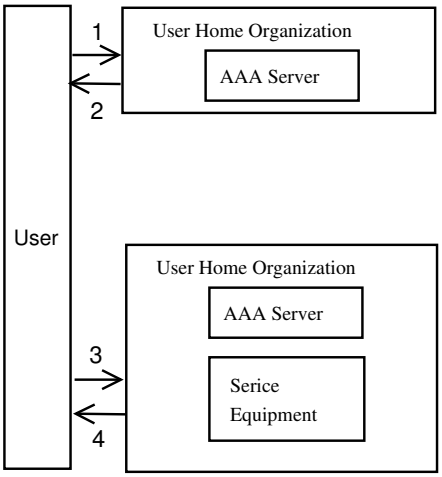


Figure 3.6: Roaming push sequence

3.4 Example of AAA framework using the pull sequence model

In this section, we describe an example of typical AAA framework using the pull sequence model as described in Section 3.3.2 for authenticating clients requesting the network access service. The framework uses Diameter [9] as a AAA protocol, and PANA [13] as front-end authentication agent and AAA client.

PANA PANA (Protocol for carrying Authentication Network Access) [13] is a new authentication protocol currently designed at the IETF in the PANA working group. It is link-layer agnostic and thus can be used over any access technologies (802.11, 802.16, xDSL, GPRS, 3G, etc). It permits clients to dynamically select ISPs. Any authentication method can be used as PANA carries the EAP (Extensible Authentication Protocol [10]) protocol, which is an authentication framework that supports many authentication mechanisms such as certificates and one-time passwords [14]. The PANA protocol also introduces the *Enforcement Point* (EP), an equipment on which security policies are applied. This Enforcement Point can be configured by the authentication agent using SNMPv3 [15]. The PANA protocol is used between the PANA

client (PaC) and the PANA Authentication Agent (PAA) which relies on a AAA server to authenticate clients using the EAP protocol. The AAA protocol used between the PAA and the remote AAA server must be able to carry EAP packets.

Diameter Diameter [9] is the next generation AAA protocol. It aims at replacing the well-known RADIUS protocol [8]. Diameter offers several advantages over RADIUS and is intended to provide an Authentication, Authorization & Accounting (AAA) framework for applications such as network access or IP mobility. Diameter is composed of a Base protocol extended by other mechanisms called *Diameter-applications*. Network access is an example of such a *Diameter-application*. Another example is the Diameter EAP application [16] which can be used as the AAA protocol by the PAA.

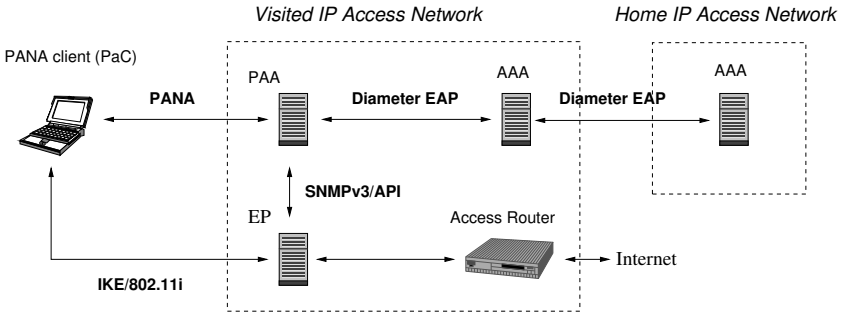


Figure 3.7: Diameter/PANA AAA framework

As shown in Figure 3.7, the PANA protocol is used between the mobile node and the network access servers to bootstrap the authentication process in a first stage. The pana authentication agent then, transports EAP messages between the client and the back end AAA server. The PANA protocol is used to encapsulate the EAP protocol between the PaC and the PAA. The PAA uses the Diameter AAA API to send EAP packets encapsulated in Diameter AAA messages. The local Diameter AAA server uses realm based routing to forward the AAA message to the AAA server of the user's ISP. The EAP session is thus established between the user and the EAP authenticator deployed in the home IPS's AAA server. Once the EAP session starts the PAA observes the exchanges until the end of the EAP session. If the result of the EAP session states the success of the user authentication, then the PAA will setup the NAS service to provide network access for the client.

Chapter 4

Kerberos based AAA framework

The objective of this chapter is to give an overview of the Kerberos protocol and the related work. After a brief description of the Intra-realm operations, we will focus on the cross-realm support in Kerberos. The cross-realm support defines how the protocol delivers credentials for clients in both scenarios described in Section 2.2. We discuss the advantages and shortcomings of the Kerberos protocol as a cross-realm AAA framework. The different proposals (or Kerberos extensions) to improve the cross-realm operations are then studied and discussed to demonstrate their advantages and shortcomings.

4.1 Kerberos intra-realm operations : The basic model

Kerberos [7] is a widely deployed authentication system based on the Needham and Schroeder protocol [17]. It offers three-party scheme for session key distribution between services and clients [18]. The Kerberos protocol involves Kerberos servers or **Key Distribution Centers (KDC)** and **principals**. The principals correspond to the users and services of the realm and they share long-term keys with the KDC.

While describing the Kerberos protocol, we will often refer to the following definition :

Session Key

The session key is a secret that is (or will be) shared between the client and the server for a pre-defined period of time (session time). The session key is generated by the KDC and shared between the user and the service using a method based on the Needham and Schroeder protocol.

Ticket

A message that contains informations about a certain user and a service, the ticket is encrypted using the secret key shared between the KDC and the service. The ticket certifies that its holder is allowed to access the service. Only the service in question can decrypt the ticket and explore its contents. The ticket is used by the KDC to communicate a session key to the service. When accessing services, the client sends the ticket to the service. By doing this, the service and the client theorically share a secret session key. Based on this fact, the user and the service can perform mutual authentication and secure further communication.

Authenticator

The authenticator is a message that contains information about the user and the target service. It also contains timestamps as well as the IP address of the client machine. The authenticator is encrypted using the session key that is included in the service ticket for the target service. It is sent to an application service along with a service ticket. It allows the user to prove to the service that he/she knows the session key included in the accompanying service ticket.

Credentials

Kerberos credentials consists on a ticket and a session key. Each ticket and session key pair allows the user to access some service. In order to obtain these credentials, the user must communicate directly with the KDC using the Kerberos protocol

As specified by the standard [7], the basic Kerberos authentication process proceeds as follows: A client sends a request to the authentication server (AS) for *credentials* for a given server. The AS responds with these credentials which consists of a *ticket* for the server and a temporary encryption key (called a *session key*). The session key is encrypted using the secret key shared between the user and the AS. The ticket in the other hand, contains the same session key and is encrypted using the long term key shared between the service and the AS. The client transmits the ticket to the server. The session key is obtained by the application service after decrypting the ticket using the long term key shared with the AS. Based on the knowledge that each party have obtained the same session key, the client and the application service can proceed to mutual authentication.

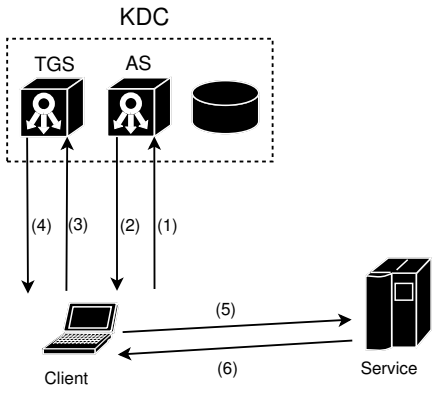


Figure 4.1: The Kerberos authentication protocol

Figure 4.1 shows the details of the Kerberos protocol components and operations. The KDC have two components; an **Authentication Server (AS)** and a special *principal* (service) called **Ticket Granting Service (TGS)**. The AS delivers the credentials required for sharing a session key and performing authentication with the TGS, while the TGS delivers credentials required for sharing a session key and authentication with the application servers.

- **(1)** : The user requests *global credentials* from the AS. These global credentials would allow the user to share a key and perform mutual authentication with the TGS service. In the **AS-REQ** message, the user specifies his name and the target service name (which corresponds to the service name of the TGS principal).

- **(2)** : The AS extracts the user's key and the key of the TGS service from the principals database. In the **AS-REP** message, the AS delivers the global credentials to the client. These credentials include a session key encrypted using the long-term key shared with the KDC. We will refer to this session key as **TGS session key**. The user also receives a **Ticket Granting Ticket (TGT)** which is a ticket that contains the same *TGS session key* and encrypted using the TGS's key shared between the TGS and the KDC.

- **(3)** : The client sends the TGT to the TGS, along with an **authenticator** in a **TGS-REQ** request asking for credentials required for authentication with a certain application service. The *authenticator* is a message encrypted using the *TGS session key* and acts as a proof that the client has successfully obtained the *TGS session key* after decrypting it using his long-term key shared with the KDC.

- **(4)** : After decrypting the TGT (which was encrypted by the AS using the long term key shared between the TGS and the KDC), the TGS extracts the *TGS session key*. Using that session key, it authenticates the user by decrypting and validating the *authenticator*. Finally, the TGS creates a **TGS-REP** message to convey the requested credentials to the client. The TGS-REP message includes a newly generated session key encrypted using the *TGS session key*. We will refer to this session key as **Service Session Key**. The user also receives a **service ticket (ST)** from the TGS, which is a message that contains the same *service session key* and encrypted using the key shared between the application server and the KDC.

- **(5)** : The client sends the *service ticket* to the application server, along with an *authenticator* in a **AP-REQ** request message asking for access to the provided service. The *authenticator* here, is encrypted using the *service session key* and acts as a proof that the client has successfully obtained the *service session key* after decrypting it using the *TGS session key* shared with the TGS.

- (6) : On reception of a client request, the application server decrypts the *service ticket* using its own secret key and obtains the *service session key* shared with the client. Using this session key, the application server authenticates the client by decrypting and validating the *authenticator*.

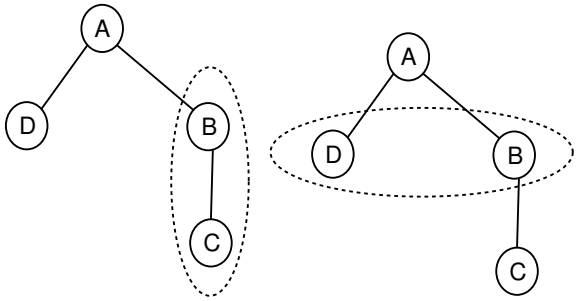
The tickets (TGT and ST) have a lifetime after which they expire and must be renewed. Time-stamps included in the tickets indicate to the servers if the ticket has expired or not.

4.2 Kerberos cross-realm authentication support

The Kerberos protocol was enhanced since version V to support authentication of users across organizational boundaries. The cross-realm authentication support in Kerberos allows clients to acquire credentials for accessing services offered in a realm different than the realm to which they belong. In the following section, we study the cross-realm authentication in Kerberos.

4.2.1 Inter-realm trust

If a set of two or more realms decide to allow cross-realm authentication of users, a trust relationship must be built between the realms.



(a) Direct trust (b) Transitive/Indirect trust

Figure 4.2: Inter-realm trust model

Direct trust The establishment of a trust relationship consists on making KDCs from different realms trust each others by means of symmetric key cryptography. This implies that each KDC needs to share a secret key (called **cross-realm key**) with the other KDC. In practice, this is done by adding the TGS service of each KDC as a principal in other KDCs.

In Figure 4.2(a) the realms B and C have direct trust relationship, A and B also have a direct trust relationship, the realms A and D as well. Links between two vertices of the graph symbolizes direct trust relationship established trough shared secret keys between the corresponding realms.

Transitive trust With the specification of cross-realm authentication feature of Kerberos, the notion of **authentication path** was introduced. In order to perform cross-realm authentication between two realms, there is no need to have direct trust relationship. The **transitive trust** relationship allows clients to access services offered by certain realms that does not have direct trust relationship with the user's home realm. This relationship is based on the direct trust and works as follows: If realm D and A trust each others and realm A and realm B trust each other, then realm D and realm B trust each others. In Figure 4.2(b) realms D and B have indirect trust relation ship, the realms A and C as well. Each couple of realms that do not have direct trust relationship can rely on transitive trust relationship to build and authentication path for cross-realm authentication. In our example, the authentication path between the realms D and C is the following : D/A/B/C. In order to deliver credentials for the clients in realm D, the KDC of realm D must know the next step in the trust path i.e. the KDC of realm A.

4.2.2 Cross-realm authentication process

When a user wants to access some service offered in a foreign realm, he/she needs to exchange messages with the Kerberos Key Distribution centers of the home realm and the service provider in order to be able to get the required Kerberos credentials. In the following paragraphs we detail the operations performed by the client and the Kerberos KDCs during this process. We do not

make any assumption about the location of the user and the application service. The only fact is that the user and the service are administrated by different realms that use different KDCs.

Determining the full Kerberos principal name of the service

First of all, the client needs to know the principal name of the server and the realm where the application server is registered.

Current implementations of the Kerberos AS and TGS protocols, as defined in [7], use principal names constructed from a known service name and realm. A service name is typically constructed from the service's name and the DNS host name of the computer that is providing the service.

The user have to be able to build the full Kerberos principal name which have the following format :

PRINCIPAL-NAME/HOST-FQDN@REALM-NAME

where,

- *PRINCIPAL-NAME* : Kerberos principal name of the application server. For the PRINCIPAL-NAME part, the client can rely on a naming convention adopted in the remote realm. Naming conventions state for example, to give the principal name "HTTP" for web servers, "FTP" for ftp servers etc.
- *HOST-FQDN* : The fully qualified domain name of the host where the application server is running. The client can rely on nameserver to perform reverse-lookup on an IP address. Still the client need to know the IP address of the service beforehand.
- *REALM-NAME* : The Kerberos realm name of the domain or organization that is deploying application server. The REALM-NAME part can be be determined in several ways. It might be known beforehand or, it might be stored in a nameserver, it might be also obtained from a configuration file. MIT Kerberos implementation uses a matching scheme that allows clients to determine the realm name of the services given the FQDN of these service. Hence, If the client knows the HOST-FQDN of the service which can be *ftp.example.org*, the client deduces that the service belongs to the domain *example.org*. The mapping between the DNS name of the realm and the Kerberos realm name is then performed using the configuration file of the Kerberos client.

As an example : HTTP/www.example.org@EXAMPLE.ORG is the full Kerberos principal name of a web server deployed by the realm EXAMPLE.ORG.

