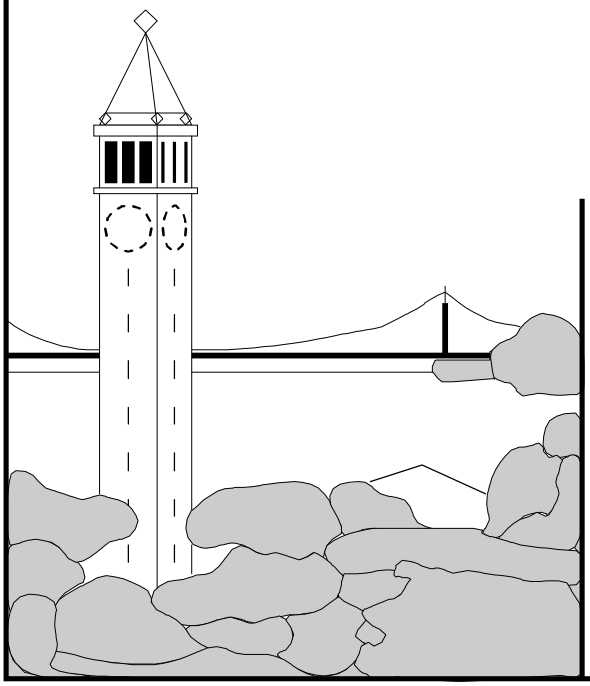
Secure Authentication System for Public WLAN Roaming

Ana Sanz Merino
asanz@eecs.berkeley.edu



Report No. UCB//CSD-05-1398
May 2005

Computer Science Division (EECS)
University of California
Berkeley, California 94720

This technical report is supported by grant number EIA-0122599

Secure Authentication System for Public WLAN Roaming

by

Ana Sanz Merino

ABSTRACT

Secure Authentication System for Public WLAN Roaming

by

Ana Sanz Merino

Master of Science in Computer Science

University of California at Berkeley

Professor Randy H. Katz, Research Advisor

A serious challenge for seamless roaming between independent wireless LANs (WLANs) is how best to confederate the various WLAN service providers, each having different trust relationships with individuals and each supporting their own authentication schemes, which may vary from one provider to the next. We have designed and implemented a comprehensive single sign-on (SSO) authentication architecture that confederates WLAN service providers through trusted identity providers. Users may automatically select the appropriate SSO authentication scheme from the authentication capabilities announced by the WLAN service provider, and can block the exposure of their sensitive information while roaming. In addition, we have developed a compound Layer 2 and Web authentication scheme that ensures cryptographically protected access while preserving pre-existing public WLAN payment models. Our experimental results, obtained from our prototype system, show that the total authentication delay is below 2 seconds in the worst case. This time is dominated primarily by our use of industry-standard XML-based protocols, yet is still small enough for practical use.

Contents

List of Figures

List of Tables

1 Introduction

Low deployment costs and high demand for wireless access have led to rapid deployments of public WLAN hotspot services by many providers, including startups and telecom operators [1]. Most service providers cannot cost-effectively deploy as many access points as needed to achieve good wide-area coverage, and thus supporting inter-operator roaming is a natural strategy for enlarging their service area. In such a roaming model, users may connect to the Internet via access points owned by providers that are unknown to them and for whom a trust relationship may not exist. Security mechanisms that protect both the user and the network are required. The roaming architecture works well in the cellular phone network because of its standard methods for determining user and service provider identity, as well as for service accounting and settlement. Such a standardized architecture for authentication, authorization, and accounting, agreed to by a larger and more heterogeneous set of service providers, simply does not exist nowadays for public WLANs. Even with mobility extensions to the Internet's routing protocols, the lack of such an architecture makes it difficult to confederate WLAN service providers.

Our main challenge is to define security mechanisms that protect both the user and the network. This is not an easy task in a public WLAN environment. Security always involves a tradeoff between convenience and risk. For public WLANs, users require that authentication, authorization and charging information (such as user unique identifier and credit card information) be protected against imprudent exposure to providers unless explicitly permitted by the user. At the same time the user may want seamless roaming by avoiding manual sign-on if it does not violate the user's security policy. From the provider's viewpoint, strict network access control is necessary to prevent theft of service from malicious attackers. On the other hand, WLAN providers are normally interested in giving IP-level access to users before authentication to allow various authentication and authorization options, such as one-time credit card payment or to provide free local and advertisement content for non-subscribers. This strategy of giving IP-level

access without authentication yields a vulnerability to theft of service through IP or MAC address spoofing.

In light of these problems, we have developed a comprehensive security solution for public WLAN services, as an overlay on existing standard authentication and authorization models. We assume multiple underlying authentication methods; that is, we do not require a single method that is universally adopted. Our solution is designed to achieve three goals: 1) to confederate wireless LAN service providers with different authentication schemes and under different inter-provider and user-provider trust relationships, 2) to protect user authentication information from unwilling exposure while also minimizing the amount of user intervention during sign-on, and 3) to strictly control network access by cryptographic methods while supporting the existing alternative authorization methods currently used in deployed public WLANs.

To achieve the first goal, we use single sign-on (SSO) authentication technologies to confederate WLAN service providers via trusted identity providers. To accommodate the possible coexistence of multiple authentication methods, we have created an authentication flow adaptation framework. Its key component is our Authentication Negotiation Protocol. In a heterogeneous scenario like the one we pursue, it is tremendously valuable that users know beforehand the authentication methods supported by each particular server and offering users the possibility to choose between the available alternatives. It is fundamental to do it in a standard way to allow automatic processing of the information at the client side. For this purpose we have designed a new XML web-based protocol, the Authentication Negotiation Protocol, which allows servers to announce their authentication capabilities, as well as other relevant information for users, such as the available charging options, and also permits users to communicate their authentication choice to the server. We have prototyped both the server and the client for this protocol.

For the second goal, we have developed a client-side policy engine that automatically selects the authentication information to be sent to the service provider based on user-defined policies (written in XML) and considering the communication context. The policy engine can be used in conjunction with the Authentication Negotiation Protocol client, which can request the policy engine to determine the information to be sent to the Authentication Negotiation Protocol server based on the capabilities of the underlying authentication server, instead of asking the user to

provide that information manually. This way, the policy engine can be considered part of the authentication adaptation framework. But it can also be used independently, as it provides a generic API for access to authentication information. It can be invoked not only by web-based mechanisms, but also by other mechanisms like link-layer authentication. In addition, the policy engine supports alternative access control models, such as one that requires additional actions to be taken before authorization (a.k.a. provisional action) [5] [6].

As for the third goal, we have developed a compound Layer 2 (L2) and Web authentication scheme to ensure cryptographically protected access in public wireless LANs. In our compound scheme, the user first establishes a L2 session key by using a guest (anonymous) account in an IEEE 802.1X authentication. The user then embeds the L2 session key digest in web authentication. By binding the L2 and Web authentication results, our scheme prevents theft of service, eavesdropping, and message alteration in public WLANs.

To illustrate our use of authentication flow adaptation and the role of the policy engine, consider the following scenario:

"A user walks out of his office with his PDA, roaming into a campus-wide WLAN network. The network announces its identity and authentication scheme. This network uses a web-based authentication scheme, and is strongly trusted by the user. The policy engine allows the user to automatically submit his identity and credentials to the campus-wide authentication system, thus allowing him to get connected to the campus-wide WLAN network.

As he leaves the campus and strolls into a cafe, the PDA detects the presence of a public WLAN. As before, the WLAN network announces its identity and authentication capabilities. The policy engine finds both the RADIUS-based and the Liberty-based authentication methods are available on that network, and selects the Liberty-based authentication as per the user's preferences. The public WLAN uses time-based charging, and the user's policy file indicates that automatic roaming should not be performed in such a situation. The policy engine asks the user if he wants to connect to the public WLAN. If he acknowledges it, the policy engine submits his authentication information to the authentication server of the WLAN service provider that provides network access for the cafe. He then gets access to the Internet."

To verify and evaluate our proposed architecture, we have developed a federated WLAN testbed. Measurement results show that the additional delays originated by the authentication

adaptation process, the client-side policy engine and the compound L2 and web authentication process are 139 msec, 35 msec and 120 msec, respectively.

The rest of this report is organized as follows. Section 2 relates our design to previous works. Sections 3, 4, 5 and 6 detail our single sign-on authentication model, the authentication adaptation framework, the client-side policy engine, and the compound authentication scheme. Section 7 presents some implementation details. Section 8 describes the prototype system and our evaluation results. Section 9 points out areas for future work. Finally, Section 10 concludes the paper.

2 Related work

Link layer authentication

IEEE 802.1X standard [7] provides port-based access control based on the authentication results for devices interconnected by IEEE 802 LANs, and is considered a promising solution for securing corporate 802.11 networks. IEEE 802.11i [8], ratified as a standard in June 2004, provides robust security in 802.11 wireless LANs. Its authentication scheme is based on 802.1X. This employs the Extensible Authentication Protocol (EAP), with which any mechanism can be used for authenticating both the user and the network and establishing an L2 session key dynamically. Per-user encryption and message integrity check keys are derived from the L2 session key. Those keys protect the data packets sent over the air. Examples of authentication methods in the wireless LAN environment include EAP-TLS [9] and EAP-TTLS [10]. The former uses client and server certificates for mutual authentication. The latter also uses a certificate-based approach for server authentication, and allows the user to use various client authentication schemes (e.g., MS-CHAPv2 [19]). Although these methods work well in corporate WLAN environments, they exclude one-time credit-card authorization options and free advertisements as would be useful for public WLANs, because they assume a pre-shared secret between user and network.

Web-based authentication and network layer access control

Many public WLAN providers employ web-based authentication in conjunction with IP packet filtering at a network access server based on the MAC and IP addresses. Since address spoofing is easily accomplished using readily available tools, this method does not provide strong security for network providers. Malicious users can monitor the wireless channel, acquire MAC and IP addresses of authenticated users, and send packets with spoofed addresses to perform theft of service or DoS attacks. Such an attack can be prevented by deploying IPsec Authentication Headers [11] to check for packet integrity and to control access based on its result, but at a

substantial cost in bandwidth and computational overhead. To roam across different WLAN service providers, the authentication web server acts as a RADIUS client and forwards authentication messages to a RADIUS server in the user's home provider [4]. Although RADIUS-based roaming is increasingly being deployed, the user must show his identity and credentials to the WLAN service provider regardless of the level of trust relationship, due to HTTPS-RADIUS protocol conversion at the service provider. Authentication schemes based on Zero knowledge proofs [12] or Secure Remote Passwords [13] can help hide user credentials at service providers by using a one-way hash function and ephemeral public keys. However, they do not help protect the user identifier from the service provider.

Several WLAN providers deploy their own proprietary network access client, not for security reasons but rather for user convenience to select the closest access point. They use their own authentication protocols in the serving network, and the authentication message flow is fixed regardless of user preference.

Finally, the CHOICE network [14] makes use of Microsoft Passport as a web authentication database, and uses a proprietary security sublayer between the link layer and IP layer. In their scheme, as a result of web authentication, the network gives the user terminal a (key, token) pair for the purpose of network access control thereafter. Our work differs from theirs in that an individual user can choose its own identity provider and use only standard-based link layer security, while they assume a centralized authentication server and require proprietary security sublayer. Moreover, our scheme makes it possible for a user to select the most preferred authentication method and identity provider among others, and prevent unwilling security information exposure according to the user's policy, which are not mentioned in their paper. This is mainly because our policy management function resides both in the network and the client, while theirs assumed a policy controller only at the network side.

To summarize, existing web-based authentication does not provide sufficient security against the theft of service, dynamic selection of different authentication schemes, or a user's policy-based exposure of privacy information while roaming.

Interoperability of authentication schemes

Microsoft and Sun Microsystems have recently published two draft specifications to enable browser-based Web Single Sign-On between security domains that use Liberty Alliance and Microsoft's Web Services Security (WS-Security). The Web Single Sign-On Metadata Exchange Protocol addresses the problem of identity providers and web services supporting multiple and different authentication protocols. It defines a mechanism whereby web services can determine the protocol suites supported by the client's identity provider. This enables the web service to select the preferred option among the commonly supported protocols. The Web Single Sign-On Interoperability Profile defines a subset of interoperability mechanisms for Liberty Identity Federation and WS-Federation-based applications within the Web Single Sign-On Metadata Exchange Protocol framework.

Similarly to our Authentication Negotiation Protocol, the web single sign-on metadata exchange protocol uses XML messages to exchange authentication capabilities. As our authentication flow adaptation framework, Microsoft and Sun's joint proposal also considers that the user selects the desired identity provider. However, a crucial difference between our solutions is that their main focus is the flexibility offered to the web service, and we focus on the flexibility provided to the user. The goal of their protocol is to inform the web service about the different authentication alternatives, whereas in our protocol the target of that information is the user. Therefore, in their approach it is the web service that chooses the authentication method to use, while in our case it is the user.

3 Single sign-on confederation model

In this chapter we describe our confederation model and the single sign-on (SSO) and authentication technologies selected for our prototype. We use SSO authentication technologies to confederate WLAN service providers via trusted identity providers and we accommodate multiple underlying authentication methods. Since our goal was to build an overall architecture that integrates existing technologies while improving user security and control, we adopted current standard technologies rather than inventing new ones whenever possible.