To summarize, the configuration file in the client side allows to determine, given the HOST-FQDN of a service, the Kerberos realm name and the location of the KDC corresponding to the domain where the service is located. This mechanism requires that each client have very detailed configuration information about the hosts that are providing services and their corresponding realms. Having client side configuration information can be very costly from an administration point of view - especially if there are many realms and computers in the environment. The Internet-draft [19] documents a method for a Kerberos Key Distribution Center to respond to client requests for Kerberos tickets when the client does not have detailed configuration information on the realms of users or services. This method referred to as **KDC referrals** simplifies administration by minimizing the configuration information needed on each

computer using Kerberos. The draft is a proposed extension under consideration for integration in RFC4120. The Windows 2000 implementation of Kerberos uses the KDC referrals.

Obtain a TGT by issuing recursive TGS requests

When a client, physically located and registered in a realm A, wishes to access a service deployed in a realm B. the client must obtain a TGT that allows him/her to communicate with the TGS of the remote realm (realm B). This special TGT, called **cross-realm TGT**, is delivered by the home KDC of the client or by an intermediary KDC, depending on the nature of the trust relationship established between the home realm and the target realm.

The client checks in its ticket cache to determine if he got a usable cross-realm TGT. If the needed cross-realm TGT is not present in the cache, the client issues a TGS-REQ message to its home KDC. The Kerberos KDC as well as the client must be aware of the cross-realm operations. The cross-realm operations, in deed require special behavior from the Kerberos client and the KDC.

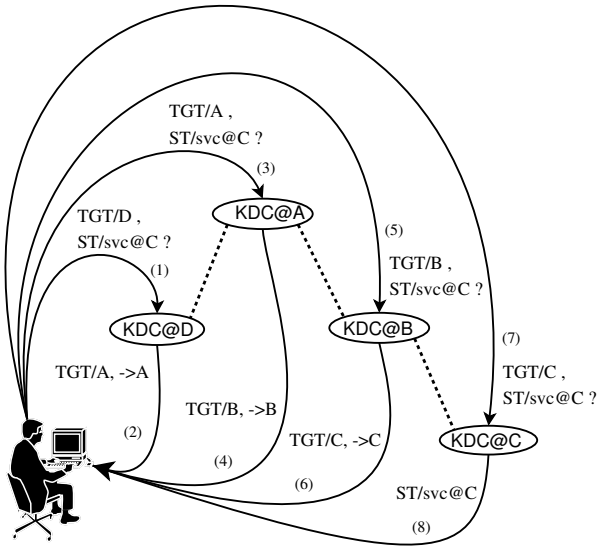
On reception of a TGS request message, the home KDC may return a TGT for the desired realm (when the home realm A and the remote realm B have a direct trust relationship), in which case one can proceed. Alternatively, the Kerberos server may return a TGT for a realm that is 'closer' to the desired realm (further along the trust path between the client's realm and the requested server's realm). In this case, the client issues a new TGS request to the Kerberos server in the realm specified in the returned TGT. If the home KDC does not return no TGT at all, then the request must be retried with a Kerberos server for an intermediary realm in the trust path (if the client have a TGT for the corresponding KDC).

When the client receives a TGT for a closer realm, it may use local policy configuration to verify that the authentication path used is an acceptable one. Alternatively, a client may choose its own authentication path, rather than rely on the Kerberos server to select one. In either case, any policy or configuration information used to choose or validate authentication paths, whether by the Kerberos server or by the client, must be obtained from a trusted source.

The protocol described so far is referred to as KDC referral technique. *KDC referrals* is the technique used in cross-realm authentication for allowing clients to recursively obtaining TGTs from intermediary realms until obtaining a TGT for the target realm. The Figure 4.3 shows the different TGS exchanges between the client and different KDCs of the authentication path.

The KDC referral technique proceeds as follows: The client sends a TGS request to his home realm "Arrow (1) in Figure 4.3" for a service located in a remote realm "C". The service principal name of the service corresponds to the service that the user wishes to access. The principal name used in this example is "svc".

If the KDC of the user's home realm (KDC@D) and the KDC in the foreign realm (KDC@C) have an established direct trust relationship by means of pre-shared secret keys, then the home KDC can deliver the TGT to the user in the TGS-REP message (2). However if the home realm does not share cross-realm keys with the foreign realm, the home KDC will provide a cross-realm TGT (TGT/A) to use with the KDC of the intermediary realm, KDC@A, that is known to have direct or indirect trust relationship with the target KDC. The client uses this intermediary cross-realm TGT and the information about the next KDC to locate the intermediary KDC send to it a TGS-REQ message (3). This request is similar to the initial request sent to the home KDC except that this time, the cross-realm TGT obtained from the home KDC (TGT/A) is used to communicate with the TGS of the intermediary realm.



KDC@X	KDC of realm X.
TGT/X	TGT usable in realm X.
ST/svc@C?	The message is a request for a service ticket, to be used with the service "svc" registered in the foreign realm C.
KDC/X, ST/svc@C?	The client provides credentials for communicating with the KDC@X (TGT/X) and asks for a service ticket (ST/svc@C).
->X	Referral for the realm X.
TGT/X, ->X	The KDC can not process the client request and delivers a TGT for the next realm in the authentication path. The KDC reply includes a referral that indicates the next KDC to contact.

Figure 4.3: Kerberos cross-realm operations using KDC referrals

The intermediary KDC will have the same behavior as the home KDC ; If the Intermediary realm and the target realm does not share cross-realm keys, the KDC will point the user to another intermediary KDC "KDC@B in the Figure 4.3" (just as in the first exchange between the user and his home KDC), however in the opposite case, when the intermediary KDC has direct trust relationship with the desired foreign realm (as it is the case with KDC@B), the intermediary KDC will issue a cross-realm TGT that can be used in the target realm (8).

In the case where only indirect trust relationship is linking the user's home realm and the target realm. The KDCs must determine an authentication path. The authentication path tells to the KDC which is the next intermediary KDC to be contacted.

According to the current standard, Kerberos clients are aware of the cross-realm operations. The client can refuse to go trough a certain intermediary realm or completely specify the authentication path its self, according to a certain policy. In such case, the client will issue requests and obtain cross-realm TGSs from its home KDC until the target realm's KDC.

When realms are be organized hierarchically. Each realm shares a key with its parent and a different key with each child. The hierarchical organization allows an authentication path to be easily constructed. If a hierarchical organization is not used, it may be necessary to consult a database in order to construct the authentication path.

Cross-realm support implementations

The reference implementation of the Kerberos protocol written in MIT [20] is based on [21] and [7]. In MIT Kerberos implementation, the cross-realm authentication requires the client to know the structure of the domain hierarchy. The client uses his knowledge of the realm hierarchy to retry TGS requests with realms higher in the hierarchy than his own realm, in case the home realm did not provide a cross-realm TGT.

This process makes the Kerberos clients very complicated and hard to maintain. For example, if an organization's realm structure changes, the configuration file on all MIT Kerberos clients must be manually updated. Therefore, the current MIT Kerberos client is not scalable in an environment with multiple realms where the realm hierarchy is dynamic [22].

Both [21] and [7] recommend that the client knows the trust relation ship between the different realms. The main reason behind this is to allow the client to verify that the authentication path used is an acceptable one. The second reason is to allow the client to choose its own authentication path, rather than rely on the Kerberos server to select one. The MIT implementation allows this feature by using configuration files on the client side that indicates the trust relationships between the realms.

The Windows 2000 Kerberos implementation, uses the *server referral* [19] for resolving the full Kerberos principal name of the services and request credentials for these services in the same time. This leverages the client side from building the full Kerberos principal name using the client's resources. The Windows 2000 implementation of cross-realm authentication assumes that the client is not aware of the realm hierarchy neither of the trust relationships. The client relies heavily on the KDC for principal name resolution and cross-realm referrals.

4.3 Evaluation of Kerberos Cross-realm operations

In this section, we discuss the issues and the possible areas of improvement related to the Kerberos protocol. The contribution of this thesis is based on the specification of these issues. Indeed, after doing the state of the art in current AAA standards. We focused on the Kerberos protocol, we found that the Kerberos protocol have several advantages over the other AAA framework candidates. On the other hand, we found out that several aspects of the Kerberos protocol can be improved. In the following paragraphs, we start be discussing the advantages of a Kerberos based AAA framework compared to other solutions. Then, we discuss the issues related to Kerberos and explain their relevance and their impact when considering the domain of application in which we are interested, i.e. mobile devices with low computational capabilities.

4.3.1 Comparison of Kerberos against other authentication frameworks

The Kerberos protocol is well accepted in the industry, enterprises and academic world as a legacy authentication protocol. The reputation of the Kerberos protocol is mainly due to the light processing requirements of DES [23]operations. Additionally, the Kerberos protocol presents a novelty which consists on the third-party based secret key distribution scheme between services

Criteria/AAA framework	**Kerberos**	**PKI**	**LDAP with MD5 challenge**
Processing Requirements	Simple DES operations	Expensive RSA computations	Simple Hash computations
Mutual authentication	Yes. Client authenticates service and can authenticate KDC	Yes. Client and service are authenticated	Only client is authenticated
Credentials with lifetime	Yes. Kerberos tickets have a pre-defined lifetime	No. But, CRL mechanism can be used to invalidate credentials	No. Manual intervention on the LDAP database is required
Provides Key sharing between client and service	Yes. The generation of shared session keys is a basic part of the protocol	No. Generation of a session key have to be implemented separately	No. Generation of a session key have to be implemented separately
Single sign-on	Yes. The Kerberos TGT is acquired once by the client and used to obtain credentials of application services without typing the secret key	No. The PKI offer authentication for each service independently	No. The authentication for each service is performed independently
Support of delegation of authentication	Yes. Application services can use the user's credentials for authentication with other application services	No.	No.

Positive **Negative**

Table 4.2: Comparison of Kerberos against other popular systems

and clients. Furthermore, the Kerberos scheme offers single sign-on feature that requires the user to type his/her password only once.

The Table 4.3.1 summarizes the advantages of the Kerberos protocol compared to a PKI based authentication system and an LDAP [24] based authentication system with a DIGEST MD5 [25] challenge mechanism.

The characteristic of the Kerberos authentication system consists in its excellent $Performance/Robustess$ ratio. Furthermore, The third-party based scheme allows the client and service to perform mutual authentication and share session keys in the same time. This makes Kerberos usable as authentication and key distribution system. KINK [26] specifies the use of Kerberos as key distribution framework for IPSec [27] security associations. For these reasons, we estimate that the Kerberos protocol is the best choice regarding the domain of application that we fixed. The low overhead and multiple functionalities and features makes Kerberos an excellent candidate for a AAA framework destinated controlling a set of services

for mobile and small devices.

4.3.2 Issues of the Kerberos protocol

In the following paragraphs, we discuss some issues of the Kerberos protocol that can be improved in the objective of enhancing the cross-realm operations and considering the domain of application presented in Section 1.2.

Exposure of the KDC

One of the assumptions made by Kerberos is that users can communicate with KDCs located in remote realms. This assumption suggests that KDCs in each realm are open for client requests coming from anywhere in the Internet. This can introduce security threats related to the fact that KDCs are open to the public. One can think about limiting the access to the clients from the trusted realms. But even such filtering rules are not enough since attacks targeting the KDCs can come even from realms considered "trusted". Hence, limitation of the connections to the KDCs to a certain number of realms is not a solution to protect the KDC. And this issue can not be resolved unless very strict firewall rules are established. For this, the specific list of external hosts allowed to communicate with the local KDC must be defined. But this task requires a non realistic effort from the administrators. For this reason, the actual protection of the KDCs is using a more relaxed but unsafe policy.

Cross-realm operations are client-centric

In the cross-realm operations, Kerberos clients have to issue TGS requests and process TGS replies with all the KDCs of the trust path. In some cases where the client have limited computational capabilities, the overhead of these cross-realm exchanges may grow into unacceptable delays.

As an example, if we consider a device based on a 8051-compatible 8-bit CPU[1], the processing time requested for encrypting/decrypting a 1000 Bytes of data using the 3DES algorithm, takes about 2.5 seconds [6]. If we suppose that there is a single KDC in the trust path between the user's home realm and the service's realm, then the client device needs to issue and process three TGS exchanges. Each TGS exchange will demand a 3DES encryption for creating the authenticator and a decryption operation to process the TGS-REP message. In these circumstances, such a device would take more than 2.5 x 2 x 3 = **15** seconds to obtaining a service ticket for a remote service.

Pre-authentication problem in roaming scenarios

In roaming scenarios, the client need to contact its home KDC to obtain cross-realm TGT for the local (or visited) realm. However, the policy of the network access providers does not allow clients to communicate with hosts in the Internet unless they provide valid authentication credentials. In this manner, the client falls in a dead-lock situation where the Internet connection is needed for obtaining credentials and the credentials are needed to obtain the Internet connection. As a result, the Kerberos protocol can not be used as it is for authenticating roaming clients requesting network access.

[1] Such as the DS80C390, by Maxim/Dallas Semiconductor

The pre-authentication issue is linked to the semantics of the Kerberos protocol. Since the Kerberos operations follow the *push* authentication model (see Section 3.3.3), it is normal to find the pre-authentication issue in Kerberos.

Inter-realm trust chain

The cross-realm operations assume that the realms have a direct or indirect trust relationships as explained in Section 4.2.1. The trust relationship is maintained trough shared keys between the KDCs of the realms. Direct trust relationship between realms is hard to maintain since it requires that KDCs maintain shared keys with all KDCs of the known realms. It is not a scalable trust management solution especially, if we consider that the inter-realm agreements are subject to frequent changes (or if the business model imposes that the realm must cooperate even with unknown realms).

When cross-realm authentication is taking place, the process of acquisition of a TGT (see Section 4.2.2) depends not only on the user's home KDC and the KDC of the service provider, but it also depends on a set of third-party realms that are part of the authentication path. If any of the KDCs belonging to the authentication path is not available, the end-point realms can not perform cross-realm operations.

4.4 Proposed extensions for the Kerberos protocol

Since the early stages of Kerberos, critics raised, mainly regarding the cross-realm key management and the related scalability issues for use in large and multi-networked environments. For these reasons, proposals and variants of the protocol were proposed to improve the Kerberos authentication system. Probably, the most relevant proposals consists on the use of public key cryptography as solution for the scalability problem found in initial authentication of clients and in cross-realm operations.

In this chapter, we survey the extensions of the Kerberos protocol. We start by explaining the PKINIT [28] extension. Then, we explain the operations of the PKCROSS [29] proposal which is its self based on the PKINIT extension and which defines a method for using public key cryptography in cross-realm authentication. Finally we explain the IAKERB [30] proposal, that suggests the use of store and forward proxies in the edge of the access network in order to allow, for instance, the use of Kerberos authentication between clients and network access servers.

4.4.1 The PKINIT proposal

The PKINIT [28] Internet draft specifies an extension to the Kerberos protocol for using public key authentication between the user and the KDC instead of using the classic symmetric key cryptography based authentication. This extension provides a method for integrating public key cryptography into the initial authentication exchange, by using asymmetric-key signature and/or encryption algorithms in pre-authentication data fields.

The PKINIT extension defines the following changes to the Kerberos protocol :

1. The client indicates the use of public-key authentication by including a special pre-authenticator in the initial request. This pre-authenticator contains the client's public-key data and a signature.

2. The KDC tests the client's request against its authentication policy and trusted Certification Authorities (CAs).

3. If the request passes the verification tests, the KDC replies as usual, but the reply is encrypted using either:

   - a. A key generated through a Diffie-Hellman (DH) key exchange [31] [32] with the client, signed using the KDC's signature key; or
   - b. A symmetric encryption key, signed using the KDC's signature key and encrypted using the client's public key.

     Any keying material required by the client to obtain the encryption key for decrypting the KDC reply is returned in a pre- authentication field accompanying the usual reply.

4. The client validates the KDC's signature, obtains the encryption key, decrypts the reply, and then proceeds as usual.

The PKINIT extension allows KDCs to provide TGTs to clients even when the clients do not share a secret key with the KDC. The PKINIT proposal, thus, makes the Kerberos protocol scalable for a large number of users and realms. The drawbacks are that since PKINIT uses public-key cryptography, the overhead of the authentication messages can increase, especially for small devices with low computational power. For this reason, we exclude PKINIT from the solution space of the scalability problem. In [33] the authors explore the performance of authentication from mobile devices, and an alternative solution based on PKINIT called M-PKINIT is described in the aim of providing lighter version of the PKINIT extension. The M-PKINIT proposal tries to minimize public key operations in the client side, however it introduces a new protocol component that consists in an authentication proxy, which may increase the complexity of the protocol and make the deployment of such framework hard to maintain. More over, the PKINIT extension amplifies the DoS attack vulnerability of Kerberos. The KDCs become more vulnerable to DoS attacks since the public-Key cryptography operations performed by the KDC are relatively expensive.

4.4.2 The PKCROSS proposal

The PKCROSS [29] proposal defines extensions to the Kerberos protocol that provides a method for using public key cryptography to enable cross-realm authentication. This extension allows KDCs from different realms to generate cross-realm key dynamically. This leverage the scalability issue of the cross-realm operations since no inter-realm keys have to be maintained. Using public-key cryptography, KDCs can generate cross-realm keys when needed. The PKCROSS proposal, reuses the extensions defined by the PKINIT proposal in the sense that it uses te same packet formats and fields used in PKINIT.

In the following, we explain the basic operations of PKCROSS. For this purpose, let us assume that some client which belongs to a realm "l", requests credentials from his realm's KDC "KDC-L" in order to access a service offered in a remote realm "r". We will refer to the KDC of the remote realm as "KDC-R". The basic operation of the PKCROSS protocol are specified as follows:

1. The client submits a request to the local KDC asking for credentials for the remote realm. This is just a typical cross-realm request that may occur with or without PKCROSS.

2. The local KDC submits a PKINIT request to the remote KDC to obtain a "special" PKCROSS ticket. This is a standard PKINIT request, except that PKCROSS flag (bit 9) is set in the kdc-options field in the AS_REQ. Note that the service name in the request is for "pkcross/realm@REALM" instead of "krbtgt/realm@REALM".

3. The remote KDC responds as per PKINIT, except that the ticket contains a TicketExtension, which contains policy information such as lifetime of cross realm tickets issued by KDC-L to a client. The local KDC must reflect this policy information in the credentials it forwards to the client.

   In order to generate the reply message, KDC-R creates a cross-realm key (that will be shared with KDC-L) and creates a PKCROSS ticket. Call this ticket XTKT_(l,r) to indicate that this ticket is used to authenticate the local KDC to the remote KDC. The XTKT ticket is encrypted using the secret key of the principal "pkcross" registered in KDC-R. The cross-realm key is included in XTKT and in the encrypted part of the PKINIT reply (the encrypted part is encrypted using the public-key of KDC-L or a DH generated key). Note that the cross-realm key is this not kept in the memory of the KDC-R.

4. The local KDC obtains the cross-realm key by decrypting the encrypted part of the PKINIT reply using its public-key for example. It passes then a ticket, TGT_(c,r) (the cross realm TGT between the client and remote KDC), to the client. This ticket contains in its TicketExtension field the ticket, XTKT_(l,r), which contains the cross-realm key. The TGT_(c,r) ticket is encrypted using cross-realm key and contains a session key that will be shared between the client and the remote KDC. The local KDC may optionally include another TicketExtension type that indicates the hostname and/or IP address for the remote KDC. The encrypted part of the reply of the local KDC contains the same session key included in TGT_(c,r). The reply is then encrypted using the client's secret key.

5. The client uses its secret key to decrypt the reply from KDC-L. It obtains the session key with KDC-R. The client then builds the authenticator and a TGS request then submits the request directly to the remote KDC, as before.

6. The remote KDC extracts XTKT_(l,r) from the TicketExtension . It uses the secret key of the "pkcross" principal to decrypt the XTKT key and obtain the cross-realm key. Using the cross-realm key, KDC-R can decrypt the encrypted part of TGT_(c,r). Once the contents of the TGT_(c,r) are decrypted and verified against the authenticator, the KDC-R can issue a service ticket to the client.

The Figure 4.4 illustrates the PKCROSS protocol. We assume that the user wishes to access a service offered in a remote realm R. For that, the client needs to have a TGT that can be used to obtain service ticker from the KDC of the remote realm. The user first requests cross-realm credentials from its home KDC (1), the home KDC uses the PKCROSS protocol extension and generates a cross-realm TGT (2). The client uses the cross-realm TGT to obtain the service ticket from the KDC of the remote KDC.

The PKCROSS extension allows the use of public key to avoid the scalability problem related to the trust management between realms. The trust between realms can be established dynamically. There is no need to have pre-shared cross-realm keys between two realms to enable cross-realm authentication. Furthermore, the PKCROSS extension, eliminates the need for the

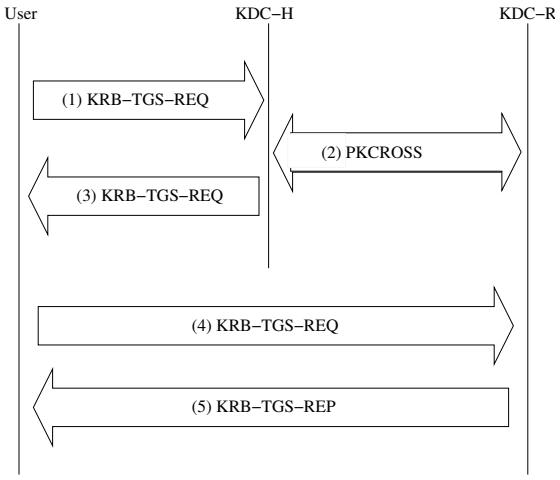


Figure 4.4: Using PKCROSS for obtaining cross-realm TGTs

concept of trust path. The cross-realm operations benefit from a direct communication model, in which, the KDC of the home realm can provide cross-realm TGT to the client without the need for referrals as specified by the current Kerberos standard. All these advantages makes the PKCROSS proposal suitable for use in frameworks where a large number of realms are involved, and more specifically, when the maintenance of cross-realm agreements is hard to maintain due to the large number of unknown realms that the clients from the home realm can interact with. The PKCROSS extension however, does not resolve the problem of using Kerberos for network access authentication. It does not resolve neither the problem of user to KDC message exchanges across realm boundaries.

4.4.3 The IAKERB proposal

The IAKERB [30] extension, defines a mechanism that enables a client to obtain Kerberos tickets for services where the KDC is not accessible to the client. Some common scenarios where lack of accessibility would occur are when the client does not have an IP address prior to authenticating to an access point, the client is unable to locate a KDC, or a KDC is behind a firewall.

The IAKERB proposal defines a proxy service which is an application server called **IAKERB proxy**. The proxy acts as a protocol gateway between the client and the KDC. The IAKERB proxy is responsible of locating the KDC to which the client's request is destinated. In this manner, the client can communicate trough the proxy with its home KDC and obtain cross-realm ticket for authentication with a NAS in a visited realm.

In the following, we illustrate the use of IAKERB using an example. We assume that a roaming user wishes to authenticate to a network access server using Kerberos credentials. We assume, also, that an IAKERB proxy is deployed in the edge of the access network and that it is reachable from the client's device. We will refer to the KDC of the home realm as "KDC-H" while referring to the KDC of the visited realm as "KDC-V".

The Figure 4.5 shows the IAKERB proxy operations. The client uses the IAKERB proxy to communicate with his own home KDC in order to obtain cross-realm TGT. The cross-realm TGS request is forwarded by the IAKERB proxy to the user's home KDC (1). The home KDC then follows the standard Kerberos protocol specification of the cross-realm protocol (or it can

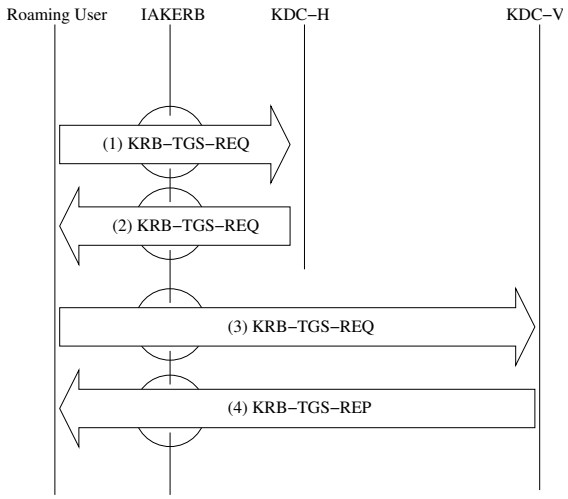


Figure 4.5: IAKERB proxy for authenticating roaming users

use PKCROSS). If the home KDC and KDC-V share cross-realm keys, then KDC-H provides a cross-realm TGT that can be used with the KDC of the visited realm's KDC (KDC-V) (2). If no cross-realm keys are shared between the two KDCs then the referral protocol is used. The client thus may have to communicate with several KDCs that are in the trust path between the home KDC and the visited realm's KDC. During these cross-realm exchange, the IAKERB proxy relays the messages between the client and the different KDCs from different realms. Finally, when the client obtains a cross-realm TGT for KDC-V, it requests a service ticket using this cross-realm TGT (3). The TGS request is forwarded by the IAKERB proxy to the local KDC (KDC-V) which replies to the client and provides the requested service ticket (4). The service ticket is is then used by the client to authenticate with the NAS for example.

The IAKERB allows the authentication of users when the KDCs are not reachable. A typical use of IAKERB would be authentication for network access. However the IAKERB proposal, does not resolve the issues related to cross-realm trust management and the fact that the cross-realm authentication is client-centric.

4.4.4 IAKERB with PKCROSS for authentication of roaming users

In this section, we show how IAKERB and PKCROSS protocol can be combined and used to offer roaming support. The IAKERB proxy is deployed in the edge of the access network while PKCROSS is used between KDCs to dynamically generate cross-realm keys.

As shown in the Figure 4.6, the roaming user can request cross-realm TGT from the KDC of his home realm (1). The IAKERB proxy is the component that allows this indirect communication. The home KDC then uses PKCROSS with the KDC of the visited realm (2). The result of the PKCROSS exchange will allow the home KDC to generate a cross-realm TGT in a two messages. The cross-realm TGT then is delivered to the client (3). The IAKERB acting as a store and forward agent does not need to be trusted, since the Kerberos protocol semantics are kept and the assumption of being in open environment is still being respected by both the client and the KDC. Finally, the TGS exchange takes place between the roaming user and the KDC of th visited realm (4) and (5). The client uses the cross-realm TGT provided by the home KDC. The KDC of the visited realm will process the client's TGS request as defined in the PKCROSS

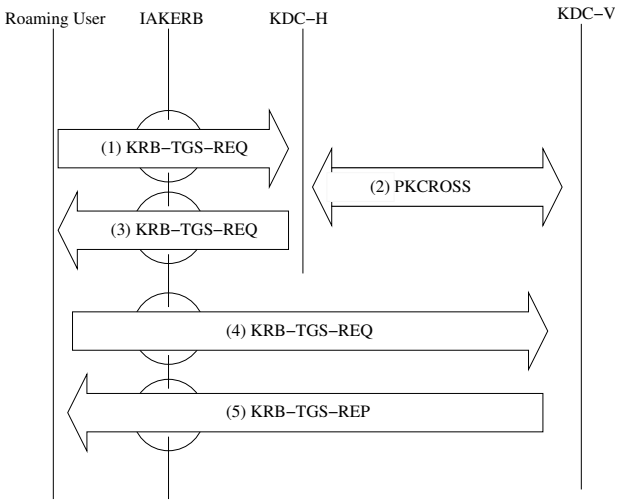


Figure 4.6: Using IAKERB with PKCROSS to authenticate roaming users

extension defined in Section 4.4.2.

This example scenario was meant to demonstrate the combination of the IAKERB and PKCROSS extensions. Indeed both proposals can work together theorically. The union of these extensions produces an authentication framework that have the following advantages :

1. Possibility of using Kerberos as authentication for network access. In deed the IAKERB extension resolve the pre-authentication issue described in Section 4.3.2.

2. Scalable cross-realm trust model based based on a PKI and public key for authentication and generation of cross-realm keys. The PKCROSS protocol does not rely on pre-shared keys neither on a trust path. Thus, the issue identified in Section 4.3.2 can be resolved.

3. Minimize the message exchanges. Thanks to the PKCROSS protocol, the client have to communicate only with the home KDC. This TGS exchange will result on cross-realm TGT. In this manner, the issue related to the protocol overhead when the client device have limited computational power (issue described in Section 4.3.2) is resolved.

However, the framework based on IAKERB and PKCROSS still can not avoid client to KDC exchanges across realm boundaries. As discussed in Section 4.3.2. If the KDC is open to the public. There is a risk that the KDC be victim of attacks coming from any where in the Internet. The description of the attacks and risks that the KDC can be victim of are out of the scope of this paper. A study of the weaknesses of the Kerberos protocol are detailed in [34].

4.5 Conclusion

The Kerberos protocol have many advantages that makes it an ideal candidate for building AAA frameworks targeting the emerging markets based on Internet and small mobile computing devices. However there is some issues related to cross-realm authentication that still need to be addressed. Some proposals suggested improvements to the basic protocol. The IAKERB and PKCROSS protocol could resolve the majority of the issues that we identified. However, we estimate that a better approach can be developed. In deed, the IAKERB extension involves

heavy changes on the AAA framework since it requires the use of a new protocol component (a proxy). The PKCROSS on the other hand is based on the PKINIT draft which is its self still not adopted standard.