In public WLANs, users are required to authenticate every time they roam into a different service provider's network. Before describing our model, we enumerate our assumptions.

- The user terminal can validate the certificates of the service provider's and identity provider's authentication servers.
- There are static trust relationships between the user and the identity provider, and between the service provider and the identity provider.
- The user can authenticate the service provider's authentication server via the identity provider's authentication server, and vice versa.

These assumptions are necessary for clients and service providers to authenticate each other to establish dynamic trust relationships.

Let's focus now on our confederation model. As depicted in Figure 1, we assume that there is at least one and possibly several identity providers with whom the WLAN service providers have roaming agreements, and with which the user has strong trust relationships. Roaming agreements between a service provider and an identity provider typically include supported authentication protocols, configuration of secure communication channel, and the method of charging/revenue settlements. Examples of such identity providers are ISPs, credit-card companies, roaming service providers (wireless LAN aggregators), cellular network operators, etc. Some identity providers are already doing business in the public wireless LAN roaming market. The presence of such identity providers exempts users from having multiple identities and credentials, and helps them

roam across WLAN service providers under different levels of trust. The user is really authenticated by the identity provider, and the service provider, based on its trust relationship with the identity provider, relies on the authentication result. In Figure1, the user can roam between the networks of WLAN Service Provider A and B because he has an account in Identity Provider 1 while both service providers have roaming agreements with Identity Provider 1. To roam into WLAN Service Provider C's network, the user should have an account at Identity Provider 2 because no other identity provider is available. Although the identity provider and the WLAN service provider are logically different entities, both may belong to a single administrative domain.

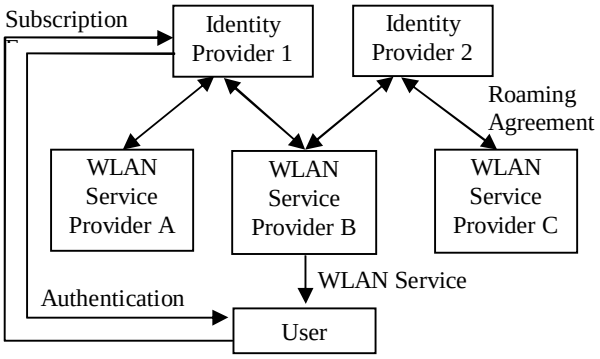


Figure 1: Roaming Model

We focus on developing a decentralized authentication framework for inter-service provider roaming where each provider adopts its own authentication methods, with arbitrary roaming agreement relationships between identity providers and WLAN service providers. Our architecture is independent of the authentication methods of identity or service providers, and allows users to choose their preferred identity provider and authentication scheme. In particular, we considered two industry-standard SSO authentication standards in our architecture: RADIUS [2] and Liberty Alliance Architecture [3]. These were selected because they represent methods that are currently widely deployed (RADIUS) [4] and other alternative schemes that are just now becoming available (Liberty). Besides, each of them uses a different communication approach with different security guarantees associated, allowing us to test our system under both conditions. Again, our architectural approach is not limited to these specific authentication standards; they

have been chosen to ensure that our architecture is flexible enough to handle multiple, realistic underlying methods.

RADIUS utilizes a proxy based authentication scheme; the user sends the authentication data to the service provider and the service provider forwards this to the user's identity provider. Figure 2 shows a simplified message sequence in RADIUS-based authentication. RADIUS' major weakness is that the user must expose its identity and credentials to the untrusted service provider, due to the HTTPS-RADIUS conversion process at service provider's web server. To prevent intermediate providers from learning the user's credentials, they may be end-to-end encrypted between user and identity provider. However, many identity providers do not support this option.

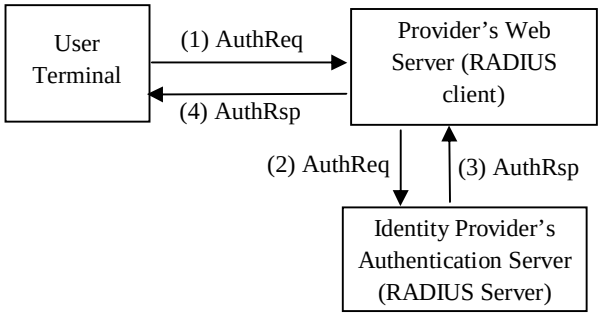


Figure 2: Proxy-based RADIUS Authentication

On the other hand, the Liberty Browser Artifact Profile makes use of a redirect-based authentication model (Figure 3). The user informs the service provider of the name of his identity provider, and then the service provider redirects the user to that identity provider. The user then sends the login information directly to the identity provider, receives its result, and forwards it to the service provider. The identity provider's result includes a pointer to an authentication assertion. Following the pointer, and making use of the SAML protocol [15], the service provider contacts the identity provider to obtain a secure confirmation about the authentication result. Although Liberty-based authentication is more complex than the RADIUS method, it has the advantage of hiding a user's identity and credentials from weakly-trusted WLAN service providers. To enable the Liberty flow, a hole must be opened in the service provider's firewall during authentication. This allows the direct communication between the user and the identity

provider across the service provider's network. The need to open the firewall is in fact a common characteristic to every redirect-based authentication mechanism.

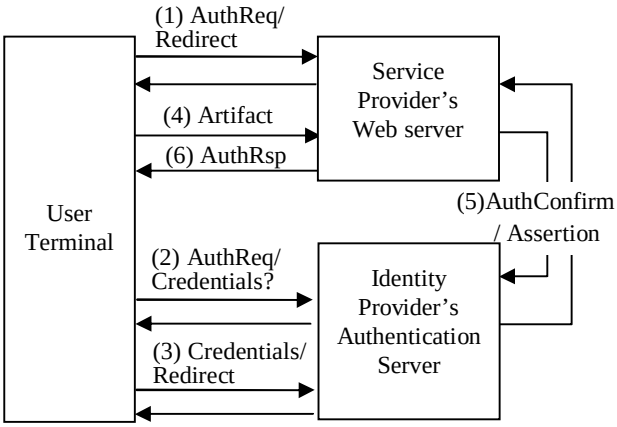


Figure 3: Redirect-based Liberty Browser Artifact Profile Authentication

4 Authentication flow adaptation framework

This section presents our authentication adaptation framework. Section 4.1 gives an overview of the framework components and data flow. Section 4.2 describes the server and client architecture. Finally, Section 4.3 presents the details of the framework key component, the Authentication Adaptation Protocol.

4.1 Overview

To integrate providers with different underlying authentication technologies, we have designed an architecture that can accommodate alternative authentication methods. An advantage of this architecture is that not all the confederated providers need support the same authentication scheme. As in traditional systems, a provider can only interact with others that use the same authentication technologies in our architecture. But our innovation is that our framework allows each provider to support more than one authentication scheme, permitting it to communicate with a larger number of providers. More importantly, in the case where a WLAN service provider supports multiple authentication options, users can select the method they prefer. User choice is particularly useful in a scenario where providers that maintain relationships with different trust levels with the user are confederated, as some authentication methods are more secure than others. A user can select one depending on their trust level with the service provider.

To allow users to choose, service providers must communicate their supported authentication methods. Even when the service provider only supports one method, it is useful that the user can determine which it is. This can affect his decision about whether to roam into the service provider network. Apart from the specific authentication technologies supported, the users need to know the information that the server requires for authenticating them. Thus they can determine if they are willing to provide this information. Users may also want to know the identity of the service

provider, and the companion information to verify it, to determine their level of trust for that provider. This almost certainly influences their decision. The charging schemes used by the server are also a determinant. Depending on the payment method, price, minimum charging period, and services granted, a user can decide whether using the provider services is of value and which of the available charging options he prefers. Finally, in our architecture there can be more than one identity provider. A user with accounts at several identity providers would like to know with which ones the service provider has roaming agreements. We call these different pieces of information relevant to the user at the moment of authentication *server authentication capabilities*. In our architecture, service providers must communicate these server authentication capabilities to users.

To facilitate roaming, it is useful if the communication of authentication capabilities and the user's selection are accomplished with minimal user intervention. The user should avoid having to make use of a web browser to exchange this information. We have designed a new XML web-based protocol, the *Authentication Negotiation Protocol*, which automates this process by allowing service providers to announce their capabilities to users and users to communicate their choices to service providers. This requires no web browser. Rather, we install in the user's machine a specific client, the *authentication negotiation client*. This is a "thin" software component that could be downloaded from the user's identity provider's web portal. The details of the protocol are defined in Section 4.3.

Note that users can still get authenticated whether or not the authentication negotiation functionality is installed on the server or the user's terminal utilizes the authentication negotiation client. In their absence, users are authenticated through a conventional web interface using manual methods. All that is lost is the automation of capability advertisement and user selection. Thus, legacy user terminals and service provider's authentication servers are readily supported.

The following figure shows the authentication flow adaptation sequence. As depicted in the diagram, in response to an authentication request from a new user wanting access to the external network, the authentication server presents him with the available alternatives and the information required for using each. The user then selects the appropriate authentication method manually or via the policy engine's processing of the user's pre-defined policy (described in the next section). The user then provides the information requested by the service provider for that particular

scheme. The service provider's authentication server, switching between alternative methods according to the user preferences, processes the information. Finally, the server communicates to the client the authentication result. When redirection of the user to another provider is required to complete the authentication, as in Liberty Alliance, an additional pair of messages is required. This is necessary so that the user communicates back to the service provider the redirection provider's authentication result and the service provider notifies the user about the final authentication decision. In this case, steps (7) to (10), marked in gray color, would also be needed.

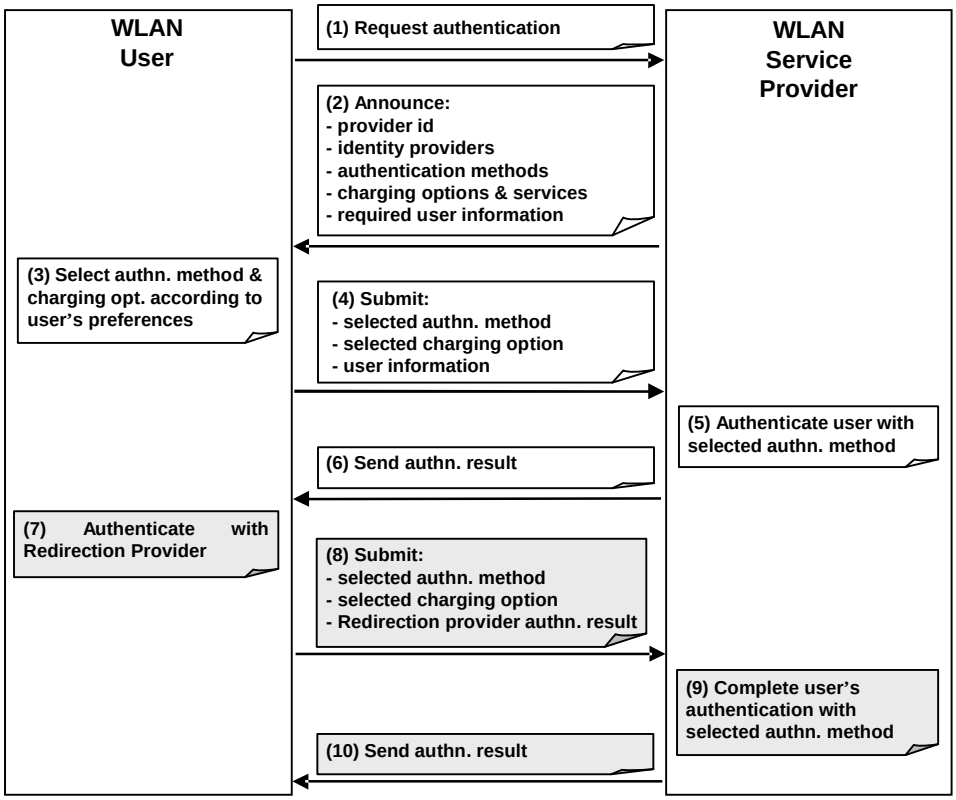


Figure 4: Authentication Flow Adaptation Sequence

4.2 Architecture

In this subsection we describe the general architecture of our authentication adaptation framework and its key functional components. On the client side, the authentication negotiation client is in charge of encoding and decoding the authentication negotiation messages, presenting

the authentication capabilities information to users through its graphical user interface and collecting the user's choice and information. Another component of our authentication adaptation framework located in the client side is the policy engine, which can be used independently of this framework. The authentication negotiation client is configured to use the policy engine rather than requesting the inputs from the user manually. In this case, the authentication negotiation client passes the decoded authentication capabilities information to the policy engine. The function of the policy engine as part of authentication adaptation is to automatically evaluate the authentication capabilities sent by the server, select the authentication method and charging option according to user-defined policies, and provide the user information requested by the server. This way, user intervention is not required, unless explicitly specified in the user's policies. The information to be sent to the service provider is returned by the policy engine to the authentication negotiation client, which takes care of encapsulating it in an authentication negotiation message and sending it to the server. The characteristics of the policy engine are discussed in Section 5.