In the next chapter, we present our approach to address the issues identified in Section 4.3.2. Our proposal defines an unified solution for the cross-realm authentication issues. It does not involve new actors such as proxies or agents, and it does not rely on other work except the standard specification. This gives the advantage to our approach when compared to the IAKERB and PKCROSS extensions.

Chapter 5

The XKDCP proposal

The XKDCP protocol, that we propose in this chapter, defines a cross-realm protocol between Kerberos Key Distribution Centers for providing Kerberos cross-realm credentials to clients. The XKDCP protocol defines two sub-protocols "ASP" and "TGSP" used respectively in roaming and remote service access scenarios defined in Section 2.2. The XKDCP protocol addresses the issues listed in the Section 4.3.2. The objective is to provides the same benefits as PKCROSS and IAKERB combined, with the advantage of resolving the issue related to KDC exposure which is still not resolved by the existing Kerberos extensions. In the following sections, we start by giving an overview of the XKDCP protocol. We explain then, the operations and the message specifications of the ASP protocol. We then, do the same with the TGSP protocol. An example is used to explain the operations of the ASP protocol. The reader is highly encouraged to follow the example for better understanding

5.1 Protocol Overview

The semantics of the Kerberos protocol requires that the client asks credentials only to the KDC of the realm to which the client belongs (Home KDC). The idea behind the XKDCP protocol is to allow clients to obtain Kerberos credentials from the closest KDC, no matter if the closed KDC is the home KDC or if it is a KDC that belongs to a visited realm(this occurs in the roaming scenario). With our approach, the client does not need to communicate with a KDC that is located across the current realm's boundaries. Which resolves many issues related to cross-realm authentication.

In cross-realm scenarios [1], the user must obtain permission (or cross-realm ticket) from the KDC of his/her home realm, in order to be able to communicate with the KDC of the service provider. Only then, the client can ask for service tickets necessary for accessing the actual application service. In the XKDCP proposal, we adopt another semantic which consists on the following : "Clients are not aware of their location, neither of the location of the application service. Clients use the topologically closest KDC to ask for credentials. All cross-realm exchanges

[1] Cross-realm scenarios refer to the situation where the realm of the service provider and the realm to which the client belongs are different.

until the acquisition of service tickets are handled by the KDCs". This implies that the KDCs use a protocol to build the credentials needed by the client. The protocol must not, however, reveal the service keys neither the user's secret key. The XKDCP allows a roaming user to obtain a TGT from the KDC of the visited realm (not from his/her home KDC) thanks to the ASP protocol. The TGSP protocol in the other hand is used in remote service access scenario, to allow users to obtain service tickets, for services deployed in remote realms, from the user's home KDC. The TGSP protocol eliminates the need for the client to obtain a cross-realm TGT in order to obtain a service ticket for a remote service.

The TGSP protocol get its name from the fact that it defines a protocol between Ticket Granting Services of two realms. The goal of TGSP is to provide a service ticket trough the collaboration of two TGSs. The ASP protocol in the other hand takes its name from the fact that it defines a protocol between Authentication Servers. The goal of ASP is to provide a TGT to users in roaming scenarios.

In the following sections, we explain the principle and the details of the XKDCP protocol components ; ASP and TGSP.

5.2 The ASP protocol

In case of a roaming scenario, the client might wish to access services offered by the visited realm (Ex. Internet access, Online library, Multimedia streaming). The first objective of the client is to obtain a TGT usable with the KDC of the visited realm (Visited KDC[2]). This TGT will be used to ask the visited KDC for tickets to access the offered services. The ASP protocol defines a message exchange between the Visited KDC and the Home KDC. This exchange when used , allows the visited KDC to deliver a TGT to roaming users.

The Figure 5.1 depicts the system architecture and the message exchange between the different entities. The client sends an AS request to the visited KDC (KDC@V). The Visited KDC then, initiates an ASP exchange with the user's home KDC (1) and (2) in Figure 5.1. The result of the ASP exchange is that the Visited KDC can now issue an AS-REP message containing a TGT an a TGS session key encrypted using the user's key shared with the home KDC.

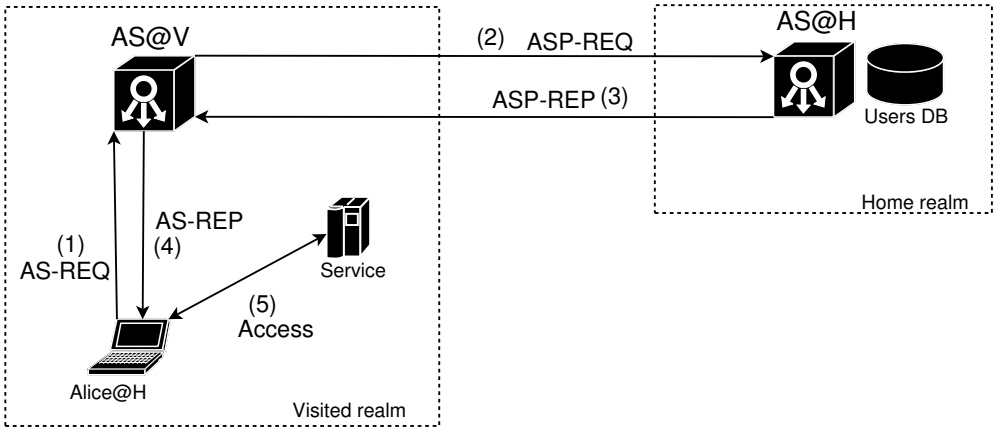


Figure 5.1: ASP : The Inter Authentication Server Protocol

[2]In roaming scenarios, the visited KDC refers to the Key distribution center of the realm where the user is physically located. Hence, when the user is located in his home realm, there is no notion of visited KDC.

5.2.1 The ASP protocol operations

In the following sections, we describe in details the operations that allow roaming clients to obtain TGTs in visited realms using the ASP protocol.

The client discovers the KDC of the visited realm

When a client wishes to access some service. It must be sure to have a TGT for the local KDC (the visited KDC in the roaming scenario). If a TGT must be obtained, then the client needs first to locate the KDC of the visited realm. If the user have information about the current location, the informations can be provided manually to the Kerberos client. However, a discovery protocol would be more convenient.

The discovery of information about the local realm can be performed using multicast advertisements in the edge of the network. The client just needs to join the multicast group and obtain all the information needed. The information conveyed to the client must at least include the local realm name and the IP address of the KDC.

The client should use the discovery protocol if any of the following conditions is true :

- The client does not have a valid TGT.
- The currently 'local' KDC is not reachable : This informs the client that there was a possible cross-realm hand-over that happened.

The client sends an AS request to the visited realm's KDC

The client sends a Kerberos AS-REQ message to the Visited KDC. This request is a normal AS-REQ message as defined in [7]. The AS-REQ message is depicted in Listing 5.1. The AS-REQ message is used in the standard Kerberos protocol by clients to ask for TGTs from their home KDC. When the ASP protocol is used between KDCs of the realms involved in the scenario, the user consider him self at home, and ask for a TGT as if he was located in the home realm and querying his home KDC.

In situations where the network access is not provided to the client, care must be taken not to block the AS-REQ messaged from reaching the local KDC. Generally, the access network is protected trough enforcement points and firewalls. However the access control framework must allow the client to send AS request to the Visited KDC located at the back end. The client can use global addresses configured using the router advertisements from the access routers at the edge of the access network. The address is however blocked by the access control framework until the client is authenticated. The Internet connectivity is not needed by the client to obtain the Kerberos credentials. The connectivity granted to the client must be calibrated to only allow the client Kerberos requests to reach the local KDC.

The KDC of the visited realm issues an ASP request

Based on the realm name of the client (field [42] Realm in Listing 5.1), the visited realm's KDC knows that it is dealing with a roaming client when the client's home realm name is different from the local realm name. Normally, the visited KDC does not know anything about the roaming user. Thus, if it only relies on its local resources, it can not issue the AS-REP message. The AS-REP message, indeed, contains one part (field [6] "EncASRepPart" in Listing 5.2) that must

Listing 5.1: The AS-REQ message

```
AS-REQ          ::= SEQUENCE {
        pvno            [1] INTEGER (5) ,
        msg-type        [2] INTEGER (10),
        padata          [3] SEQUENCE OF PA-DATA OPTIONAL,
        req-body        [4] AS-REQ-BODY
}
AS-REQ-BODY     ::= SEQUENCE {
        kdc-options             [40] KDCOptions,
        cname                   [41] PrincipalName OPTIONAL,
        realm                   [42] Realm              -- Client's realm
        sname                   [43] PrincipalName OPTIONAL,
        from                    [44] KerberosTime OPTIONAL,
        till                    [45] KerberosTime,
        rtime                   [46] KerberosTime OPTIONAL,
        nonce                   [47] UInt32,
        etype                   [48] SEQUENCE OF Int32 -- EncryptionType
        addresses               [49] HostAddresses OPTIONAL,
        enc-authorization-data  [410] EncryptedData OPTIONAL,
        additional-tickets      [411] SEQUENCE OF Ticket OPTIONAL
}
```

be encrypted using the user's private key. The only entity that can build the "EncASRepPart" is the home KDC[3].

The ASP protocol enters in action at this stage to allow the visited KDC to create the "EncASRepPart" in collaboration with the home KDC. The Visited KDC get the home realm name of the user from the AS-REQ message. It then needs to determine how to contact the Home KDC. For this purpose, the Visited KDC can use its configuration files or DNS SRV resource records as defined in [35].

The other important part of the AS-REP message is the Ticket (field [5] Ticket in Listing 5.5). The Ticket (aka TGT in case of AS exchange) is encrypted by the visited KDC using the key of the "krbtgt" service, hence, this part of the AS-REP message can be issued without need to contact any third party.

The ASP-REQ message must provide all the informations necessary for the home KDC to build the EncASRepPart. The Visited KDC uses the informations from the client's AS-REQ message to issue an ASP-REQ to the home KDC. The only information needed in the home KDC to build the EncASRepPart and not provided by the Visited KDC is the *TGS session Key* (field [60] in Listing 5.2 and [531] in Listing 5.3). This field is generated by the home KDC.

The TGS Session key is a secret key that is included in both the TGT and the "EncASRepPart" section of the AS-REP message. It must not be revealed to anyone except to the KDCs, and to the user. For this reason, we use a public key based encryption mechanism to protect this sensible information. The public key mechanism is used also to perform mutual authentication between the Visited KDC and the Home KDC. The Visited KDC, since it is the initiator of the ASP exchange, includes his public key and a signature in the ASP-REQ. The home KDC verifies the identity of the Visited KDC by checking the authenticity of the public key and the validity of the signature. If the signature is verified, the Home KDC generates the TGS Session Key and the "EncASRepPart" (which includes the same Session key). It then issues an ASP-REP

[3]In all scenarios, the home KDC refers to the Key distribution Center of the home realm of the user. The term home KDC is used In roaming scenario, as well as in remote service access scenario.

Listing 5.2: The AS-REP message

```
AS-REP          ::= SEQUENCE {

        pvno            [0] INTEGER (5),
        msg-type        [1] INTEGER (11),
        padata          [2] SEQUENCE OF PA-DATA OPTIONAL
        crealm          [3] Realm,
        cname           [4] PrincipalName,
        ticket          [5] Ticket,         -- encrypted using the TGS key
        enc-part        [6] EncASRepPart    -- encrypted using the user key

}

EncASRepPart    ::= SEQUENCE {

        key             [60] EncryptionKey, -- TGS session key
        last-req        [61] LastReq,
        nonce           [62] UInt32,
        key-expiration  [63] KerberosTime OPTIONAL,
        flags           [64] TicketFlags,
        authtime        [65] KerberosTime,
        starttime       [66] KerberosTime OPTIONAL,
        endtime         [67] KerberosTime,
        renew-till      [68] KerberosTime OPTIONAL,
        srealm          [69] Realm,
        sname           [610] PrincipalName,
        caddr           [611] HostAddresses OPTIONAL

}
```

Listing 5.3: The Kerberos Ticket format

```
Ticket          ::=  SEQUENCE {

        tkt-vno         [50] INTEGER (5),
        realm           [51] Realm,
        sname           [52] PrincipalName,
        enc-part        [53] EncTicketPart

}

EncTicketPart   ::=  SEQUENCE {

        flags                   [530] TicketFlags,
        key                     [531] EncryptionKey, -- session key
        crealm                  [532] Realm,
        cname                   [533] PrincipalName,
        transited               [534] TransitedEncoding,
        authtime                [535] KerberosTime,
        starttime               [536] KerberosTime OPTIONAL,
        endtime                 [537] KerberosTime,
        renew-till              [538] KerberosTime OPTIONAL,
        caddr                   [539] HostAddresses OPTIONAL,
        authorization-data      [5310] AuthorizationData OPTIONAL

}
```

message containing the EncASRepPart and the Session Key encrypted using the public key of the Visited KDC. More details on the operations performed by the Home KDC on reception of an ASP-REQ message are detailed in the next Section.

The Home KDC receives the ASP request

When the Home KDC receives an ASP-REQ from the Visited KDC, it performs the following operations :

- The public key of the Visited KDC is verified against the corresponding certificate authority. The signature is then verified. If these operations succeed the Visited KDC is trusted. If the authentication fails the ASP-REQ message is simply dropped.
- Extract the informations about the user. The home realm of the user must correspond to the local realm name of the KDC. If the realm name does not match , an error is reported to the Visited KDC. The Home KDC then look-up the user database to extract the secret key that corresponds to the roaming client's principal name. If the principal name is not found in the database, the home KDC returns an error to the Visited KDC.
- A random session key is generated. This session key will be the *TGS session Key* (field [60] "EncryptionKey" in Listing 5.2 and [531] in Listing 5.3).
- The Home KDC generates an "*EncASRepPart*" data structure using the informations provided in the ASP-REQ message. The newly generated session key is included in the "EncASRepPart".
- The session key is encrypted using the public key of the Visited KDC. And the "EncASRepPart" containing the same session key is encrypted using the user's secret key.
- The ASP-REP message containing the "EncASRepPart" and the encrypted TGS session key, is sent back to the Visited KDC.

In the ASP-REP message the home KDC includes its public key and a signature. This will allow the Visited KDC to trust the ASP-REP message after authenticating the Home KDC.

The Visited KDC receives the ASP reply

When the Visited KDC receives the ASP-REP message from the home KDC, it performs the following operations :

- The public key of the Home KDC is verified against the corresponding certificate authority. The signature is then verified. If these operations succeed, the Home KDC is trusted. If the authentication fails, the ASP-REP message is simply dropped and the user is informed about the failure of the authentication.
- The *TGS session key* is extracted from the ASP-REP message, the Visited KDC uses his public key for decrypting the corresponding field of the ASP-REP message.
- The Visited KDC creates a Ticket. The *TGS session key* obtained from the previous step is placed in the encrypted part of the ticket (field [531] of the "EncTicketPart" as shown in Listing 5.3).

- The AS-REP message is now ready, the Visited KDC puts the ticket along with the "EncASRepPart" (received from the Home KDC) in an AS-REP message and send it to the roaming user.

The client receives the AS reply message

When the roaming Kerberos client receives an ASP-REP message, it processes it as specified in RFC 4120. The Ticket Granting Ticket is extracted from the message, then the "EncASRepPart" is decrypted using the user's secret key shared with the Home KDC. The client then obtains the TGS session key and can issue TGS-REQ messages to obtain service tickets.

5.2.2 ASP protocol message specification

There are several alternatives for implementing the ASP protocol. The protocol can be implemented as an extension using the typed holes in the Kerberos messages. Or, it can be implemented using completely different message formats. The former approach however is the recommendation of the Kerberos working group as described in [7]. Kerberos provides a general mechanism for protocol extension. The protocol messages contain typed holes (sub-messages) that contains a general purpose field along with an integer that defines how to interpret the field data. The field in question is the "padata" field (field number [3] in the AS-REQ message shown in Listing 5.1 and field [2] in the AS-REP message, Listing 5.2). The padata field contains two sub-fields, "padata-type" and "padata-value". The first field indicated the type of the data transported by the padata field. "padata-value" contains the data its self. The padata field is shown in Listing 5.4. The Kerberos entities (KDC or clients) refer to the "padata-type" to decide how to process the additional data embodied in the Kerberos message. The Kerberos extensions are used for this purpose to interpret the padata fields. The different values that the padata-type can take are standardized and listed in [7].

The ASP-REQ message

The ASP-REQ message is an extended AS-REQ message. It is built by the Visited KDC as follows :

- The Visited KDC creates a copy of the AS-REQ received from the client.
- msg-type (field [2] of the AS-REQ message) is set to the value 660. The msg-type 660 means that the message is an ASP-REQ message and must be processed as such, using the ASP protocol specification.
- The padata (field [3] of the AS-REQ message) is filled with PA-DATA structure as defined in Listing 5.4.
  - The padata-type is set to 111 (XKDCP-AUTH-DATA)[4].
  - The LRealm field contains the realm name of the Visited KDC.
  - The CRealm field contains the realm name of the client, it is extracted from the "realm" field of the AS-REQ.

[4]This Pre-Authentication data type value is not used according the the RFC 4021

Listing 5.4: ASP-REQ PA-DATA

```
PA-DATA         ::= SEQUENCE {
                padata-type      [1] Int32,        -- 111 XKDCP-AUTH-DATA
                padata-value     [2] XKDCP-AUTH-DATA
}

XKDCP-AUTH-DATA {
    kippu           [20] Kippu, -- Used in ASP-REP
    LRealm          [21] Realm, -- The local realm name
    CRealm          [21] Realm, -- The realm name of the client
    SubjectName     [22] OctetString,
    PublicKeyValue  [23] PublicKey,
    Signature       [24] Signature
}
```

- The SubjectName is the name associated to the X.509 public key of the Visited KDC [36].
- The X.509 public Key of the Visited KDC [36].
- Signature is a digital signature computed using the Visited KDC's private key as specified in [37].

The ASP-REP message

The ASP-REP message generated by the home KDC is an extended AS-REP message that is built as follows :

- The home KDC generates an random session key (TGS Session Key).
- An EncASRepPart is created and encrypted using the user's key. The EncASRepPart contains the randomly generated TGS session key. The EncASRepPart is created just as for a normal AS-REP message except that the field "srealm" contains the realm name of the Visited KDC instead of the home realm's name. The value of "srealm" is obtained from the field "LRealm" of the XKDCP-AUTH-DATA (see Listing 5.4).
- The Home KDC prepares a padata field of type PA-DATA (as defined in Listing 5.4). The padata field will contain the public Key and the signature of the Home KDC so that the visited KDC can verify the authenticity of the Home KDC if he wishes.
- The TGS session key is encrypted using the Public Key of the Visited KDC and placed in the "padata" field. We will use a new type "Kippu" for the encrypted session key. The Kippu type contains two fields "kippu-length" that gives the length of the encrypted data, and "kippu-data" which contains the actual Session key encrypted using the public key of the Visited KDC. The " kippu" structure also contains the IP address of the roaming user as extracted from the "addresses" field of the AS-REQ-BODY of the ASP-REQ message. The padata field of type XKDCP-AUTH-DATA looks finally as shown in Listing 5.5.
- The home KDC creates an AS-REP and fills all the field as for a normal AS-REP, except the "padata" and "EncASRepPart" as explained above. Furthermore, the "ticket " field

is not used in the ASP-REP. The field is left empty. Finally, the "msg-type" field is set to 661 indicating that the message is an ASP message.

Finally, the ASP-REP message looks like in Listing 5.5.

5.2.3 Example of roaming scenario using ASP

As an example, let us suppose that a user called "Alice" that belongs to the realm named "JAIST" is in a roaming scenario, and the name of the visited realm is "MIT". Alice wants to authenticate to the Kerberized network access server (NAS) to gain Internet access. For that, Alice starts by using her Kerberos client to join a special multicast group and listen to the advertisements in order to obtain information about the local realm. These informations include, the visited realm name "MIT", the IP address of the KDC and the port number. The multicast messages are sent by the network access servers located at the edge of the network.

Alice then creates an AS-REQ destinated to the KDC of the realm "MIT" (KDC@MIT). Listing 5.6 shows the AS-REQ.

On reception of the AS-REQ from Alice, KDC@MIT notices that the realm name "JAIST" is different from "MIT". KDC@MIT assumes then, that Alice is a roaming user which belong to the realm JAIST. The KDC@MIT, locates the KDC of the realm JAIST by looking in its configuration file. Then, the ASP-REQ message is built as described in Section 5.2.2 and sent to KDC@JAIST. The ASP-REQ message and its contents is depicted in Listing 5.7.

On reception of the ASP-REQ message, the KDC in the realm JAIST, notice that the "msg-type" of the request is set to 660, this indicates that the ASP extension must be used to process the message. KDC@JAIST, creates an ASP-REP message as specified in Section 5.2.2. The ASP-REP sent from KDC@JAIST looks as shown in Listing 5.8.

Finally KDC@MIT receives the ASP-REP message. The "msg-type" field notifies KDC@MIT that the message have to be treated as an ASP-REP using the ASP extension. The processing is defined in Section 5.2.1. The KDC of the realm MIT issues an AS-REP message to Alice (The IP address of Alice is extracted from the "caddr" field of the "kippu" structure from the ASP-REP message.

Alice processes the AS-REP message as described in [7] and obtains a TGT and a session key that allow her to obtain service ticket from KDC@MIT and authenticate to the network access server, then gain Internet access.

Listing 5.5: The ASP-REP message

```
ASP-REP          ::= SEQUENCE {

        pvno             [0] INTEGER (5),
        msg-type         [1] INTEGER (661),
        padata           [2] PA-DATA,          -- Contains XKDCP-AUTH-DATA
        crealm           [3] Realm,
        cname            [4] PrincipalName,
        ticket           [5] Ticket,           -- Not used.
        enc-part         [6] EncASRepPart  -- encrypted using the user key

}

PA-DATA        ::= SEQUENCE {

            padata-type      [21] Int32,         -- 111 XKDCP-AUTH-DATA
            padata-value     [22] XKDCP-AUTH-DATA

}

XKDCP-AUTH-DATA {

    kippu            [220] Kippu,
    LRealm           [221] Realm,
    CRealm           [221] Realm,
    SubjectName      [222] OctetString,
    PublicKeyValue   [223] PublicKey,
    Signature        [224] Signature

}

Kippu     ::= SEQUENCE {

        kippu-length     [50] INTEGER ,
        caddr            [50] HostAddresses  -- IP address of the roaming client
        kippu-data       [51] OctetString    -- TGS session key
                                             encrypted using the PK
                                             of the Visited KDC

}

EncASRepPart     ::= SEQUENCE {

        key              [60] EncryptionKey, -- TGS session key
        last-req         [61] LastReq,
        nonce            [62] UInt32,
        key-expiration   [63] KerberosTime OPTIONAL,
        flags            [64] TicketFlags,
        authtime         [65] KerberosTime,
        starttime        [66] KerberosTime OPTIONAL,
        endtime          [67] KerberosTime,
        renew-till       [68] KerberosTime OPTIONAL,
        srealm           [69] Realm,           -- The visited realm name
        sname            [610] PrincipalName,
        caddr            [611] HostAddresses OPTIONAL

}
```

Listing 5.6: The AS-REQ issued by Alice

```
AS-REQ          ::= SEQUENCE {

        pvno                [1] (5),
        msg-type            [2] (10),
        padata              [3] (NA),
        req-body            [4] AS-REQ-BODY

}

AS-REQ-BODY     ::= SEQUENCE {

        kdc-options                 [40] (0),
        cname                       [41] (ALICE),
        realm                       [42] (JAIST),
        sname                       [43] (krbtgt),
        from                        [44] (NA),
        till                        [45] (NA),
        rtime                       [46] (NA),
        nonce                       [47] (5082342423),
        etype                       [48] (DES-CBC-MD5),
        addresses                   [49] (IP address of Alice),
        enc-authorization-data      [410] (NA),
        additional-tickets          [411] (NA)

}
```

Listing 5.7: The ASP-REQ issued by KDC@MIT

```
ASP-REQ         ::= SEQUENCE {

        pvno                [1] (5),
        msg-type            [2] (660),
        padata              [3] PA-DATA,
        req-body            [4] AS-REQ-BODY

}

AS-REQ-BODY     ::= SEQUENCE {

        kdc-options                 [40] (0),
        cname                       [41] (ALICE),
        realm                       [42] (JAIST),
        sname                       [43] (krbtgt),
        from                        [44] (NA),
        till                        [45] (NA),
        rtime                       [46] (NA),
        nonce                       [47] (5082342423),
        etype                       [48] (DES-CBC-MD5),
        addresses                   [49] (IP address of Alice),
        enc-authorization-data      [410] (NA),
        additional-tickets          [411] (NA)

}

PA-DATA         ::= SEQUENCE {

            padata-type     [1] (111) -- XKDCP-AUTH-DATA
            padata-value    [2] XKDCP-AUTH-DATA

}

XKDCP-AUTH-DATA {

    kippu           [20] (NA)
    LRealm          [21] (MIT),
    CRealm          [21] (JAIST),
    SubjectName     [22] (KDC-MIT),
    PublicKeyValue  [23] (PK of KDC@MIT),
    Signature       [24] (A signed message)

}
```

Listing 5.8: The ASP-REP issued by KDC@JAIST

```
ASP-REP          ::= SEQUENCE {

        pvno             [0] (5),
        msg-type         [1] (661),
        padata           [2] PA-DATA,    -- Contains XKDCP-AUTH-DATA
        crealm           [3] (JAIST),
        cname            [4] (ALICE),
        ticket           [5] (NA),
        enc-part         [6] EncASRepPart  -- encrypted using Alice's secret key

}

PA-DATA          ::= SEQUENCE {

           padata-type      [21] (111)
           padata-value     [22] XKDCP-AUTH-DATA

}

XKDCP-AUTH-DATA {

    kippu            [220] Kippu,
    LRealm           [221] (JAIST),
    CRealm           [221] (JAIST),
    SubjectName      [22] (KDC-JAIST),
    PublicKeyValue   [23] (PK of KDC@JAIST),
    Signature        [24] (A signed message)

}

Kippu    ::= SEQUENCE {

        caddr            [50] (IP address of Alice),
        kippu-length     [51] 128,
        kippu-data       [52] F3t?9#ED@3q34GRx5...    -- TGS session key
                                                      encrypted using the PK
                                                      of the Visited KDC

}

EncASRepPart     ::= SEQUENCE {

        key              [60] (6546756286567), -- TGS session key
        last-req         [61] (020220061532),
        nonce            [62] (5082342423),    -- Same as in AS-REQ issued by Alice
        key-expiration   [63] (NA),
        flags            [64] (0),
        authtime         [65] (030220061454),
        starttime        [66] (030220061454),
        endtime          [67] (030220061954),
        renew-till       [68] (NA),
        srealm           [69] (MIT),           -- The visited realm name
        sname            [610] (krbtgt),
        caddr            [611] (IP address of Alice),

}
```

5.3 The TGSP protocol

In Section 5.2, we specified the ASP protocol, the first component of the XKDCP protocol. We explained how ASP is used for providing a TGT to roaming users. In this chapter, we present the second part of XKDCP protocol. The TGSP protocol offers a cross-realm authentication model for the remote service access scenario (see Section 2.2.1). The TGSP protocol, allows a user located in a certain realm (can be the home realm or a visited realm) to obtain service tickets for services offered in remote realms. The characteristic feature of the TGSP protocol is that the client does not need to have a cross-realm TGT for the remote realm and does not need to contact the remote KDC at all. In fact, from the user point of view, the local KDC delivers the service ticket as if the service was registered as a principal in the local realm. The cross-realm operations are managed by the local KDC and made thus transparent to the client. The local KDC uses the TGSP protocol with the KDC of the realm where the service is registered. The result of the TGSP exchange is that the local KDC have enough materials to deliver a service ticket to the client.

Figure 5.2 illustrates the TGSP operations involved in the process of delivering service ticket for a local user. The client located in his home realm, issues a TGS-REQ message to the local KDC (1). In the normal Kerberos operations, the KDC@H would send a referral to the client, however, in TGSP, KDC@H will try to build the service ticket and deliver it to the client. As a result of the TGSP exchange (2)(3), the elements needed to build the requested service ticket are gathered by KDC@H. It then builds the TGS-REP message that contains the credentials requested by the client (4). The user then can authenticate with the remote service (5).