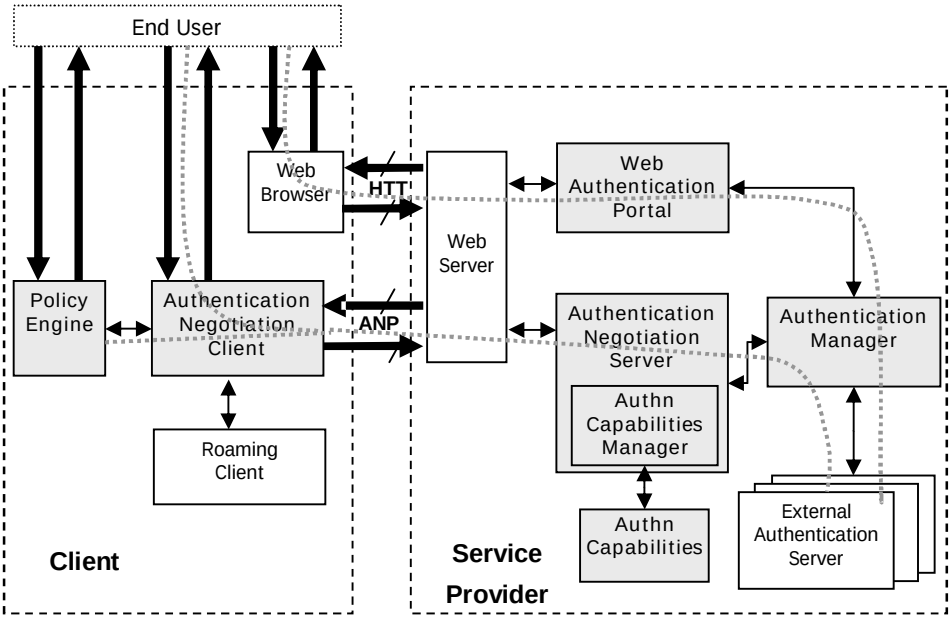


Figure 5: Authentication Flow Adaptation Architecture

On the server side, analogously to the client side, the authentication negotiation server is in charge of decoding and encoding the authentication negotiation messages. This module also processes the different protocol messages received. When requested for the server's authentication capabilities information, it retrieves it from the authentication capabilities manager. This is a subcomponent in charge of maintaining that information and having it updated and ready

for usage by other modules. For authentication requests, it extracts the user provided information and authentication choice. Once checked that all the required information is available, it passes them to the authentication manager to take care of the authentication. It then communicates back the result. The authentication manager knows the different authentication servers available in the service provider, and switches between them according to the user's choice.

The authentication negotiation server makes use of the service provider's web server for the exchange of HTTP messages with the client, that is, for the underlying transport functionality. A simple web application detects if a received message belongs to the Authentication Negotiation Protocol, in which case it is passed to the authentication negotiation server. If it is a plain HTTP message, then a typical web portal takes care of it. Clients that do not support the Authentication Negotiation Protocol can still be authenticated through the service provider standard web portal, which includes a web page for authentication in which the available authentication options are presented. The web portal, like the authentication negotiation server, makes use of the authentication manager to actually authenticate the user.

Figure 5 shows in gray the main components of our authentication flow adaptation architecture. Additional external components used by the authentication flow adaptation subsystem are showed in white. It can be observed that there can be more than one authentication server at the service provider, each corresponding to a different authentication technology. Two main flow sequences are possible, marked with dotted line in the figure. If the client does not have the authentication negotiation client installed, then the upper flow is followed, where the client's web browser and the server's web portal are the components involved. In this case the user needs to open the web browser and access the server's portal, the authentication web page will show him the available authentication alternatives, and the user will manually enter the information in the web form. If the user's terminal has an authentication negotiation client, this can be launch to automatically initiate the negotiation with the server. We provide as part of our system a roaming client component that, upon connection to a new WLAN network, automatically invokes the authentication negotiation client, avoiding user intervention. The lower flow will be followed in this case, with two possible variants: manual authentication by the user or, if the policy engine is present, automatic authentication. The policy engine, by its side, can decide under certain circumstances that user intervention is required.

4.3 Authentication Negotiation Protocol

The Authentication Negotiation Protocol is a very simple request-response protocol that consists of the exchange of four messages between client and server: authentication capabilities request/response, corresponding to steps (1) and (2) of the authentication flow adaptation sequence in Figure 4, and authentication request/response, corresponding to steps (4) and (6). An additional message is defined to support the case in which redirection to another provider is required, the authentication continuation request, corresponding to step (8). For step (10), no new message is required, since another authentication response message is used to communicate the final authentication result to the user. The content of all these messages is expressed in XML, to provide flexibility in its construction. An XML schema has been defined specifying the valid XML elements for the protocol and the requirements they must satisfy.

Part of the Authentication Negotiation Protocol functionality is user authentication, a task done in a different way by OASIS SAML protocol [15], an XML-based framework for exchanging security information. We have decided to keep our new protocol as close as possible to SAML, conceiving it as its extension. In the same way as SAML defines particular queries and statements for specific kinds of information to be exchanged, which are encapsulated inside general SAML request and response structures, we have defined the queries and statements needed for our protocol. While these can be used inside SAML requests and responses, we have defined our own request and response structures, which are close to SAML's. By doing so, we avoid some of the security overhead of SAML messaging not needed in our protocol. For example, the service provider does not need to authenticate every user that requests the authentication capabilities information, since this is not sensitive. The authentication negotiation requests and responses are extensible, and can accommodate SAML statements and queries if needed in the future.

In much the same way as SAML is used for message exchange outside of web-browsers, we use XML-encoded SOAP messages sent over HTTP/HTTPS for authentication negotiation messages not involving web browsers. To avoid unnecessary overhead, the Authentication Negotiation Protocol relies on the underlying protocols for message encryption. Authentication

Queries should always be carried over HTTPS to prevent users' credentials from traveling in clear text.

The queries and statements defined by the Authentication Negotiation Protocol are the following:

- Authentication Capabilities Query: used by clients to ask servers for their authentication capabilities. An optional nonce may be specified by the client, so that authenticity of the corresponding authentication capabilities response may be verified, as explained below.
- Authentication Capabilities Statement: used by servers to announce their authentication capabilities. This information is structured on the identity providers supported, so that processing is easier in the client. The identity providers are split into groups according to the authentication and charging options they support. For each group of identity providers, the authentication methods and charging options available are listed. This way, clients can look for their identity providers within the different identity providers groups and read the associated authentication and charging information. Each authentication method and charging option is assigned an identifier unique in the scope of the server, so they can be referenced in subsequent queries. A timestamp indicating the last time the server's authentication capabilities were updated is included, so clients can detect whether there have been modifications in the authentication capabilities if they decide to cache them. In case the client specified a nonce as part of the corresponding Authentication Capabilities Query, a copy of such nonce will be included in the Authentication Capabilities Statement, and the ANP Response will be digitally signed by the server. This way, the client can verify the authenticity, and integrity of the authentication capabilities information. The nonce provides protection against reply attacks. This level of protection based on nonces and signatures introduces overhead in the server and increases the authentication process latency. Therefore, it should be avoided when possible. It may not be necessary if HTTPS is used as underlying transport protocol for the exchange of authentication capabilities information. In that case, the client may attempt to verify the server's certificate and, if successful, be sure of its trustworthiness. The SSL session keys guarantee that the communication channel is secured and only that server can send the Authentication

Capabilities Statement over it, preventing the alteration of the statement by man in the middle attacks.

An example of a simple Authentication Capabilities Statement with one identity provider, two authentication methods and one charging option is shown in Figure 6. The Authentication Capabilities Statement is represented by the XML element *AuthnCapabilitiesStatement*. The timestamp indicating the last time that the server capabilities were modified is included as an attribute of this element. Following this, all the subelements that appear inside the *AuthnCapabilitiesStatement* element are described.

The identity of the service provider announcing the information and associated information to verify it are specified in the XML children element *Subject*, which is defined as part of the SAML protocol. Specifically, the name of the service provider, "my_service_provider_1" in our example, is indicated in the subelement *NameIdentifier*, and the subelement *SubjectConfirmation* contains the verification information. There are different methods to verify a server's identity. They are detailed in the documentation of the SAML protocol.

The element *IDPGroup* is used to specify each of the groups of identity providers supported and the authentication methods and charging options available for each of these groups. Inside it, the *IDPList* element contains the identities of the members of the group, each of them indicated with an *IDPName* element. The charging options supported are specified with *ChargingOptionIDReference* elements; and the authentication methods, with *AuthnMethodIDReference* elements. In this example only one identity provider is supported, "my_identity_provider_1". Therefore, there is only one *IDPGroup* with just one identity provider in it. There is only one charging option available for the group, "monthly_rate", and two authentication methods, "radius" and "liberty".

The user information required to authenticate using a particular authentication method is indicated within an *AuthnMethod* element. Each piece of required user information is specified with a *UserInfoDesignator* element, which has two attributes to uniquely identify that piece of information: the name of the user attribute that identifies it, and the namespace where that name is defined. The *AuthnMethod* element also contains an

A*uthnMethodID* element, which contains the identifier with which the authentication method must be referred within other elements, for example, within *IDPGroup* elements.

The characteristics of a specific charging option are detailed in a *ChargingOption* element. Inside this, the *ChargingOptionID* element contains the identifier with which the charging option must be referred within other elements. The *ChargingInterval* elements are used to describe the charging scheme. The charging characteristics can vary over time. As a result, it may be necessary to split their description into time intervals. Each *ChargingInterval* element has an attribute called *Order* to specify the position of that interval within the chain of all the intervals in which the description has been temporarily split. In our example the charging characteristics do not vary with time. Therefore, we have only one charging interval, and this is assigned *Order* 1 (first in the chain). Within a *ChargingInterval*, the *UnitPrice* element indicates the amount of money in USD that will be charged to the user for using the server's services during the amount of time specified in the *TimeUnit* element. The *TimeUnit* element contains a *Period* subelement to indicate the amount of time and a *Unit* attribute to indicate the units in which that amount of time is expressed (seconds, minutes, hours…). The *ChargingMode* element specifies the charging mechanism. Two of the defined values are 'Discontinued', which means that once passed the period of time indicated in *TimeUnit* the service is stopped, and 'Constant', which means that the user can use the service without stopping, being charged the *UnitPrice* for each *TimeUnit* passed. In our example, the user would be charged 0.10 USD per minute. Apart from these subelements, all shown in the example, the *ChargingInterval* element can contain additional elements to indicate, for example, an amount of money charged at the beginning of the interval, independently of how long the user makes use of the service provider's services. If specific user information is required to charge the user with this option, each piece of that information is specified with a *UserInfoDesignator* element, already described when detailing the *AuthnMethod* element. Finally, for each charging option, a *ServiceIDReference* element is used to indicate each of services offered to the user if he selects that charging option.

The description of a specific service is done with a *Service* element. This element includes a *ServiceID* subelement, which contains the identifier with which the service must

be referred within other elements, for example, within *ChargingOption* elements. And it includes a *ServiceDescription* subelement, which contains the description of the service.

```
<anp:AuthnCapabilitiesStatement LastUpdateInstant="2005-01-01T00:00:00Z">
        <saml:Subject>
                <saml:NameIdentifier>my_service_provider_1</saml:NameIdentifier>
                <saml:SubjectConfirmation>
                        <saml:ConfirmationMethod>…</saml:ConfirmationMethod>
                        <ds:KeyInfo>...</ds:KeyInfo>
                </saml:SubjectConfirmation>
        </saml:Subject>
        <anp:IDPGroup>
                <anp:IDPList>
                        <anp:IDPName>my_identity_provider_1</anp:IDPName>
                </anp:IDPList>
                <anp:ChargingOptionIDReference>monthly_rate</anp:ChargingOptionIDReference>
                <anp:AuthnMethodIDReference>radius</anp:AuthnMethodIDReference>
                <anp:AuthnMethodIDReference>liberty</anp:AuthnMethodIDReference>
        </anp:IDPGroup>
        <anp:AuthnMethod>
                <anp:AuthnMethodID>radius</anp:AuthnMethodID>
                <anp:UserInfoDesignator AttributeName="UserName" AttributeNameSpace="my_userinfo_namespace"/>
                <anp:UserInfoDesignator AttributeName="UserPassword" AttributeNameSpace="my_userinfo_namespace"/>
        </anp:AuthnMethod>
        <anp:AuthnMethod>
                <anp:AuthnMethodID>liberty<anp:AuthnMethodID>
                <anp:UserInfoDesignator AttributeName="IDPName" AttributeNameSpace="my_userinfo_namespace"/>
        </anp:AuthnMethod>
        <anp:ChargingOption>
                <anp:ChargingOptionID>monthly_rate</anp:ChargingOptionID>
                <anp:ChargingInterval Order="1">
                        <anp:UnitPrice>0.1</anp:UnitPrice>
                        <anp:TimeUnit Unit="Minute">
                                <anp:Period>1</anp:Period>
                        </anp:TimeUnit>
                        <anp:ChargingMode>Constant</anp:ChargingMode>
                </anp:ChargingInterval>
                <anp:UserInfoDesignator AttributeName="ContractNumber" AttributeNameSpace="my_userinfo_namespace"/>
                <anp:ServiceIDReference>private_contents</anp:ServiceIDReference>
        </anp:ChargingOption>
        <anp:Service>
                <anp:ServiceID>private_contents</anp:ServiceID>
                <anp:ServiceDescription> Access to private contents through the provider's web portal </anp:ServiceDescription>
        </anp:Service>
</anp:AuthnCapabilitiesStatement>
```

Figure 6: Authentication Capabilities Statement Example

- Authentication Query: used by users to request authentication. The identifiers of the authentication method and charging option selected must be indicated, and all the user information needed for those particular options must be also included.