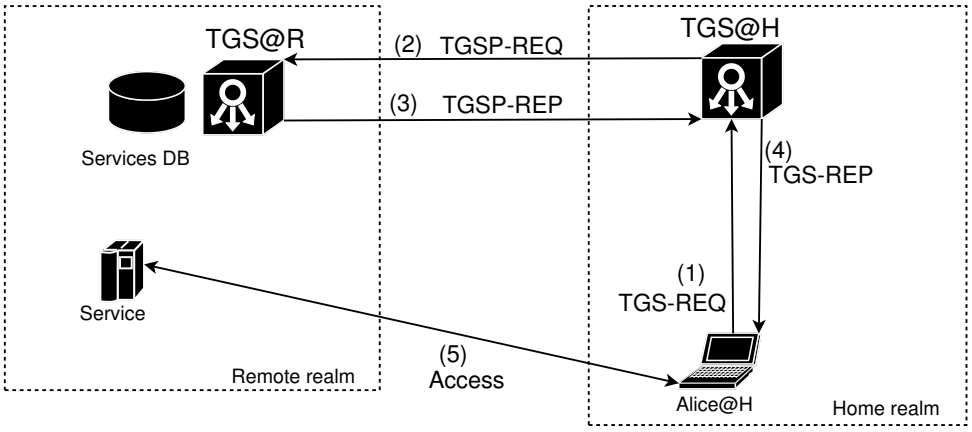


Figure 5.2: TGSP: The Inter Ticket Granting Service Protocol

In the following sections, we will describe the operations of the TGSP protocol in details and explain how KDC@H could build the TGS-REP containing the service ticket requested by the user.

5.3.1 The TGSP protocol operations

The TGSP protocol specifies a method that allows a Ticket Granting Service to build a service ticket even though the service's secret key is not known to the KDC. In the coming sections, we describe in details the operations involving the TGSP protocol. We assume that the client have already obtained a ticket granting ticket in the local realm (which can be the user's home

Listing 5.9: The TGS-REQ message

```
TGS-REQ          ::= SEQUENCE {

        pvno             [1] INTEGER (5) ,
        msg-type         [2] INTEGER (12),
        padata           [3] SEQUENCE OF PA-DATA OPTIONAL
        req-body         [4] TGS-REQ-BODY
}
TGS-REQ-BODY     ::= SEQUENCE {

        kdc-options              [40] KDCOptions,
        cname                    [41] PrincipalName OPTIONAL,
        realm                    [42] Realm                    -- Server's realm
        sname                    [43] PrincipalName OPTIONAL, -- Server name
        from                     [44] KerberosTime OPTIONAL,
        till                     [45] KerberosTime,
        rtime                    [46] KerberosTime OPTIONAL,
        nonce                    [47] UInt32,
        etype                    [48] SEQUENCE OF Int32 ,
        addresses                [49] HostAddresses OPTIONAL,
        enc-authorization-data   [410] EncryptedData OPTIONAL,
        additional-tickets       [411] SEQUENCE OF Ticket OPTIONAL

}
```

realm or a visited realm, in the latter case, the TGT was obtained using the ASP protocol as described in Section 5.2).

The client requests service ticket from the local TGS

In order to obtain a service ticket for a service, the client issues a TGS-REQ to the local KDC. Before issuing the TGS-REQ, the client needs to be sure that he have a TGT for the local realm. If the TGT is not in the ticket cache, then the client must do KDC discovery, then, issue an ASP-REQ to obtain a TGT for the local realm, as described in Section 5.2.1.

The TGS-REQ here is issued by the client as if the service is deployed by the local realm. Recall that, in the standard specification, the Kerberos client must ask for a cross-realm TGT for the remote realm. The TGSP protocol makes the client completely unaware of the cross-realm operations.

The TGS request issued by the client is filled as shown in Listing 5.9. The TGSP protocol mandates that the client puts the service name in the field “sname”. The “srealm”, must contain the realm name where the service is deployed. The difference with the standard specification is that the standard Kerberos protocol mandates that the client put the “krbtgt” principal name in the “sname” field when it notice that the realm name of the service is different from the client’s home realm name (meaning that the client is requesting a cross-realm TGT for the KDC of the remote realm).

The local KDC issues a TGSP request

When the local KDC receives a TGS-REQ message, it first checks the validity of the client’s TGT and the validity of the authenticator. The TGT could be obtained trough the ASP protocol for roaming users, or following the standard if the client belongs to the local realm. The KDC performs the verification of the credentials as defined in [7]. The processing of TGS requests does not make any difference between roaming and local users. This is because the issuer of the TGT is always the local KDC and the TGT was encrypted using the local TGS’s secret key.

Listing 5.10: The TGS-REP message

```
TGS-REP          ::= SEQUENCE {
        pvno             [0] INTEGER (5),
        msg-type         [1] INTEGER (13),
        padata           [2] SEQUENCE OF PA-DATA OPTIONAL
        crealm           [3] Realm,
        cname            [4] PrincipalName,
        ticket           [5] Ticket,          -- encrypted using the service key
        enc-part         [6] EncTGSRepPart    -- encrypted using the TGS session key
}

EncTGSRepPart    ::= SEQUENCE {
        key              [60] EncryptionKey, -- service session key
        last-req         [61] LastReq,
        nonce            [62] UInt32,
        key-expiration   [63] KerberosTime OPTIONAL,
        flags            [64] TicketFlags,
        authtime         [65] KerberosTime,
        starttime        [66] KerberosTime OPTIONAL,
        endtime          [67] KerberosTime,
        renew-till       [68] KerberosTime OPTIONAL,
        srealm           [69] Realm,
        sname            [610] PrincipalName,
        caddr            [611] HostAddresses OPTIONAL
}
```

If the local KDC relies only on local resources, it can not can not build all the components of the TGS-REP message and will not be able to provide the credentials requested by the client. In deed, the TGS-REP message contains a part that must be encrypted using the service's secret key. This part consists on the service ticket (field [5] "ticket" in the Listing 5.10.)

The TGSP protocol enters in action at this stage to allow the local KDC to build a service ticket in collaboration with the remote KDC.

The other important part of the TGS-REP message is the "EncTGSRepPart" structure. This part must be encrypted using the TGS session key shared between the client and the TGS. The "EncTGSRepPart" must contain the service session key (the same as the key embedded in the service ticket). For this reason, the local KDC needs to obtain not only the service ticket, containing the correct information, but also, it needs the service session key that goes with it.

The goal of the TGSP protocol thus is to deliver a service ticket and the service session key embedded in it to the local TGS.

The local TGS issues a TGSP-REQ message to the remote KDC. The TGSP-REQ message contains all the informations necessary for building the ticket (see Listing 5.3 for details about the ticket contents). The information required to build the TGSP-REQ message are obtained by the local KDC from the client's TGS-REQ message. After sending the TGSP-REQ message, the local KDC expects the remote KDC to provide the service ticket for the requested service and the service session key that goes with the service ticket.

The service session key is a secret key that is included in both the service ticket and the "EncTGSRepPart" of the TGS-REP message. It must not be revealed to anyone except to the KDCs, and to the user. For this reason, we use a public key based encryption mechanism to protect this information.

The public key mechanism is used also to perform mutual authentication between the local KDC and the remote KDC. The local KDC, since it is the initiator of the ASP exchange, includes his public key and a signature in the TGSP-REQ message. The remote KDC verifies the identity of the local KDC by checking the signature and the authenticity of the public

key. If the signature is verified, the remote KDC generates a service session key and a service ticket (including the same session key). It then issues a TGSP-REP message containing the EncASRepPart and the Session Key encrypted using the public key of the local KDC. More details on the operations performed by the remote KDC on reception of an TGSP-REQ message are detailed in the next section.

The remote KDC receives the TGSP request

When the remote KDC receives a TGSP-REQ from the local KDC, it performs the following operations :

- The public key of the local KDC is verified, then the signature is authenticated. If these operations succeed the local KDC is trusted. If the authentication fails the TGSP-REQ message is simply dropped.
- Extract the informations about the service. The realm name of the requested service realm must correspond to the realm name of the remote KDC. If the realm name does not match, an error is reported to the local KDC. The remote KDC then look-up the principals database to extract the secret key that corresponds to the requested service. If the principal name of the service is not found in the database, the remote KDC returns an error to the local KDC.
- A random session key is generated, this will be the *service session Key* (field [60] "EncryptionKey" in Listing 5.10 and field [531] in Listing 5.3).
- The remote KDC builds a service ticket as defined by [7]. The ticket is filled using the information from the TGSP-REQ message. The newly generated session key is included in the "EncTicketPart" structure of the ticket (see Listing 5.3). The "EncTicketPart" is encrypted using the service secret key.
- The service session key is encrypted using the public key of the local KDC.
- The TGSP-REP message containing the service ticket and the encrypted service session key, is sent back to the local KDC.

The TGSP-REP message contains the public key of the remote KDC and a signature. These informations will allow the local KDC to trust the TGSP-REP message after authenticating the remote KDC.

The local KDC receives the TGSP reply

When the local KDC receives the TGSP-REP, it performs the following operations :

- The public key of the remote KDC is verified then the signature is authenticated. If these operations succeed the remote KDC is trusted. If the authentication fails the TGSP-REP message is simply dropped and the user is informed about the failure of the authentication.
- The service session key is extracted from the TGSP-REP message, the local KDC uses his public key for decrypting the service session key.

- The local KDC creates a "EncTGSRepPart" structure (see Listing 5.10). The service session key obtained from the previous step is placed in the field "EncryptionKey" .The "EncTGSRepPart" structure is then encrypted using the *TGS session key* extracted from the TGT provided by the user in the TGS-REQ message.
- The TGS-REP message is now ready, the local KDC put the service ticket received from the remote KDC, along with the "EncTGSRepPart" in a TGS-REP message and send it to the user.

The client receives the TGS reply message

When the client receives a TGS-REP message, it processes it as specified in RFC 4120. The service ticket is extracted from the message, then the "EncTGSRepPart" is decrypted using the *TGS session key* shared with the local KDC. At this stage, the client can issue an AP-REQ message to the application service.

5.3.2 TGSP protocol message specification

The specifications for the TGSP protocol message formats and the implementation guidelines follow the same approach as for the ASP protocol (see Section 5.2.2). The TGSP protocol is an extension of the standard Kerberos protocol and is implemented using the typed holes in the Kerberos messages.

The TGSP-REQ message

The TGSP-REQ message is derived from the TGS-REQ message (shown in Listing 5.9). It is built by the local KDC as follows :

- The local KDC creates a copy of the TGS-REQ received from the client.
- "msg-type" (field [2] of the TGS-REQ message) is set to the value 662. The msg-type 662 means that the message is a TGSP-REQ message and must be processed as such, using the TGSP protocol specifications.
- The padata (field [3] of the TGS-REQ message) is filled with a PA-DATA structure as shown in Listing 5.11. The contents and the format of the PA-DATA are the same as the PA-DATA of the ASP-REQ message defined in Section 5.2.2.
  - The padata-type is set to 111 (XKDCP-AUTH-DATA)[5].
  - The LRealm field contains the realm name of the local KDC.
  - The CRealm field contains the realm name of the client. This information is extracted from the field "crealm" in the EncTicketPart of the TGT.
  - The SubjectName is the name associated to the X.509 public key of the local KDC [36].
  - The X.509 public Key of the local KDC [36].
  - Signature is a digital signature computed using the local KDC's private key as specified in [37].

[5]This Pre-Authentication Data Types value is not used according the the RFC 4021

Listing 5.11: The TGSP-REQ message

```
TGSP-REQ          ::= SEQUENCE {

        pvno              [1] INTEGER (5) ,
        msg-type          [2] INTEGER (662),
        padata            [3] PA-DATA,
        req-body          [4] TGS-REQ-BODY

}
TGS-REQ-BODY      ::= SEQUENCE {

        kdc-options               [40] KDCOptions,
        cname                     [41] PrincipalName   -- Client name
        realm                     [42] Realm           -- Server's realm
        sname                     [43] PrincipalName , -- Server name
        from                      [44] KerberosTime OPTIONAL,
        till                      [45] KerberosTime,
        rtime                     [46] KerberosTime OPTIONAL,
        nonce                     [47] UInt32,
        etype                     [48] SEQUENCE OF Int32 ,
        addresses                 [49] The client's address
        enc-authorization-data    [410] Not Used.
        additional-tickets        [411] Not Used.

}
PA-DATA         ::= SEQUENCE {

            padata-type      [1] Int32,        -- 111 XKDCP-AUTH-DATA
            padata-value     [2] XKDCP-AUTH-DATA

}

XKDCP-AUTH-DATA {

    kippu            [220] Kippu,  -- used in TGSP-REP and ASP-REP
    LRealm           [221] Realm,     -- the realm name of the local KDC
    CRealm           [221] Realm,     -- the realm name of the client
    SubjectName      [222] OctetString,
    PublicKeyValue   [223] PublicKey,
    Signature        [224] Signature

}
```

The TGSP-REP message

The TGSP-REP message generated by the remote KDC is an extended TGS-REP message that is built as follows :

- A random session key is generated, this will be the *service session Key.*
- An EncTicketPart is created and encrypted using the application service's key. The EncTicketPart contains the randomly generated session key. The "caddr" filed is filled using the "addresses" field from the TGS-REQ-BODY of the TGSP-REQ message. This address corresponds to the client's IP address. The "crealm" field which corresponds to the realm name of the client is filled using the field "CRealm" embedded in the PA-DATA structure of the TGSP-REQ message.
- The remote KDC builds a service ticket using the previously created EncTicketPart. The "realm" field is set to the realm name of the remote KDC.
- The remote KDC prepares a padata field of type XKDCP-AUTH-DATA. The padata field will contain the public Key and the signature of the remote KDC so that the local KDC can verify the authenticity of the remote KDC if it wishes.
- The service session key is encrypted using the Public Key of the local KDC and placed in the "kippu" structure of the "padata" field.
- The remote KDC creates a TGSP-REP message and fills all the field as specified above. The "EncASRepPart" is left empty. Finally the "msg-type" field is set to 663 indicating that the message is a TGSP message. The "crealm" field which corresponds to the realm name of the client is filled using the field "CRealm" embedded in the PA-DATA structure of the TGSP-REQ message.

Finally, the TGSP-REP message looks as shown in Listing 5.12.

5.4 Conclusion

In this chapter, we have specified the ASP and TGSP protocols. The ASP protocol allows a roaming client to obtain a TGT in a visited realm. The obtained TGT is usable with the visited KDC as if the client was actually registered in the visited realm. We have shown how the TGSP protocol can be used to deliver service tickets for remote services. The design of the XKDCP protocol is aimed to be smoothly integrated in the current Kerberos standard. For this reason, the recommended extension mechanism was adopted. In deed, the protocol-specific data is conveyed using the typed holes in the Kerberos messages. Furthermore, we specified two new application types, used to distinguish the XKDCP protocol messages from the normal Kerberos messages.

In the following paragraphs, we discuss the features of the XKDCP proposal and we demonstrate how our proposal resolves the issues described in Section 4.3.2.

Listing 5.12: The TGSP-REP message

```
TGSP-REP          ::= SEQUENCE {

        pvno              [0] INTEGER (5),
        msg-type          [1] INTEGER (663),
        padata            [2] PA-DATA,       -- Contains the service session key
        crealm            [3] Realm,
        cname             [4] PrincipalName,
        ticket            [5] Ticket,        -- The service ticket
        enc-part          [6] EncTGSRepPart  -- Empty

}
PA-DATA       ::= SEQUENCE {

            padata-type     [1] Int32,        -- 111 XKDCP-AUTH-DATA
            padata-value    [2] XKDCP-AUTH-DATA

}

XKDCP-AUTH-DATA {

    kippu             [20] Kippu,
    LRealm            [21] Realm, -- The local realm name
    CRealm            [21] Realm, -- The realm name of the client
    SubjectName       [22] OctetString,
    PublicKeyValue    [23] PublicKey,
    Signature         [24] Signature

}
Kippu    ::= SEQUENCE {

        kippu-length    [50] INTEGER ,
        caddr           [50] HostAddresses  -- IP address of the client
        kippu-data      [51] OctetString    -- service session key
                                            encrypted using the PK
                                            of the local KDC
}
Ticket          ::=  SEQUENCE {

            tkt-vno         [50] INTEGER (5),
            realm           [51] Realm,           -- The realm name of the service
            sname           [52] PrincipalName,   -- The service name
            enc-part        [53] EncTicketPart,

}

EncTicketPart    ::=  SEQUENCE {

            flags                   [530] TicketFlags,
            key                     [531] EncryptionKey, -- session key
            crealm                  [532] Realm,
            cname                   [533] PrincipalName,
            transited               [534] TransitedEncoding,
            authtime                [535] KerberosTime,
            starttime               [536] KerberosTime OPTIONAL,
            endtime                 [537] KerberosTime,
            renew-till              [538] KerberosTime OPTIONAL,
            caddr                   [539] HostAddresses OPTIONAL,
            authorization-data      [5310] AuthorizationData OPTIONAL

}
```

5.4.1 Clarifications

How roaming users obtain service tickets for remote services

Once the client have a TGT for the local realm (it does not matter if the client belongs to the local realm or if he/she is a roaming user). The local KDC can provide service tickets for this client to access services offered in the local realm as well as services offered in remote realms.

When the service is offered in a realm different than the local realm. Then, the KDC uses the TGSP protocol with the KDC of the remote realm. From the KDC's point of view, there is no difference between users who have obtained a TGT using the ASP protocol and those who belong to the local realm and have obtained a TGT without involving the ASP protocol. This is because the TGT is in both cases encrypted using the secret key of the local TGS principal ("krbtgt"). In case of roaming users, the information that the user does not belong to the local realm is kept in the TGT.

The XKDCP protocol is not involved when the client requests tickets for services offered by the local realm. In this case, the standard Kerberos protocol for intra-realm operations is used (even though the TGT was obtained using the ASP protocol).

Handling of ticket options and realm policies

Each KDC must have control on the ticket options in order to implement the policy of the realm. In XKDCP, since the EncTicketPart parts of tickets are created by the KDC of the service's realm. The ticket options always reflect the policy of the issuer's realm. These options are sealed using the secret key of the service. This secret key is only known to the service and its KDC. Therefore, the service can apply the realm policy as stated in the sealed ticket and it can be sure that the parameters conveyed in the encrypted part of the ticket were not tampered.

Is the protocol vulnerable to DoS attack? The public key operations performed by the Visited KDC are mainly for authenticating the Home KDC. However, this authentication is not necessary. In deed, if the home KDC is fake, then it obviously will not be able to generate the EncTicketPart since it does not know anything about the user. The Visited KDC thus should only perform the authentication of the Home KDC when the policy of the realm requires it. In the ASP protocol for example, the risk of DoS attach exists if an attacker located inside the realm claims that he belongs to a legitimate realm known to the visited KDC. The principal name provided by the attacker must exist in the home realm's database. In such case, the home KDC as well as the Visited KDC might be vulnerable to DoS attack since both will undertake cross-realm exchanges involving asymmetric cryptography operations.

The attacker must be physically located within the visited realm. This risk thus consist on an internal DoS attack, and it falls in the category of internal risks not in the category of Internet risks.

We assume that the protocol might be vulnerable for DoS attacks as described. However a deeper study is needed to confirm this impression. And probably simple solutions can be found to resolve this issue.

5.4.2 How the XKDCP proposal resolves the cross-realm issues

In this section, we describe our approach for addressing the issues relate to the cross-realm authentication in Kerberos (Section 4.3.2). In the following paragraphs, we demonstrate how

the XKDCP proposal resolves each of these issues.

Exposure of the KDC, resolved ?

The XKDCP proposal is an inter-KDC protocol, which mean that it only involves KDCs. When using the XKDCP protocol for handling cross-realm authentication, only KDCs communicate with other KDCs. Thus, The KDC can benefit from a safer policy by specifying filtering rules that only accept connections from a list of trusted KDCs. So far, the filtering rules had to allow connections from a while range of IP addresses that belongs to the trusted realms. We estimate thus, that the issue related to the exposure of the KDC is this resolved.

Cross-realm operations are client-centric, resolved ?

In order to obtain a TGT in a visited realm, roaming users have to perform an AS exchange with the visited KDC. In order to obtain tickets for services deployed by remote realms. The client must perform a TGS exchange with the local KDC. In both cases, the client only issues the minimal Kerberos exchange with the closest KDC. We estimate thus, that there is no load on the client device for obtaining Kerberos credentials. And the protocol is not client-centric.

Pre-authentication problem in roaming scenarios, resolved ?

The standard Kerberos protocol requires that the client contacts his home realm and thus requires that the client have Internet connectivity to be able to reach nodes beyond the boundaries of visited realm. When the ASP protocol is used, roaming users do not need to contact their home realm. All that is required is that they issue an AS-REQ message to the KDC of the visited realm. The Internet connectivity is thus not required by the client and the deadlock situation is avoided. The client can have a limited connectivity that only allows him to contact the local KDC.

Inter-realm trust chain resolved ?

When the XKDCP protocol is used, the KDCs does not need to share inter-realm keys. There is no notion of direct and indirect trust. The KDCs of the realms involved in the cross-realm authentication use public keys and an existing PKI to authenticate each others. For this reason, the trust chain is not needed and the scalability of the protocol when the realm number augment is assured. Also, if the realms hierarchy change, there is impact on the cross-realm operations.

Chapter 6

The Diameter-XKDCP application

In the following chapter, we give an overview of the design of the Diameter-XKDCP application which can be used by Kerberos KDCs to transport the inter-realm XKDCP messages.

6.1 Introduction

Diameter [9] is a AAA protocol (see Section 3.1.3). It offers message delivery across the Internet. Upper layer protocols can use the Diameter protocol as underlying framework that offers inter-realm routing and security. In order to use Diameter as underlying framework, a Diameter application must be designed. Diameter applications are extensions of the basic Diameter protocol that uses the AAA framework for the purpose of transporting messages of a specific protocol. As an example, Diameter-EAP [16] is used to transport EAP messages between realms. Both end points of the Diameter application must implement the Diameter application in order to process the encapsulated protocol correctly. The Diameter applications are build as independent application specific modules pluggable to the AAA server as explained in Section 3.1.2. The Diameter server routes the Diameter messages to the application modules based on the application ID field of the message. Each application module is responsible of parsing and processing the Diameter message that it receives. Diameter messages that are destinated to remote realms are routed to another AAA server or agent using the realm based routing table of the Diameter protocol.

In the next sections, we start by describing the process of building a Diameter application and its integration within an existing AAA framework. Then, we design the Diameter-XKDCP application which can be used by Kerberos KDCs to transport messages XKDCP messages between realms. At the end of this section, we discuss the advantages of using Diameter for the cross-realm operations and how does the Diameter-XKDCP application enhances the XKDCP proposal. The implementation specific details are tightly related to the WIDE Diameter implementation "WIDI" available from [38].

6.2 Creating a new Diameter application

In order to create a Diameter application, the designer must specify the following :

- The AVPs that will transport the protocol messages. If the AVP types provided by the base protocol are not suitable for a certain type of data. The application must specify a new type of AVP. Each specified AVP must have an associated code and type.
- The definition of the messages. The Diameter application will use a specific message format that is composed of a specific set of AVPs. Additionally, each message type have an identifier that is used by the application in order to determine know how to process the received message.
- Call back functions. The Diameter application must provide a function that will be passed to the Diameter API. This call back function will be used by the Diameter API each time a message having the corresponding application ID (The same application ID as the application that registered the call-back function) is received. The call back function must process the Diameter message and deliver it to the upper layer protocol.

The AVPs, and message specifications are added to the Diameter dictionary following the XML syntax specified by the Diameter implementation.

6.3 Overview of the Diameter-XKDCP application

The KDCs can use the Diameter AAA framework as a web of trust and routing. The Diameter API in deed connects several realms, it manages the routing of packets from any member realm to the other. The routing of the Diameter AAA framework is maintained at the Diameter API level using configuration files. In our proposal, we suggest the use of Diameter as an underlying layer for the XKDCP protocol. In deed, KDCs can be leveraged by delegating the burden of realm routing and inter-realm message security to the Diameter API. The Diameter-XKDCP application allows the KDCs to use the Diameter AAA framework for transporting ASP and TGS messages. For this purpose, the KDC initiates the Diameter API and registers a call back function that will accept the messages for the Diameter-XKDCP application. The Diameter API launches a Diameter AAA Server if none is up, in the host where the KDC is running. The KDC then can receive Diameter messages transporting ASP or TGSP protocol messages. This implies that the KDC and the Diameter AAA server are co-located and that the Diameter AAA server is in-fact an underlying protocol layer. In order to send XKDCP messages to other realms using the Diameter AAA framework, the KDCs must create a Diameter-XKDCP message that contains the requested AVPs. The application ID in the Diameter message must be set to the specific application ID assigned to the Diameter-XKDCP application. The message is then sent to the remote realm using the Diameter API. We can assimilate, thus the Diameter AAA framework as a replacement for the sockets facility provided by the operating system. The Diameter API of the initiator KDCs determines the next hop where the message will be sent, by looking at the realm routing table. The message can be forwarded to the target realm or can be forwarded to an intermediary AAA agent. When an agent receives a Diameter message it will forward it to the next hop determined using the realm routing table. When the message reaches the target realm (more precisely the AAA server of the remote realm where the target KDC is running). When the Diameter layer notices that the message is destinated to the local

Listing 6.1: Specifying a new AVP

```
<avp name="msg-type" code="667" mandatory="must">
  <type type-name="Unsigned32"/>
</avp>
```

realm, the call-back function registered by the KDC for the Diameter-XKDCP application will be called, and the Diameter-XKDCP message passed to the XKDCP module of the KDC. The role of the call-back function is to provide an interface for delivering messages from the Diameter AAA server to the upper layer protocols.

6.4 Implementation of Diameter-XKDCP application using WIDI

In this section, we specify the Diameter-XKDCP application described in the previous section. We undertook the design and implementation of the Diameter-XKDCP application using the WIDI Diameter implementation. The guidelines for creating a Diameter application (Section.6.2) are followed step by step in the following sections.

6.4.1 Adding new AVPs

New AVPs are added to the dictionary file "dictionary.xml" using the xml syntax similar as in Listing 6.1 The Table 6.2 shows the list of AVPs that are defined and used in the Diameter-XKDCP application.

Name	type	Code	Description
pvno	666	INTEGER	The Kerberos protocol version number. This AVP must be set to the value 5, indicating the Kerberos version 5.
msg-type	667	INTEGER	Indicates the message type. This field must be set to one of the following values (660/ASP-REQ, 661/ASP-REP, 662/TGSP-REQ, 663/TGSP-REP).
padata	668	OCTET STRING	The padata AVP corresponds contains an XKDCP-AUTH-DATA structure as specified by the XKDCP protocol.
req-body	669	OCTER STRING	This AVP corresponds to the request body of the TGSP-REQ or ASP-REQ message.
crealm	670	STRING	The realm name of the client, used in the TGSP-REP and ASP-REP.
cname	671	STRING	The principal name of the client, used in the TGSP-REP and ASP-REP.
ticket	672	OCTET STRING	Contains the service ticket in the TGSP-REP message.
enc-part	673	OCTET STRING	Contains the EncASRepPart of the ASP-REP message.

Table 6.2: List of AVPs defined by the Diameter-XKDCP application

Listing 6.2: The Diameter-XKDCP commands

```
<command name="Diameter-ASP" code="675" flags="PXY">
  </fixed>
          <avprule name="pvno" maximum="1" minimum="1"/>
          <avprule name="msg-type" maximum="1" minimum="1"/>
          <avprule name="padata" maximum="1" minimum="1"/>
  <fixed>
  <requestrules>
          <avprule name="req-body" maximum="1" minimum="1"/>
  </requestrules>
  <answerrules>
          <avprule name="cname" maximum="1" minimum="1"/>
          <avprule name="crealm" maximum="1" minimum="1"/>
          <avprule name="enc-part" maximum="1" minimum="1"/>
  </answerrules>
</command>

<command name="Diameter-TGSP" code="676" flags="PXY">
  </fixed>
          <avprule name="pvno" maximum="1" minimum="1"/>
          <avprule name="msg-type" maximum="1" minimum="1"/>
          <avprule name="padata" maximum="1" minimum="1"/>
  <fixed>
  <requestrules>
          <avprule name="req-body" maximum="1" minimum="1"/>
  </requestrules>
  <answerrules>
          <avprule name="cname" maximum="1" minimum="1"/>
          <avprule name="crealm" maximum="1" minimum="1"/>
          <avprule name="ticket" maximum="1" minimum="1"/>
  </answerrules>
</command>
```

6.4.2 Defining the Diameter-XKDCP messages

Each request and response message are defined as a single Diameter command. The commands are defined in an XML syntax in the dictionary file. The command syntax is composed of two elements : "requestrules" which specifies the list of AVPs that the request message must to contain. The second element of the command is the "answerrules" which specifies the AVPs of the reply message.

The Diameter-XKDCP application defines two commands : **Diameter-TGSP** and **Diameter-ASP**. Both of these commands are depicted in the Listing 6.2. Note that the commands use different codes, we assign the code number 675 to the Diameter-ASP command, and the number 676 to the Diameter-TGSP command.

6.4.3 Implementation of the call back function

The call-back function will be used by the Diameter API to deliver a Diameter-XKDCP message to the KDC. The function must have the following prototype :

void callback(void* ptr).