  An example of an Authentication Query corresponding to the Authentication Capabilities Statement of Figure 6 is shown in Figure 7. The Authentication Query is

represented by the XML element *AuthnQuery*. The authentication method selected is indicated with the element *AuthnMethodIDReference*; the value of this element must be equal to the value of one of the *AuthnMethodID* elements of the Authentication Capabilities Statement. Analogously, the charging option chosen is indicated with the *ChargingOptionIDReference* element, and the value of this element must be equal to the value of one of the *ChargingOptionID* elements of the Authentication Capabilities Statement. In our example, we have selected "radius" as authentication method and "monthly_rate" as charging option. Each of the pieces of user information needed for the selected authentication and charging options is sent in a *UserInfo* element. The *UserInfo* element contains an *AttributeValue* subelement to include the information, and has an attribute to indicate the name of the user attribute that identifies that information.

```
<anp:AuthnQuery>
        <anp:AuthnMethodIDReference>radius</anp:AuthnMethodIDReference>
        <anp:ChargingOptionIDReference>monthly_rate</anp:ChargingOptionIDReference>
        <anp:UserInfo AttributeName="UserName">
                <saml:AttributeValue xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">my_user</saml:AttributeValue>
        </anp:UserInfo>
        <anp:UserInfo AttributeName="UserPassword">
                <saml:AttributeValue xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">my_password</saml:AttributeValue>
        </anp:UserInfo>
        <anp:UserInfo AttributeName="ContractNumber">
                <saml:AttributeValue
                xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">my_contract_number</saml:AttributeValue>
        </anp:UserInfo>
</anp:AuthnQuery>
```

Figure 7: Authentication Query Example

Another example of Authentication Query corresponding to the Authentication Capabilities Statement of Figure 6 is shown in Figure 8. In this case, "liberty" is selected as authentication method instead of "radius". Following instructions from the Authentication Capabilities Statement, the "IDPName" must be sent instead of the "UserName" and "UserPassword".

```
<anp:AuthnQuery>
        <anp:AuthnMethodIDReference>liberty</anp:AuthnMethodIDReference>
        <anp:ChargingOptionIDReference>monthly_rate</anp:ChargingOptionIDReference>
        <anp:UserInfo AttributeName="IDPName">
                <saml:AttributeValue
                xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">my_identity_provider_1</saml:AttributeValue>
        </anp:UserInfo>
        <anp:UserInfo AttributeName="ContractNumber">
                <saml:AttributeValue
                xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">my_contract_number</saml:AttributeValue>
        </anp:UserInfo>
</anp:AuthnQuery>
```

Figure 8: Authentication Query Example (II)

- Authentication Statement: used by servers to communicate the result of the authentication to the user. Three possible authentication results are defined: “Successful”, “Failed” and “Redirect”. The first two values are straightforward. They inform that the authentication process has finished and it has either succeeded or failed. “Redirect”, on the other hand, indicates that the authentication process has not completed yet and it needs to be continued by contacting another server, whose URL is returned within the Authentication Statement. In case of redirection, the Authentication Statement also contains information about whether the client should keep the HTTP session state until the authentication process finishes. This may be necessary if the server needs to maintain state information to later proceed with the authentication. Only insensitive information should be stored this way to prevent security attacks.

  Figure 9 shows an example of Authentication Statement corresponding to the Authentication Query of Figure 7. The Authentication Statement is represented by the XML element *AuthnStatement*. The authentication result is specified with the *AuthnResult* element. In the example, the authentication process is successfully completed. Therefore, no further XML elements need to be included in the message.

```
<anp:AuthnStatement>
        <anp:AuthnResult>Successful</anp:AuthnResult>
</anp:AuthnStatement>
```

Figure 9: Authentication Statement Example

Figure 10 shows another Authentication Statement example, corresponding to the Authentication Query of Figure 8. In this case, before completing the Liberty Alliance authentication process, the service provider needs the user to directly authenticate with the IDP he previously selected. The "Redirect" authentication result is indicated within the *AuthnResult* element. The information associated to the redirection is encapsulated within the *RedirectInfo* element. The URL to which the user must be redirected, which may include various query parameters, is specified with a *RedirectURL* subelement. Whether session information must be kept or not, is indicated with a *SessionInfoRequired* subelement.

```
<anp:AuthnStatement>
        <anp:AuthnResult>Redirect</anp:AuthnResult>
        <anp:RedirectInfo>
                <anp:RedirectURL>https://my_identity_provider_1:8443/Sahara_SSO_idp/authn?RequestID=63tvPVdHU6RVTj3
                GYJc0fJ%2BNZo%2F4&amp;MajorVersion=1&amp;MinorVersion=0&amp;IssueInstant=2005-04-
                18T18%3A38%3A29Z&amp;ProviderID=my_service_provider_1&amp;ForceAuthn=false&amp;IsPassive=false&a
                mp;Federate=false&amp;ProtocolProfile=http%3A%2F%2Fprojectliberty.org%2Fprofiles%2Fbrws-
                art</anp:RedirecURL>
                <anp:SessionInfoRequired>true</anp:SessionInfoRequired>
        </anp:RedirectInfo>
</anp:AuthnStatement>
```

Figure 10: Authentication Statement Example (II)

- Authentication Continuation Query: used by clients to request that a previously uncompleted authentication process proceeds using the information provided by the intermediary server.

  An example of Authentication Continuation Query corresponding to the Authentication Statement in Figure 10 is shown in Figure 11. The information returned by the IDP is sent within a *RedirectedURL* element.

```
<anp:AuthnContQuery>
        <anp:RedirectedURL>
        https://my_service_provider_1:8443/Sahara_SSO_sp/assertion?SAMLart=AANsa1OLjGwEmBu02O2zdkIINB9kaxw9nfsxaZ
        DwBTo5W%2Fu6I4CL87D%2B </anp:RedirectedURL>
</anp:AuthnContQuery>
```

Figure 11: Authentication Query Example (II)

Error handling is also supported in the protocol, but we are not going to enter in our description into that level of detail.

5 Policy engine

In this section we describe our client-side policy engine, which selects the appropriate SSO scheme while protecting the privacy of user information in public WLAN environments. In the first subsection, we give an overview of the policy engine characteristics. Section 5.2 presents the details of the XML languages employed for specification of the policy engine information. Section 5.3 describes the policy engine component blocks. Section 5.4 presents a possible optimization which was finally discarded given the doubts about its actual benefits.

5.1 Overview

In federated public WLANs, a user roams across networks operated by different providers. Users may roam independently of whether they intend to or not. To maximize the security of their user data, they should always be notified when roaming is to occur and be forced to re-authenticate or manually acknowledge it. This scheme is less usable as the frequency of inter-system roaming increases. To achieve seamless network access, we require the following desirable features for the client software:

- Protection of user authentication information against exposure to entities not allowed to see it.
- Minimize user intervention for the sign-on process.

These two features require automated access control for the user's authentication information. The access control should be performed according to user-defined policy rules.

The purpose of our policy engine is to provide automated access control for the user's authentication information, to protect his sensitive information from exposure to untrusted service providers when roaming, and at the same time reducing the need of user intervention. Our policy engine supports that desirable user-defined policy-based access control for user authentication information.

The policy engine is an independent module that can be invoked through a simple API from an authentication negotiation client, link layer network access client or standard web browser. It requires as input an XML file formatted according to the XML schema definition for the authentication capabilities statement of the Authentication Negotiation Protocol. This file contains the authentication server ID, with companion information to verify it, the requested user information, and context information, such as the different authentication alternatives available. Taking into account user defined polices and preferences, the policy engine returns as output an indication of whether any of the choices presented in the input authentication capabilities statement is acceptable. If positive, it also returns the selections made from the alternatives offered together with the required user information for them.

The advantages of public WLAN authentication using such a policy engine are:

- Generality: Since the policy engine is built as an independent component with simple APIs, network access clients, like a web browser or an EAP-TLS client, can employ it.
- Scalability: Since users define access control rules using server attributes (e.g., role) and specific server IDs, rule management overhead is kept low even when many WLAN providers with different server IDs are part of the WLAN federation.
- Flexibility: Given a standard vocabulary of the rule specification for user authentication information, users can customize their own access control rules for their authentication information.

5.2 Language specification

An XML-based access control language is used to define access control rules for the user information. An example is shown is Figure 13. Each rule is defined within a *rule* element. The policy rules specify entities to be authorized to access authentication information elements. The authentication information elements are specified with XML *authn_info* elements, and the entities, with *subject* elements. Within the subject element, individual entities may be specified by indicating its name within the *id* subelement. Alternatively, and as shown in the example, the subelement *attribute* may be used to refer to all the entities satisfying certain property. In our prototype we support three attribute values: trusted, untrusted and default. The policy engine

could determine whether the requester of the information is trusted or not using public key cryptography, although this functionality has not been implemented yet. In Section 9 some suggestions about how check the server trustworthiness are presented. The “default” attribute is defined to specify the policy engine behavior when no other rule is matched or when the information that can be accessed according to the rule matched is not enough for any of the authentication choices offered by the requester. This behavior can be defined because the policy rules may also include provisional actions to be fulfilled before returning the user information. The provisional actions are indicated with *provisional_action* elements. The values supported are: “user_notification”, which means that a message must be shown to the user announcing him that certain information is being returned; “user_ack”, where the user must be prompted to acknowledge that the information is given away; and “user_input”, where the user is required to manually enter the information he wants to give to the requester. For example, if the policies evaluation outcome was that the user’s identity should be returned along with the “user_ack” provisional action, the policy engine should ask the user and wait for his consent before passing the user’s identity to the requester. If no provisional action is required, the information is automatically returned.

```
<policy>
   <rule>
      <subject>
         <attribute>trusted</attribute>
      </subject>
      <authn_info href="loginInfo/IDPName"/>
      <authn_info href="loginInfo/Domain"/>
      <authn_info href="loginInfo/UserName"/>
      <authn_info href="loginInfo/Password"/>
      <authn_info href="contractInfo/ContractNumber"/>
      <provisional_action name="user_ack"/>
   </rule>
   <rule>
      <subject>
         <attribute>untrusted</attribute>
      </subject>
      <authn_info href="loginInfo/IDPName"/>
      <provisional_action name="user_notification"/>
   </rule>
   <rule>
      <subject>
         <attribute>default</attribute>
      </subject>
      <provisional_action name="user_input"/>
   </rule>
</policy>
```

Figure 12: Policy Rules Example

An XML-based language is also used to specify the user preferences. Figure 14 shows a preferences file example. Preferences are defined separately for authentication methods, charging options and identity providers. Given the flexibility of the language, new preferences categories could be added in the future without requiring any modification to it. There are two possible ways of indicating preferences for a group. The first one is by enumerating the different options within a *PreferenceGroup* element. Each of the alternatives is specified with a *GroupMember* subelement, where the *name* attribute indicates the option and the *weight* attribute, its assigned weight. Non-enumerated options are assigned a default weight of zero. The second alternative is specifying the algorithm used to order the different choices with a *WeightingEquation* element. The *equation* attribute indicates the ordering mechanism and the *type* attribute, the category to which it must be applied. In our prototype, only one algorithm is supported, "inverse_linear". It can only be used with numeric values, such as the price of charging options, sorting them in inverse order to their absolute value. Options to which the ordering algorithm cannot be applied are placed at the end of the sorted list.

```
<PolicyPreferences>
   <PreferenceGroup type="AuthMethod">
      <GroupMember name="local" weight="10"/>
      <GroupMember name="liberty" weight="2"/>
      <GroupMember name="radius" weight="1"/>
   </PreferenceGroup>
   <WeightingEquation type="ChargingOption" equation="linear"/>
   <PreferenceGroup type="IDProvider">
      <GroupMember name="my_identity_provider_1" weight="5"/>
      <GroupMember name="my_identity_provider_2" weight="3"/>
   </PreferenceGroup>
</PolicyPreferences>
```

Figure 13: Policy Preferences Example

Finally, the secured user's information is also specified in an XML-based language. Figure 15 shows an example of secure information file. For each account the user holds, he can indicate the data associated to it with an *IDProviderInformation* element. The identity provider is identified with an *IDPName* subelement, and its administrative domain is indicated with a *Domain* subelement. The user's name and password are specified with the subelements *UserName* and

Password respectively. Finally, the user's preferred credit card for that account can be indicated with a *CreditCardName* subelement. This last subelement acts as a reference to a *creditCardInformation* element, and its value must match the name attribute of any of the *creditCardInformation* elements contained in the secure information file. Within a *creditCardInformation* element, the credit card's data is specified with the subelements *holderName*, *number* and *expirationDate*.

```
<secureConfiguration>
   <IDProviderInformation>
      <IDPName>my_identity_provider_1</IDPName>
      <Domain>eecs.berkeley.edu</Domain>
      <UserName>my_user_1</UserName>
      <Password>my_password_1</Password>
      <CreditCardName>my_creadit_card_1</CreditCardName>
   </IDProviderInformation>
   <IDProviderInformation>
      <IDPName>my_identity_provider_2</IDPName>
      <Domain>eecs.berkeley.edu</Domain>
      <UserName>my_user_2</UserName>
      <Password>my_password_2</Password>
      <CreditCardName>my_credit_card_2</CreditCardName>
   </IDProviderInformation>
   <creditCardInformation name="my_credit_card_1">
      <holderName>my_name</holderName>
      <number>1111222233334444</number>
      <expirationDate>01/06</expirationDate>
   </creditCardInformation>
   <creditCardInformation name="my_credit_card_2">
      <holderName>my_name</holderName>
      <number>2222333344445555</number>
      <expirationDate>12/06</expirationDate>
   </creditCardInformation>
</secureConfiguration>
```

Figure 14: Secure Information Example

5.3 Component Blocks

The policy engine major components are shown in Figure 15. The policy check component is where all the logic resides. It performs a policy compliance check based on the input parameters and user-specified access control and preferences rules, executes any mandated provisional actions and outputs the granted information. The other components are information repositories containing the policy rules, policy preferences, and sensitive user's information. The policy check component is divided into two subcomponents: root component and secure component.

The root component is the only component that has direct interaction with the requester and is where the XML file with the authentication capabilities statement is input. Once given a properly formatted XML file, the root component does some initial parsing to gather information before it proceeds. It extracts the authentication methods, charging options and identity providers from the input XML file, and assigns each of them a weight according to the user-defined policy preferences. Then, the policy engine root component computes all the {authentication method, charging option, IDP} triplets allowed following the indications of the authentication capabilities statement, calculates their weight as the product of their components' weights, and orders them according to that weight. The total weight is computed as the product of partial weights to guarantee that triplets including not desired options, for which the weight equals zero, also have a zero weight. Next, the policy rules are evaluated in order until a match is found. For each rule, the root component must check first whether the provider requesting the information belongs to the group of entities specified in the *subject* element. This condition will be satisfied if the provider's identifier is explicitly included within an *id* subelement or if the provider satisfies all of the properties indicated within *attribute* elements. In our prototype, as previously mentioned, only the attributes "trusted", "untrusted" and "default" are supported. The evaluation of the server trustworthiness is not implemented yet and we just classify every provider as trusted or untrusted, as dictated by a configuration parameter. If the subject conditions are satisfied, then the root component must check if there is any triplet for which all of the required information can be accessed according to that access rule. The triplets are evaluated in the order they were set according to the policy preferences until a valid one is found. If there is no matching triplet or if the subject conditions were not satisfied, the next policy rule is evaluated. In the case that no rule is found for which at least one triplet is allowed, the root component will finalize the policy engine execution returning an indication of such event. Otherwise, it will contact the secure component to retrieve any user information required for the selected triplet. Then, it will execute the provisional actions indicated by the matched rule, if any. Finally, if the provisional actions successfully complete, it will return both the selected {authentication method, charging option, IDP} triplet and the user information, together with an indication that automatic submission of information succeeded. If any of the provisional actions fail, the root component will simply

return an indication that information retrieval was not allowed. This may happen, for example, if the user does not consent information submission in a “user_ack” provisional action.

The secure component is in charge of accessing the user’s critical information. To prevent unauthorized access to the stored information, this information is cryptographically protected. We have developed the HomebrewCryptor, a subcomponent of the secure component that takes care of encrypting/decrypting the file containing the user’s sensitive information. The HomebrewCryptor makes use for encryption of the Blowfish cipher [21], using as secret key the MD5 hash of a user supplied password. When the secure information needs to be accessed by the policy engine, the secure component must ask the user for the encrypting password, and then make use of the HomebrewCryptor subcomponent to decrypt the file. As an optimization, instead of asking the user for the password each time the information needs to be accessed and going through all the decryption process, the secure component keeps the secure information in memory after the first time it is retrieved. This approach has the disadvantage that, if the user modifies this information once it has been read by the policy engine, the secure component will not be aware of the changes and the old information will continue to be used. However, this can be easily prevented if the user restarts the policy engine after making modifications to the secure information. A utility called EncryptFile is provided to assist the user in securing its sensitive information after making modifications to it. It prompts the user to supply an encrypting password and makes use of HomebrewCryptor to encrypt plain text files.

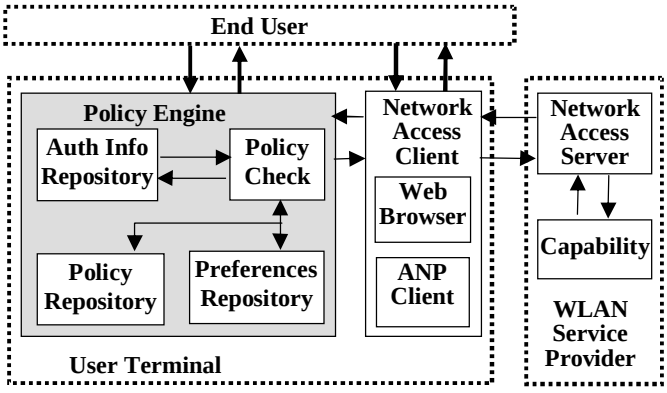


Figure 15: Policy Engine Block Diagram

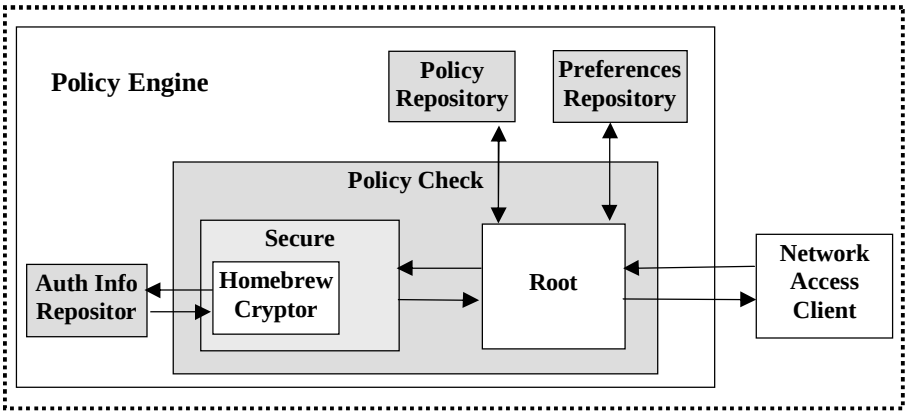


Figure 16: Policy Engine Detailed Block Diagram

5.4 Service Provider Specific Policy Rules

A possible to extension to the policies consists on maintaining prestored selections for particular entities. This would save the time of reevaluating the policy rules each time a user roams into the same network. The policies might include information about the authentication capabilities of the entities specified in the *subject* elements and the user's preferred options for those entities. The preferred options could be stored in the elements *chosen_idp*, *chosen_charging_option* and *chosen_auth_method*. Regarding the authentication capabilities, the lastUpdateInstant of the Authentication Capabilities Statement could be stored in a capabilities_lastUpdateInstant element. When first logging into a service provider's network with authentication capabilities announcement functionality, the policy engine would automatically select the appropriate options following the general policy rules. Alternatively, the user could manually specify which options he wants. After having chosen them, he would be asked whether he wants to save those options for future automated selection. If so, or if automatic selection by the policy engine was done, the policy engine would save the current options to a service provider specific policy. For subsequent times that the user roamed into that provider, the policy engine would compare the lastUpdateInstant of the authentication capabilities statement associated to the saved options from the last sign on to this service provider and the lastUpdateInstant of the currently received authentication capabilities statement. If they matched, then the policy engine would proceed with the automatic selection. However, if they did not, the policy engine would

evaluate the general policy rules again. An example of a service provider specific policy is shown in Figure 17.

```
<policy>
  <rule>
    <subject>
      <id> … </id>
    </subject>
    <options>
      <chosen_idp>…</chosen_idp>
      <chosen_charging_option>…</chosen_charging_option>
      <chosen_auth_method>…</chosen_auth_method>
      <capabilities_lastUpdateInstant>…</capabilities_lastUpdateInstant>
    </options>
  </rule>
</policy>
```

Figure 17: Service Provider Specific Policy Rule

The time savings derived from such cached selections is questionable, however. The policy engine might spend time looking for a specific policy rule and comparing it with the current Authentication Capabilities Statement and, at the end, need to evaluate the general policy rules anyway. Besides, as shown by our experimental results, the time required for general rules evaluation is small. Therefore, we have decided not to support this feature in our system.

6 Compound link layer and web based authentication

This section describes our compound link layer – web based authentication scheme. First, an analysis of the security threats faced by traditional web authentication systems is performed in Section 6.1. Section 6.2 presents and overview of our authentication proposal and its message sequence. The architecture of the subsystem in charge of carrying out the compound authentication is illustrated in Section 6.3. Section 6.4 describes the Roaming Client, a component developed to facilitate automatic authentication using our compound scheme. Finally. Section 6.5 performs a security analysis of our system.

6.1 Security Threats in Web-based Authentication

Most public wireless LAN systems use web-based authentication schemes, and users can get IP-level network access before showing their identity and credentials. Although this open-style of network authentication enables fine-grained service authorization and accounting options, lack of lower-layer cryptographic bindings yields security vulnerabilities. Examples include:

- Theft of service by spoofing IP or MAC address;
- Eavesdropping because of lack of data encryption;
- Message alteration because of no message integrity check; and
- Denial of service attack by the placement of rogue access points.

The key to avoiding these security threats is to have a cryptographic binding between the user and the network. As explained in Section 2, IEEE 802.1X port-based network access control is being deployed in corporate wireless LANs, and it uses such a cryptographic method for user authentication and network access control. Normally IEEE 802.1X adopts conventional closed-style mutual authentication and assumes a pre-shared secret between users and the network. However, we cannot assume a pre-shared secret in public wireless LANs to accommodate one-

time users that use credit-card authorization, or to provide free contents for non-subscribers. Table 1 summarizes the characteristics of web-based and Layer 2-based AAA schemes. Shaded boxes in Table 1 represent the advantage of each authentication scheme.

	Web-based	**IEEE 802.1X/11i**
Support	Most public wireless LAN service providers	Corporate networks (only in 802 LANs)
Pre-shared Secret	Not necessary (users can use credit-card authorization)	Necessary
Encryption	N/A	Per-station RC4, AES (802.11i)
Authentication	SSL-protected password	EAP-TLS (certificate-based)
Access Control	IP/MAC address	Cryptographic Method
Accounting	Fine-grained	Only at boot time and periodic re-authentication

Table 1: Comparison of Web-based and Layer 2-based AAA schemes

6.2 Compound Layer 2 and Web-based Authentication Overview

To ensure cryptographically protected access in public wireless LANs, we have developed a compound Layer 2 and Web-based authentication approach. To use this scheme, the WLAN service provider must have 802.1X-capable access points and authentication servers. The user terminal must also have an 802.1X client, but this stipulation is not an issue since some operating systems bundle an 802.1X client. If this is not the case, free 802.1X clients are readily available for download [16]. The network may accommodate 802.1X-incapable legacy user terminals to account for backward compatibility. However, allowing 802.1X-incapable clients, thus bypassing link layer authentication, the network becomes vulnerable to common web-based authentication security holes.

The compound authentication message sequence diagram is shown in Figure 18. In our scheme, the user terminal first associates with an access point (Step 1) and then establishes a L2 session key using guest (or anonymous) account in EAP-TLS message exchange in IEEE 802.1X authentication (Step 2). A guest account is used in L2 authentication for the reason that we cannot assume a pre-shared secret between users and the network in public wireless LAN. Note

that the EAP-TLS specification does not mandate client authentication. The client will not authenticate the server either, for the same reason that we cannot assume a pre-shared secret if we want to allow roaming of clients into unknown providers.

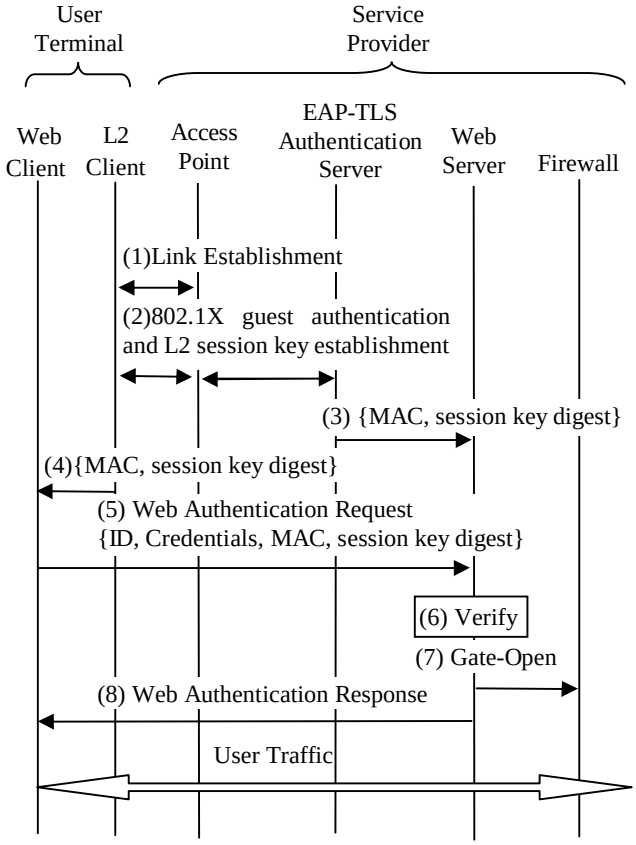


Figure 18: Compound L2 and Web Authentication Message Sequence

After Step 2, the encryption key derived from the L2 session key encrypts packets transmitted over the air. The EAP-TLS authentication server notifies the {MAC address, L2 session key digest} pair to the web server for later use (Step 3). Similarly, the L2 client in the user terminal passes {MAC address, L2 session key digest} to the Web client (Step 4). MD5 algorithm is used for generation of 64-bytes digests. Then the user terminal sends a Web authentication message to the Web server, with {ID, credentials, MAC address, L2 session key digest} quadruplets (Step 5). In Step 5, MAC address and L2 session key digest embedding is required to avoid theft of service by race timing attack from malicious clients. The session key digest is sent instead of the session key itself to save communication bandwidth. Finally, the Web server verifies the quadruplets (Step

6), changes firewall rules so that the user can access the external network if the verification succeeds (Step 7), and returns an authentication response to the user terminal (Step 8).