The function must then be registered as a call-back function for the Diameter-ASP and Diameter-TGSP messages. Each call-back function is registered for each command code. In the Diameter-XKDCP application, we have two commands (Diameter-TGSP and Diameter-ASP) thus, we need to register the function for both commands. This is performed using the function **"AAARegisterCommandCallback"** as shown in Listing 6.3.

Listing 6.3: Registering the call-back functions

```
AAARegisterCommandCallback ( 675, AAA_NO_VENDOR_ID,
                               "Diameter-ASP", 0, callback,
                               AAA_APP_INSTALL_FIRST);

AAARegisterCommandCallback ( 676, AAA_NO_VENDOR_ID,
                               "Diameter-TGSP", 0, callback,
                               AAA_APP_INSTALL_FIRST);
```

Chapter 7

Conclusion and future work

The Kerberos protocol was shown to be suitable for building AAA frameworks targeting client devices with low computational power. The cross-realm operations of Kerberos however, are still subject to improvements. We evaluated some of the major proposals like PKCROSS and IAKERB. Finally, we proposed a new extension to the Kerberos protocol. The XKDCP proposal, described in Chapters 5 and 6, is meant to resolve the issues related to the cross-realm authentication in Kerberos. In the next sections, we recapitulate the purpose of this work and we summarize the resulting output of this research. At the end, we outline some elements of our future work.

7.1 Purpose and objective

Emerging markets targeting personal/mobile computing devices In last couple of years, the advances in the next generation Internet, and the increasing popularity of network enabled personal devices, are real opportunities that offer a wide range of features and capabilities. New services mainly based on mobility, and various new applications for device control are now deployable at reasonable costs since the technology is ready for it.

Care must be taken regarding AAA operation overhead The AAA protocols are used to control the access to the network and to the application services. The Authentication and Authorization are vital elements for the deployment of any kind of service. The performance of the AAA framework is a major issue when considering devices with low computational power. Indeed, some of the cryptographic protocols are exceptionally demanding. Experimental results have shown that an RSA decryption operation on a 16bit CPU can take several dozens of seconds of processing [6]. In end-user applications and real time systems, this is an unacceptable disadvantage. Symmetric cryptography in the other hand is less expensive. An average of 1 to 5 seconds are needed by a low-end 8bit processor for decrypting a coded message using the 3DES algorithm [6].

Kerberos protocol is a suitable candidate The Kerberos protocol is a suitable authentication scheme for environments involving devices with low computational capabilities. This is due to the low processing power required in Kerberos operations. The most expensive operations in Kerberos are the decryption of the "EncASRepPart" or "EncTGSRepPart" of the AS and TGS reply messages. These operations consist on the decryption of a small message using the DES algorithm with a relatively short key (the user's password is generally an 8 character string).

The Kerberos protocol offers origin authentication, session key distribution and message integrity in open environments. All these features have made of Kerberos the industry standard in authentication frameworks.

Kerberos cross-realm operations are an issue The Kerberos protocol however, shows several issues when we consider it for use in a AAA framework for the mobility based services, as discussed in Section 4.3.2. These issues were targeted by several related works. We described the main extensions of the Kerberos protocol related to the cross-realm operations ; IAKERB and PKCROSS are independent proposals that target some aspect of the issues. The IAKERB allows the use of Kerberos in access networks, resolving the pre-authentication issue (Section 4.3.2). while the PKCROSS extension resolves the issue related to the scalability of the cross-realm operations. These extensions are presented in Section 4.4. However, even combined in a single framework, these proposals could not resolve all the identified issues.

7.2 Achievements

The objective of this thesis was to suggest an improvement for the cross-realm authentication protocol used in Kerberos. Some of the issues stated in Section 4.3.2 are well known facts about the Kerberos protocol. Other issues were identified after understanding the protocol and deep investigations. The issues that we stated are related mainly to the requirements of the service model and the client device specifications.

We undertook the design of a new cross-realm authentication model for Kerberos. Our proposal called XKDCP is composed of two sub-protocols. The ASP protocol used to deliver TGT for roaming users and the TGSP protocol used to provide service tickets for accessing remotely deployed services. Both of these protocols use the same Kerberos messages as specified in the standard. The typed holes are exploited to transport protocol specific data. New message types were defined. These message types tell the KDC that the message must be processed following the XKDCP specification. The XKDCP protocol was shown to resolve all the issues that we identified in this work (Section 5.4.2).

Although ASP and TGSP were designed as sub-protocols of the XKDCP protocol. Both of them can be used independently since they do not interact with each other and their operations are completely independent. In [39], we show and example of a business model using a single sign on framework based on the ASP protocol. In the example, we did not mention the TGSP at all, since only the functionalities of the ASP protocol were needed.

The second objective of this work was to investigate the common AAA framework models and the protocols used in these frameworks. The reason behind these investigations was to explore the possibilities of the interaction between Kerberos and the existing AAA protocols such as Diameter. As part of the investigations, we suggested in [40] a AAA framework for nested mobile environments. The proposed architecture combined the newly defined Diameter protocol as a back-end AAA protocol with PANA as front-end authentication mechanism deployed at

the network access servers. Lately, the main issues related to nested mobile environments and AAA frameworks were detailed and discussed in [41].

In an attempt to profit from the Diameter AAA protocol as underlying framework, we designed the Diameter-XKDCP application in Chapter 6. The specification is based on the WIDE [42] implementation of the Diameter protocol.

An experimental implementation of the XKDCP protocol was developed in Shinoda laboratory as a part of a generic AAA framework that goes with the name of "LOBA" [43] [44]. The LOBA framework consists on a simplified Kerberos implementation using the XKDCP protocol for cross-realm operations. Apart of the authentication system that enables clients to obtain Kerberos credentials, the LOBA framework provides access control framework that consists on a firewall controller service aimed to be deployed in network access servers. A firewall controller client integrating a Kerberos client was also developed. The client program is used by the user to perform authentication with the firewall controller service and configure the firewall of the access devices.

In collaboration with Tokyo university and members of the WIDE organization, we demonstrated [44] the readiness of the NEMO technology using the LOBA framework.[1]. The Nautilus6 [38] working group which belongs to the WIDE organization, had this demonstration of the AAA and NEMO mechanism to show cross-realm authentication and network access authorization of NEMO mobile routers. The demonstration was successfully performed during the UNS-2005 session. We have shown to the public that research is being done for easing the deployment of NEMO platforms, giving examples such as the NEMO-Bus and other mobile network scenarios using NEMO.

7.3 Future work

We estimate that the specification of the XKDCP protocol is subject to further improvements. We intend to create a detailed specification document and present it to the IETF for discussion and improvements. A standardization effort will be undertaken by proposing our work to the Kerberos working group.

The performance of our proposal has not been evaluated quantitatively. Indeed, our arguments are based on common sense reasoning, and numerical values would be a better convincing argument. For this reason, we intend in the future work to measure the performance of the XKDCP protocol and compare it to the other proposals. We plan to use simulation and/or mathematical analysis for his purpose. Some related effort [45] has been already done to modalize the Kerberos protocol in view of its analysis.

Future work will also include the re-implementation of the XKDCP protocol as specified in this document. The actual implementation used in LOBA is a stand-alone implementation and just a proof of concept. It does not follow all the specification details of this document and of RFC 4120. Therefore, it is not inter-operable yet with other implementations of the Kerberos protocol.

[1]The demonstration was performed in the occasion of The Ubiquitous Network Symposium (UNS) which is an exhibition and conference of technologies which enable ubiquitous access to the Internet, ubiquitous terminals and ubiquitous usage of various contents

Acknowledgments

I would like to express my sincere gratitude to all those who supported me and gave me the opportunity to complete this thesis.

First, I thank the Japanese government for awarding me the honorable Monbukagakusho scholarship and all their financial support.

I would like to express my thanks to Professor Mohammed ben Ahmed and Professor Takuya Katayama who helped me to start my studies in JAIST, and offered me valuable guidance that allowed me to follow my research orientations.

All my gratitude to my supervisor Professor Yoichi Shinoda for his continuous support and valuable directions, as well as for introducing me to the WIDE organization.

Special thanks to Thierry Ernst for providing me constant encouragements, and the Nautilus6 AAA team for their comments and discussions that helped me a lot in my research.

Bibliography

[1] D. Johnson, C. Perkins, and J. Arkko, "Mobility Support in IPv6," RFC 3775 (Proposed Standard), June 2004.

[2] V. Devarapalli, R. Wakikawa, A. Petrescu, and P. Thubert, "Network Mobility (NEMO) Basic Support Protocol," RFC 3963 (Proposed Standard), Jan. 2005.

[3] "Online dictionary," As of Jan 2006, http://reference.com.

[4] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120, Feb. 1978, The basics of trap-door functions and the famous RSA public key cryptosystem are presented in this paper.

[5] "Reneas technology corporation," Web page, As of Feb 2005, http://www.renesas.com.

[6] Okabe Nobuo, Sakane Shoichi, Miyazawa Kazunori, Inoue Atsuhi, Masahiro Ishiyama, and Kenichi Kamada, "Security Architecture for Control Networks using IPsec and KINK," Tech. Rep., SAINT, Feb 2005.

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

[8] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," RFC 2865 (Draft Standard), June 2000, Updated by RFCs 2868, 3575.

[9] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko, "Diameter Base Protocol," RFC 3588 (Proposed Standard), Sept. 2003.

[10] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "Extensible Authentication Protocol (EAP)," RFC 3748 (Proposed Standard), June 2004.

[11] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence, "Generic AAA Architecture," Request For Comments 2903, IETF, August 2000.

[12] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence, "AAA Authorization Framework," RFC 2904 (Informational), Aug. 2000.

[13] D. Forsberg, Y. Ohba, B. Patil, H. Tschofenig, and A. Yegin, "Protocol for Carrying Authentication for Network Access," Internet draft, IETF, January 2005, Work in progress.

[14] N. Haller, "The S/KEY One-Time Password System," RFC 1760 (Informational), Feb. 1995.

[15] Y. El Mghazli, Y. Ohba, and J. Bournelle, "PANA: SNMP usage for PAA-2-EP interface," Internet draft, IETF, October 2004, Work in progress.

[16] P. Eronen, T. Hiller, and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application," RFC 4072 (Proposed Standard), Aug. 2005.

[17] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *CACM*, vol. 21, no. 12, Dec. 1978.

[18] Mihir Bellare and Phillip Rogaway, "Provably secure session key distribution: the three party case," in *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, New York, NY, USA, 1995, pp. 57–66, ACM Press.

[19] L. Zhu and K. Jaganathan, "Generating kdc referrals to locate kerberos realms," Internet draft, IETF, July 2005, Expired January 2006.

[20] "The MIT Kerberos implementation," As of Feb 2006, http://web.mit.edu/kerberos/www/.

[21] J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510 (Proposed Standard), Sept. 1993, Obsoleted by RFC 4120.

[22] Jonathan T. Trostle, Irina Kosinovsky, and Michael M. Swift, "Implementation of cross-realm referral handling in the MIT kerberos client," in *NDSS*, 2001.

[23] National Institute of Standards and Technology, "Data encryption standard," FIPS Publication 46-2, Dec. 1993.

[24] M. Wahl, T. Howes, and S. Kille, "Lightweight Directory Access Protocol (v3)," RFC 2251 (Proposed Standard), Dec. 1997, Updated by RFCs 3377, 3771.

[25] R. Rivest, "The MD5 Message-Digest Algorithm ," RFC 1321 (Informational), Apr. 1992.

[26] S. Sakane, K. Kamada, M. Thomas, and J. Vilhuber, "Kerberized Internet Negotiation of Keys (kink)," Internet draft, IETF, July 2005, Work in progress.

[27] S. Kent and R. Atkinson, "IP Authentication Header," RFC 2402 (Proposed Standard), Nov. 1998, Obsoleted by RFC 4302.

[28] L. Zhu and B. Tung, "Public key cryptography for initial authentication in kerberos," Internet draft, IETF, January 2006, Work in progress.

[29] Matthew Hur, Brian Tung, Clifford Neuman, Ari Medvinsky, Gene Tsudik, and Bill Sommerfeld, "Public key cryptography for cross-realm authentication in kerberos," Internet draft, IETF, November 2001, Expired May 2001.

[30] Jonathan Trostle, Michael Swift, Bernard Aboba, and Glen Zorn, "Initial and pass through authentication using kerberos v5 and the gss-api," Internet draft, IETF, October 2002, Expired March 2003.

[31] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–652, Nov. 1976.

[32] E. Rescorla, "Diffie-Hellman Key Agreement Method," RFC 2631 (Proposed Standard), June 1999.

[33] Alan Harbitter and Daniel A. Menascé, "The performance of public key-enabled kerberos authentication in mobile computing applications," in *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, New York, NY, USA, 2001, pp. 78–85, ACM Press.

[34] S. M. Bellovin and M. Merritt, "Limitations of the kerberos authentication system," *SIGCOMM Comput. Commun. Rev.*, vol. 20, no. 5, pp. 119–132, 1990.

[35] A. Gulbrandsen, P. Vixie, and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)," RFC 2782 (Proposed Standard), Feb. 2000.

[36] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 3280 (Proposed Standard), Apr. 2002, Updated by RFC 4325.

[37] R. Housley, "Cryptographic Message Syntax (CMS)," RFC 3852 (Proposed Standard), July 2004.

[38] "WIDE Nautilus6 Working Group Web Page," Web page, As of Feb 2006, http://www.nautilus6.org.

[39] S. Zrelli and Y. Shinoda, "Single sign-on framework for aaa operations within commercial mobile networks," *The First International Conference on Availability, Reliability and Security*, Apr. 2006.

[40] S. Zrelli, T. Ernst, J. Bournelle, G. Valadon, and D. Binet, "Access control architecture for nested mobile environments in ipv6," *SAR05 4th Conference on Security and Network Architectures*, pp. 75–85, June 2005.

[41] J. Bournelle, G. Valadon, D. Binet, S. Zrelli, M. Laurent-Maknavicius, and JM. Combes., "Aaa considerations within several nemo deployment scenarios," *WONEMO 2006, The First International Workshop on Network Mobility*, Dec. 2005.

[42] "Widely Integrated Distributed Environment, project," Web page, As of Feb 2006, http://www.wide.ad.jp.

[43] "LOBA: A lightweigth kerberos-based aaa framework," Web page, As of Feb 2005, http://www.jaist.ac.jp/ zrelli/index.php?LOBA.

[44] Thierry Ernst and Keiichi Shima, "Nautilus6 project activity report in 2005," Report, WIDE, January 2006.

[45] Alan Harbitter and Daniel A. Menascé, "A methodology for analyzing the performance of authentication protocols," *ACM Trans. Inf. Syst. Secur.*, vol. 5, no. 4, pp. 458–491, 2002.