6.3 Architecture

The components involved in the compound link layer - web based authentication are shown in Figure 21. They are basically those previously mentioned when describing the message sequence: the L2 802.1x client and EAP-TLS authentication server, and the web client and web server. Two additional modules have been introduced to facilitate the inter-process communication between L2 and web components. Although the inter-process communication in the client (Step 4 in Figure 18) could have been accomplished using a customized web client, we have not explored this option. Rather, to increase client flexibility, we have developed a simple module, L2InfoCollector, that communicates with the 802.1x client to retrieve the link layer authentication information and make it available to other modules. Additionally, to allow backwards compatibility with commercial web browsers, the module may be directly invoked by the user, presenting him the information in a human readable format. If the service provider also provides support for compound authentication through traditional web browsers, it will ask for the link layer information using a standard web page and the user will be able to retrieve the information he needs to enter by making use of the L2InfoCollector utility.

For the inter-process communication at the server side, we use a central repository, L2InfoRepository, that accepts {MAC address, session key digest} pairs to store/ remove and requests to verify if a given {MAC address, session key digest} pair is present. The EAP-TLS authenticator, upon successful guest session establishment with a client, stores the client's MAC address and the session key digest in the repository. The web server, when an authentication request is received, requests the repository to verify the {MAC address, session key digest} pair received from the client.

In our prototype, as explained in section 4, we have considered two options for web authentication: manual authentication through standard web browsers and automatic authentication making use of the Authentication Negotiation Protocol client. For the first approach, the modifications required to support compound L2-web authentication are the ones previously described: the user should retrieve the L2 information using the L2InfoCollector, and

the server, which in this case would be a standard web portal, should ask for that information in the web page, accessing the L2InfoRepository to verify it and not authorizing any user who does not provide a correct {MAC address, session key address} pair. Actually, the web portal, does not contact the L2InfoRepository directly, but makes use of a new component which encapsulates the L2 information verification functionality, the L2 Authenticator.

For the second alternative, we introduce a new component, the Roaming Client, to preserve automatic authentication by invocation of a single component by the user. The Roaming Client, among other tasks, takes care of starting the 802.1x client and executing the L2InfoCollector, and invokes the ANP client when successful L2 connection is achieved, passing it the {MAC address, session key digest} pair. The ANP client has to send the L2 information together with the rest of user's credentials, and the ANP server has to invoke the L2 Authenticator to verify the L2 information before considering a user successfully authenticated.

As we have previously mentioned, the purpose of the Authentication Negotiation Protocol is to allow automatic processing of authentication capabilities and authentication requests in heterogeneous environments. A user may roam into networks performing compound link layer - web based authentication and networks which do not. Therefore, the protocol needs to be extended to support indication of whether {MAC address, session key digest} need to be sent and how these pieces of information should be identified. For this, a new XML element, *L2Authn*, is introduced as optionally part of the Authentication Capabilities Statement. Just its presence indicates that L2 information must be sent. The required pieces of information are specified with *UserInfoDesignator* subelements, Both ANP Authentication Queries and ANP Authentication Continuation Queries should include *UserInfo* elements containing the requested pieces of L2 information, independently of the authentication method selected. Figures 19 and 20 show examples of an Authentication Capabilities Statement and an Authentication Query resulting from adding the compound L2-web based authentication elements to the examples of Figures 6 and 8.

```
<anp:AuthnCapabilitiesStatement LastUpdateInstant="2005-01-01T00:00:00Z">
        <saml:Subject>
                <saml:NameIdentifier>my_service_provider_1</saml:NameIdentifier>
                <saml:SubjectConfirmation>
                        <saml:ConfirmationMethod>…</saml:ConfirmationMethod>
                        <ds:KeyInfo>...</ds:KeyInfo>
                </saml:SubjectConfirmation>
        </saml:Subject>
        <anp:L2Authn>
                <anp:UserInfoDesignator AttributeName="MacAddress"
        AttributeNameSpace="my_userinfo_namespace"/>
                <anp:UserInfoDesignator AttributeName="SessionKey"
                AttributeNameSpace="my_userinfo_namespace"/>
        </anp:L2Authn>
        <anp:IDPGroup>
                <anp:IDPList>
                        <anp:IDPName>my_identity_provider_1</anp:IDPName>
                </anp:IDPList>
                <anp:ChargingOptionIDReference>monthly_rate</anp:ChargingOptionIDReference>
                <anp:AuthnMethodIDReference>radius</anp:AuthnMethodIDReference>
                <anp:AuthnMethodIDReference>liberty</anp:AuthnMethodIDReference>
        </anp:IDPGroup>
        <anp:AuthnMethod>
                <anp:AuthnMethodID>radius</anp:AuthnMethodID>
                <anp:UserInfoDesignator AttributeName="UserName" AttributeNameSpace="my_userinfo_namespace"/>
                <anp:UserInfoDesignator AttributeName="UserPassword" AttributeNameSpace="my_userinfo_namespace"/>
        </anp:AuthnMethod>
        <anp:AuthnMethod>
                <anp:AuthnMethodID>liberty<anp:AuthnMethodID>
                <anp:UserInfoDesignator AttributeName="IDPName" AttributeNameSpace="my_userinfo_namespace"/>
        </anp:AuthnMethod>
        <anp:ChargingOption>
                <anp:ChargingOptionID>monthly_rate</anp:ChargingOptionID>
                <anp:ChargingInterval Order="1">
                        <anp:UnitPrice>0.1</anp:UnitPrice>
                        <anp:TimeUnit Unit="Minute">
                                <anp:Period>1</anp:Period>
                        </anp:TimeUnit>
                        <anp:ChargingMode>Constant</anp:ChargingMode>
                </anp:ChargingInterval>
                <anp:UserInfoDesignator AttributeName="ContractNumber" AttributeNameSpace="my_userinfo_namespace"/>
                <anp:ServiceIDReference>private_contents</anp:ServiceIDReference>
        </anp:ChargingOption>
        <anp:Service>
                <anp:ServiceID>private_contents</anp:ServiceID>
                <anp:ServiceDescription> Access to private contents through the provider's web portal </anp:ServiceDescription>
        </anp:Service>
</anp:AuthnCapabilitiesStatement>
```

Figure 19: Authentication Capabilities Statement Example with Link Layer Information

```
<anp:AuthnQuery>
        <anp:AuthnMethodIDReference>liberty</anp:AuthnMethodIDReference>
        <anp:ChargingOptionIDReference>monthly_rate</anp:ChargingOptionIDReference>
        <anp:UserInfo AttributeName="IDPName">
                <saml:AttributeValue xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">my_idp</saml:AttributeValue>
        </anp:UserInfo>
        <anp:UserInfo AttributeName="ContractNumber">
                <saml:AttributeValue
        xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">my_contract_number</saml:AttributeValue>
        </anp:UserInfo>
        <anp:UserInfo AttributeName="Key">
                <saml:AttributeValue
                xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">0123456789abcdef0123456789abcdef
                </saml:AttributeValue>
        </anp:UserInfo>
        <anp:UserInfo AttributeName="MacAddr">
                <saml:AttributeValue
                xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">11:22:33:44:55:66</saml:AttributeValue>
        </anp:UserInfo>
</anp:AuthnQuery>
```

Figure 20: Authentication Query Example with Link Layer Information

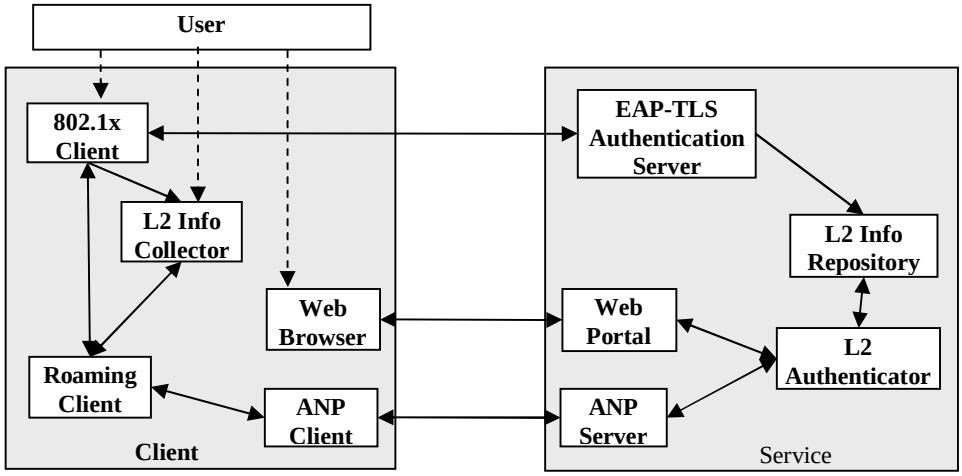


Figure 21: Compound Link Layer – Web Based Authentication Architecture

6.4 Roaming Client

As mentioned before, to facilitate automatic authentication when roaming between WLAN networks, a Roaming Client is introduced as part of our system. Part of its functionality is detecting the available wireless networks and attempt connecting to them following the priority order specified by the user. A graphical user interface that dynamically shows the available access

points and allows the user to configure each network's priority and authentication parameters is provided (see Figure 22).

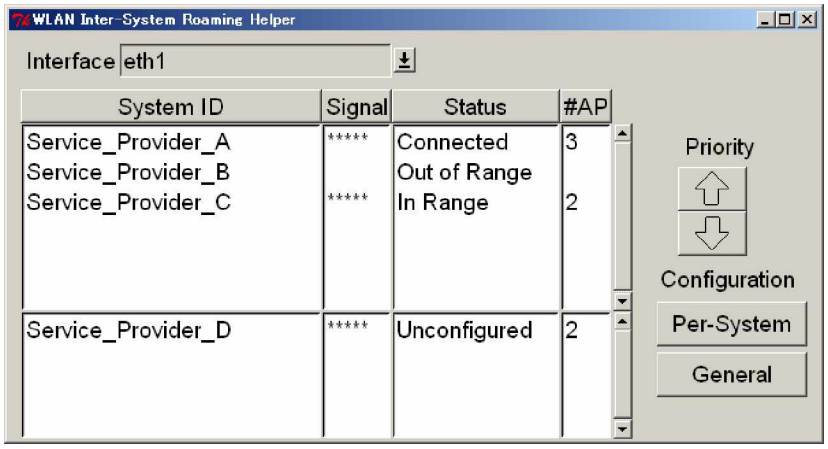


Figure 22: Roaming Client Graphical user Interface

The Roaming Client supports WLANs with no link level protection at all, WLANs with WEP authentication, and WLANs employing standard 802.1x authentication. It also supports static configuration of the IP address and rest of IP level parameters or dynamic assignment through DHCP. In this sense, our Roaming Client is similar to the Wireless Network Connection agent provided with Windows XP. However, the Roaming Client also integrates with our Compound Authentication and Authentication Adaptation Framework. It supports link level authentication using our modified 802.1x client for compound authentication, which does not validate the peer's certificate, followed by authentication negotiation execution using the ANP client, during which the client has the opportunity to verify the server's identity. In particular, this mode of operation is the one selected when attempting to connect to unconfigured networks if connection without link layer authentication fails. This way, clients can automatically and securely roam into the public networks they find protected with our proposed authentication system as they move around, without need to have any preconfigured shared secret. When connecting to a network using the modified 802.1x client, the Roaming Client invokes the L2InfoCollector to retrieve the client's MAC address and 802.1x session key. This way, they will be available to the ANP Client in case they are needed for compound authentication. Upon successful link level connection, and once the client has IP level connectivity, the Roaming Client invokes the ANP Client, so that the authentication negotiation takes place. Connection to the new network will be successful only if

the web authentication phase through the Authentication Negotiation Protocol is successful. The Roaming Client, once successful connection to a network is achieved, remains connected during a preconfigured time interval. Then it searches again for networks in range in case a more preferred option has become available.

6.5 System Security Analysis

In this section, we give a formal analysis of how the proposed compound Layer 2 and web authentication scheme can deal with various common security threats.

Theft of Service

A malicious user may spoof the IP and/or MAC address of legitimate user to take over the WLAN service. Address spoofing does not work in our scheme, because each user has its own encryption key established through the 802.1X guest authentication process. The access point simply discards the malicious user's packets if it can't decrypt them. Because all Layer 2 data frames are encrypted by a per-user key, it is difficult for a malicious user to guess the legitimate user's IP address.

A malicious user can attempt a race timing attack by taking advantage of guest authentication properties. In this case, he sends a Layer 2 authentication request just before a legitimate user's web authentication. The web server distinguishes the malicious user from a legitimate one by checking the {MAC, session key digest} pair embedded in the web authentication request with the one informed by the RADIUS server. Thus the race timing attack can be prevented.

A more sophisticated attack involves a rogue access point between the legitimate user and the legitimate access point. This rogue access point acts as a transparent SSL proxy to fake a legitimate user's network login process. Again, the web server can detect the attack by checking the {MAC, session key digest} pair embedded in the web authentication request message. It can't alter the MAC address or session key digest in the web authentication request message because the message is SSL-encrypted. It should be noted that 802.11i also has a countermeasure for such man-in-the-middle attack by conducting a 4-way handshake immediately after the 802.1X authentication.

Eavesdropping/Message Alteration

It is obvious that eavesdropping and message alteration can be prevented in our scheme, because all L2 data frames are encrypted by per-user key and have message integrity check codes. The per-user key is derived from the L2 session key that is established through EAP-TLS process in 802.1X authentication. Therefore, cracking a legitimate user's per-user key is as difficult as cracking TLS. It is also well known that the current 802.11 WEP (wired equivalent privacy) has security vulnerabilities of eavesdropping and message alteration [17]. Countermeasures for such security vulnerabilities are already taken into account in the 802.11i draft under standardization [8]. In the meantime, one can reduce the risk of such vulnerabilities by shortening 802.1X re-keying period.

Denial-of-Service

There are several DoS possibilities in an 802.11 wireless LAN [18], such as deauthenticating/disassociating a legitimate user, spoofing power save mode, and faking the carrier sense mechanism. Unfortunately, these DoS possibilities are inherent in the original 802.11 MAC protocol and our scheme does not solve such DoS attacks. Moreover, because every user is given link layer access in our scheme, a malicious user can disturb a legitimate user's communication by spoofing the latter's MAC address or flooding frames in Layer 2 networks. However, we consider theft of service to be a much more serious problem than DoS attacks, and thus have limited our scope to permit certain DoS scenarios. It is still possible for legitimate users and access points to detect DoS attack and notify it to the network management server.

7 Implementation

Two of the main components of our system, the Authentication Adaptation Framework and the Policy Engine, have been developed completely from scratch. For the server implementation we made use of the Java Web Services Developer Pack, since it provides a framework for easy development and deployment of new web applications. This decision determined the use of Java as programming language for the web portal and Authentication Negotiation Protocol server. At the client side, Perl has been mainly used, being both the ANP Client and the Policy Engine entirely implemented in this language.

Regarding the compound link layer – web based authentication, we have modified an existing 802.1x client and EAP-TLS authenticator. We have focused on clients running Linux operating system, since it was easier to find applications with available source code. The 802.1x client xsupplicant was selected, modifying it to skip the server's certificate validation and to output the established session key. The RADIUS server FreeRadius was chosen as EAP-TLS authenticator, also modifying it to support guest accounts and to store the client's MAC address and the MD5 digest of the established session key in our L2InfoRepository. The Roaming Client, L2InfoCollector and L2InfoRepository were developed from scratch using Perl.

One tricky issue to take into account is that, to allow exchange of the session key digest over HTTP when performing web authentication, only readable characters can be used. We overcome this limitation by sending the digest's hexadecimal representation. Exchange of the key or its digest between processes both at the client and the server side is also easier if they can be represented with strings. Therefore, both xsupplicant and freeRadius output the key or its digest in hexadecimal format. It really does not matter whether the digest calculation is done directly by the modules involved in the EAP-TLS authentication, that is, xsupplicant and freeRadius, or whether it is done by the modules in charge of retrieving, sending, or verifying it. In our system, the L2InfoCollector performs the transformation at the client side. At the server side, freeRadius is in charge of it.

To increase the performance of our system, we cache that information which is costly to generate and does not change often. In this category are the authentication capabilities statement at the server side and the user's secure information at the client side. At server initialization, the Authentication Capabilities Manager parses the XML file containing the server's authentication capabilities statement and stores in memory an Authentication Capabilities Response message ready to be sent to connecting clients. Each time an Authentication Capabilities Request is received, the pre-generated response is sent as reply, reducing significantly the resources consumed to answer each request as well as the latency perceived by the user. Regarding the user's secure information, it is parsed and stored in memory the first time that the Policy Engine needs to retrieve a piece of data from it. Subsequent accesses to the secure information do not require parsing the storing file again and are accomplished in almost no time. This way, the delay experienced by the client when connecting to new networks is notably reduced. One issue that must be taken care of when caching is done is updating the cached information when modifications to the sources of that information occur. In our system, the authentication capabilities statement or the secure information should be parsed again if they are edited while the server or client respectively is functioning. We have not implemented this functionality yet, although the system is prepared to easily incorporate it into the Authentication Capabilities Manager and Policy Engine.

Another implementation concern is how to integrate our Authentication Adaptation Framework, Policy Engine and Compound Link Layer – Web Authentication subsystems with other out-of-the-box components to build a complete authentication system for network access control. The most troublesome is the interaction with the firewall, which is required to block the traffic of unauthorized users. One issue to consider when developing an access control system like this, where authentication is done by a web server and actual access control is performed by a firewall, is the difference in privileges required or recommended for executing each of them. For security reasons, the firewall should only be managed by the superuser. However, the web server should not be run with superuser privileges, since this kind of software has proven to be vulnerable to attacks, and a malicious user gaining control of the web server could cause serious damage if acquiring superuser privileges. To allow firewall modification from the web server, while executing each of them with the appropriate privileges, we adopted as solution the

development and integration in our system of an intermediary process. This process executes with superuser privileges and is the only one communicating directly with the firewall, taking care of its management. The web server, running without superuser privileges, communicates its firewall modification requests to the intermediary process. Still, an attacker gaining control over the web server could send malicious requests to the firewall management process. However, the option of using the intermediary process is more secure than directly running the web server with superuser privileges.

Rather than having just one intermediary process, we have developed one for each of the possible firewall operations required by the system. This way, the processes' interfaces are simpler, since they must just provision for one operation each. The web server will communicate with the corresponding firewall management process in each case. Four firewall operations are supported, two required by the service model and two needed for redirect-based authentication schemes. The service model that we have selected for our prototype is a simple one in which unauthenticated users can only access some of the service provider local web contents. Authenticated users, on the other side, can access whatever IP address and port they want. Authorization is revoked after certain configurable time. Users desiring to keep their access to the external network enabled beyond that time, should reauthenticate before their authorization expires. To follow this model, our server must be able to interact with the firewall to give or remove unlimited access to certain users. Additionally, as mentioned in section 3, when users employ for authentication schemes based on redirection, a hole must be open in the firewall to temporarily allow communication between the user and the provider to which he must be redirected to continue the authentication. This requires that the server is also able to request access for certain users but only for certain destination address and port. The server should be capable of removing that permit too once the authentication process has finished.

One last observation regarding the integration with the firewall is that the firewall management processes are dependant on the specific firewall implementation employed, since the different alternatives available offer different management interfaces. However, the web server is abstracted from these differences, since we have designed the interface offered by the intermediary processes to be general enough so that it can be maintained invariable upon firewall changes. This way, differences between firewalls are encapsulated within the intermediary management

processes and the rest of the system does not need to be aware of them. In our prototype we have chosen as firewall the Linux IPtables implementation, and we have developed the corresponding firewall management processes adapted to it.

Apart from the firewall, the other most important software components we had to integrate with our system are the modules in charge of RADIUS and Liberty authentication. Again, the interface provided by each authentication module is completely different. To abstract our server as much as possible from this fact, we have defined a common interface for authentication modules. The adaptation of a particular module to this interface is encapsulated within the corresponding adaptation object, which is the only one that knows about the specifics of the authentication module interface. We have implemented the adaptation code required for the RADIUS and Liberty authenticators employed in our prototype. Future incorporations of new authentication modules into the system are easily achieved by simply adding the corresponding adaptation code.

8 Evaluation

We have developed a prototype to prove the viability of the architectural concepts we have described above and to evaluate the performance of a system that integrates them. In this section we describe this prototype and present its performance. Section 8.1 presents the testbed employed for functional testing; Section 8.2 describes the modifications done to the testbed for authentication latency measurement and the results obtained; finally, Section 8.3 presents the testbed employed for load testing and the maximum number of authentications per second supported by our system.

8.1 Testbed

To demonstrate the viability of the system from a functional point of view, we have built a testbed consisting on two service providers, SP1 and SP2, and three identity providers, IDP1, IDP2 and IDP3, where SP1 has trust relationships with IDP1 and IDP2, and SP2 has trust relationships with the three identity providers. All the providers support both RADIUS and Liberty authentication. Additionally, they support local authentication through LDAP. These providers belong to different administrative domains. For limitations in the number of physical resources available, all the software associated to SP1 and IDP1 share the same machine, wizard.cs.berkeley.edu, and all the system components corresponding to SP2 and IDP2 share another machine, vancouver.cs.berkeley.edu. These two machines have similar hardware characteristics: Pentium III at 864 MHz with 256 MB of RAM. A third machine, sydney.cs.berkeley.edu, houses the identity provider IDP3. This last machine is a Pentium II 266 MHz with 128 MB of RAM. The three machines are connected between them using the Katz Network, a 100Mb/s Fast Ethernet.

Each service provider offers wireless network access service through an Airespace 1200 Access Point. An Airespace 4000 WLAN Switch is used to centralize the management functions of both access points, AP1 and AP2. Therefore, this WLAN switch acts as an intermediary

between the service providers and the access points. Both the access points and the machines hosting the service providers are physically connected to this device, which switches the traffic traversing it so that packets to/from AP1 come from/to SP1 and, analogously, traffic to/from AP2 comes from/to SP2. The switch also has a direct connection to the Katz Network, to allow remote management and configuration. The Airespace equipment is 802.1x capable. To use this functionality, the address and port of the server performing EAP-TLS authentication must be configured. Unfortunately, all the access points connected to an Airespace WLAN switch must share the same EAP-TLS authenticator. As a result, both SP1 and SP2 must use the same freeRadius server to authenticate their 802.1x users, and, in consequence, the same users' database. However, although having separate EAP-TLS authenticators would have been the best option and it would have better modeled real environments, sharing is not that problematic in our system. Since we use our compound link layer – web based authentication scheme, where clients' certificates are not validated, no user's database is required, and therefore, no data needs to be shared between the two service providers. We host the EAP-TLS enabled freeRadius server in wizard.cs.berkeley.edu. Communication between the Airespace switch and the EAP-TLS authenticator, freeRadius in our case, occurs through the switch management connection. We select the RADIUS server of one of the service providers for both.

Finally, one 697 MHz wireless enabled laptop with 123 MB of RAM is utilized as user terminal. Figure 23 shows the testbed physical layout and the hardware specifications of its components:

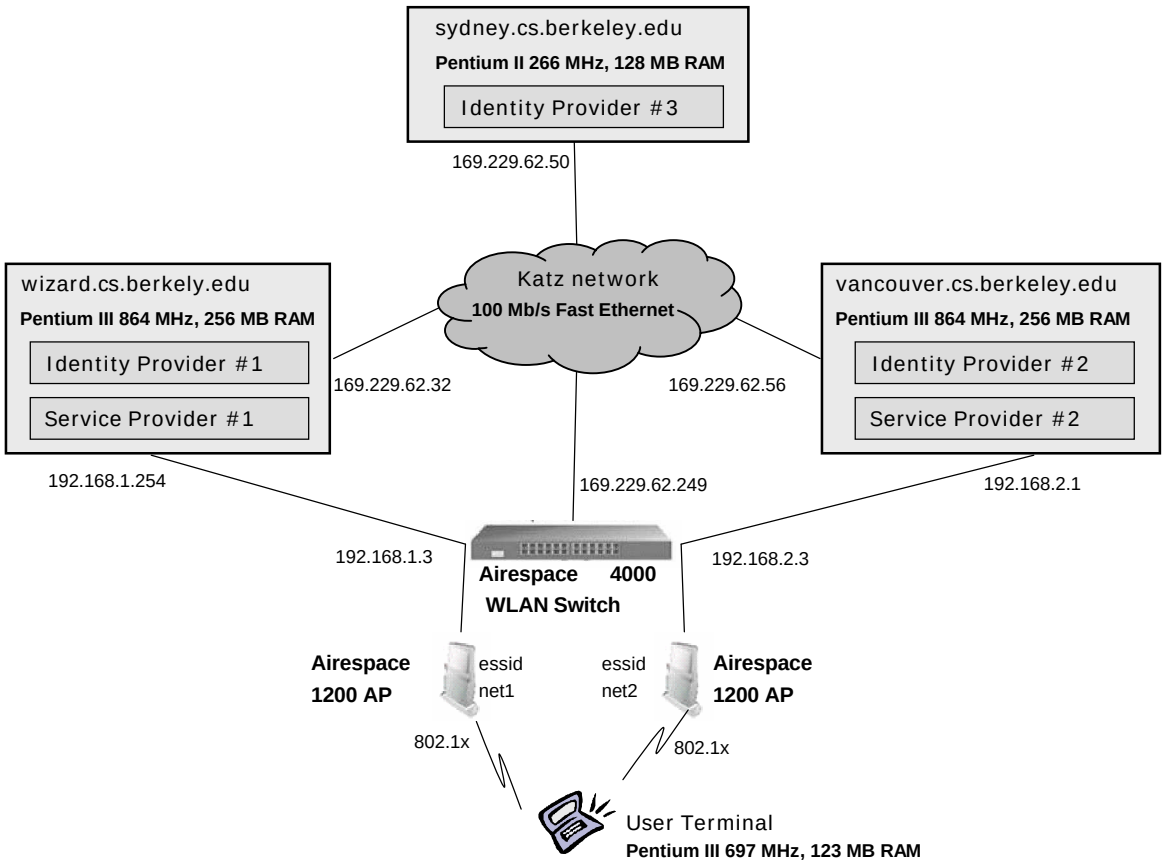


Figure 23: Testbed Physical Layout

Regarding the software running in each machine, we have used open-source software whenever possible. Red Hat Linux is employed as operating system in all the servers. On top of it, the following components have been used for each service provider:

- Iptables v1.2.6a, for the software firewall
- FreeRadius v0.8.1 for RADIUS id/password web authentication
- Sun Interoperability Liberty Prototype v0.1 for Liberty Alliance web authentication. This code has been partially modified to remove the part in charge of providing web portal services and transform it in a simple authenticator component that satisfies our common authenticator interface.
- Sun Java Web Services Developer Pack v1.0.01, as framework within which our web portal, our ANP Server and the Liberty Prototype code are executed

- ForgeNet RADIUS Client v0.9c, for communication with the local RADIUS server from the web authentication code.
- OpenLDAP LDAP Server v2.1.12, to store the local users' information
- Our web portal and ANP Server code
- Additionally, one instance of our modified version of FreeRadius v0.8.1 for EAP-TLS authentication is shared by the two service providers.

The RADIUS servers are configured to proxy authentication requests for users belonging to foreign administrative domains and with which the service provider has trust relationships. After contacting the local RADIUS server, this will directly contact the RADIUS server in charge of authentication in the user's domain if such domain is other than the local one. In case of authentication requests for local users, the RADIUS servers are configured to contact the local LDAP server for id/password validation.

The Liberty Prototype code is also configured to use LDAP for authentication of local users. The delivery package is prepared to utilize the generic interface LoginContext to perform id/password authentication, but it is not linked to any particular login module. We configured it to employ the JndiLoginModule, using as server the local LDAP server.

For each identity provider, the following components have been used:

- FreeRadius v0.8.1 for RADIUS id/password web authentication
- Sun Interoperability Liberty Prototype v0.1 for Liberty Alliance web authentication
- Sun Java Web Services Developer Pack v1.0.01, as framework within which our the Liberty Prototype code is executed
- OpenLDAP LDAP Server v2.1.12, to store the local users' information

Again, both the RADUS server and the Liberty Prototype code are configured to use the local LDAP server to perform user's id/password validation. To save resources, the SP and IDP residing in the same machine share the same instance of the LDAP server, but using separate databases. SP and IDP are properly configured to issue their authentication requests against their corresponding users' database.

Finally, the user terminal also runs Red Hat Linux as operating system, and includes the following components:

- Our modified version of Xsupplicant v0.6 for EAP-TLS authentication (802.1X client)

- Dhclient DHCP client
- Our Roaming Client, ANP Client and Policy Engine code
- Perl v5.8.0 plus the following Perl modules, for execution of the three client modules just previously enumerated (Roaming Client, ANP Client and Policy Engine):
  - Ø Tk.pm v804.027, for the Roaming Client graphical user interface
  - Ø SOAP:Lite v0.60, for the ANP Client SOAP communication
  - Ø Crypt/SSLay v0.51, for SSL communicating in the ANP Client
  - Ø XML:LibXML v1.58, XML::LibXML::Common v0.13, XML::NamespaceSupport v1.08, and XML::SAX v0.12, for parsing of XML files in the ANP Client and Policy Engine
  - Ø Digest::MD5 v2.33, for L2InfoCollector 802.1x session key digest calculation
  - Ø Term/ReadKey v2.30, Crypt/CBC v2.14, and Crypt/Blowfish 2.09 for decrypting the secure information in the Policy Engine

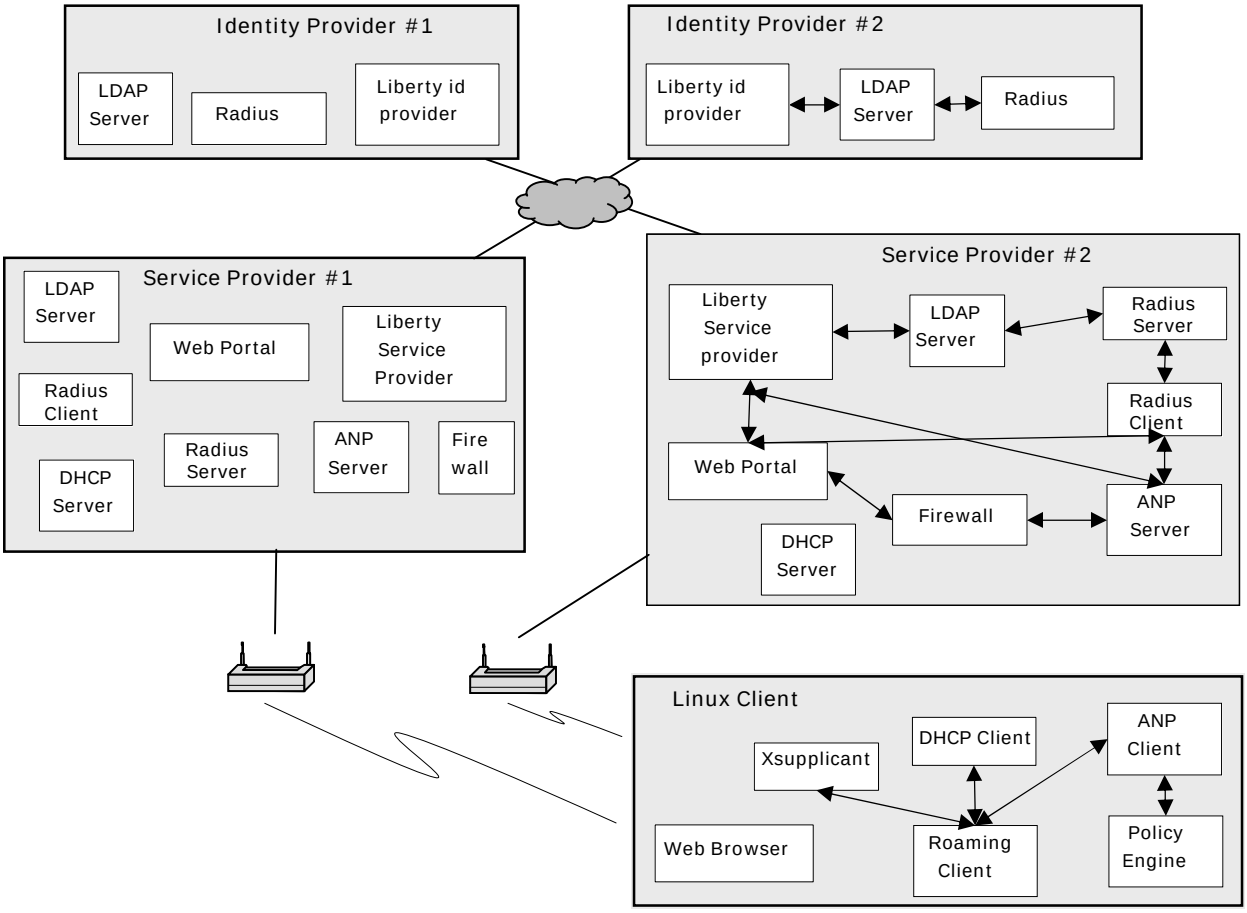


Figure 24: Testbed Software Components

Figure 24 shows the software components running in each machine. Interactions between components residing in the same machine are indicated with arrows. Interactions between different components running in different machines and the protocols used for the communication are shown in Figure 25.

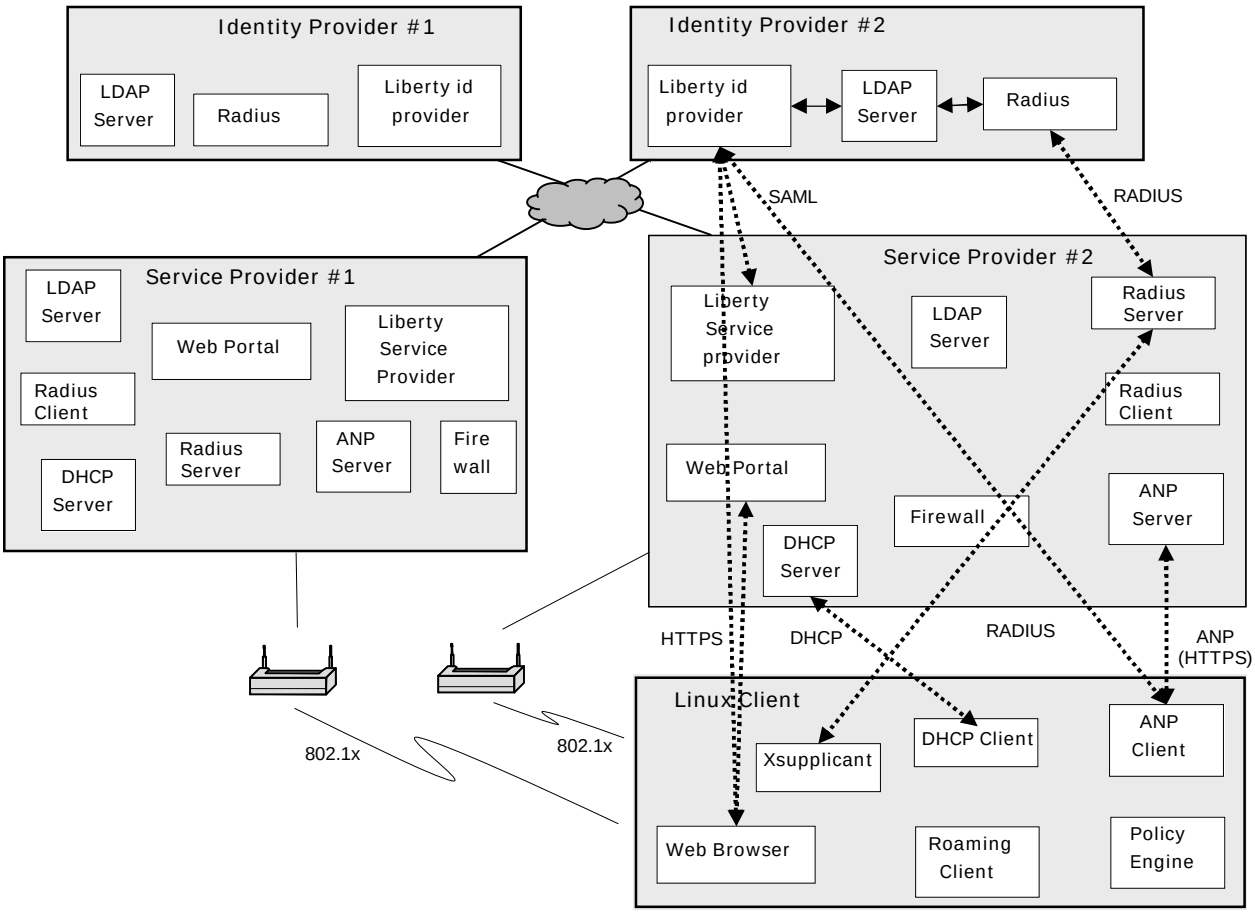


Figure 25: Testbed Communication Protocols

With the testbed previously described we have tested that our system prototype successfully authenticates legitimate users and allows them to communicate with the Internet while blocking access to unauthorized users. Both RADIUS and Liberty authentication with remote trusted domains work as expected. Local authentication through LDAP works properly too. We have also checked that users providing right login/password credentials but incorrect {MAC address, session key digest} pairs are denied access. We have run test cases where automatic authentication using the Roaming Client, ANP Client and Policy Engine is done, as well as test

cases where the user authenticates manually using a standard web browser and the L2InfoCollector utility.

8.2 Authentication Latency

To analyze the authentication latency of our system, the testbed described in previous section has been modified a bit. To avoid wireless-specific delay variance, link layer authentication delay and other delay have been measured separately. Therefore, after computing the link layer authentication latency, which results in an average value of 0.124 seconds, we substitute the wireless enabled laptop used as client by a machine connected directly to the Fast Ethernet that interconnects the servers. This way, the link delay between client and server, as well as between the servers themselves, is minimal, and all the measured latency can be attributed to the system itself. The new client machine is a Pentium III at 864 MHz with 256 MB of RAM Additionally, only the two faster servers are used, one of them playing the role of service provider and the other of identity provider.

Figure 26 shows the testbed used for non-wireless related latency measurement.

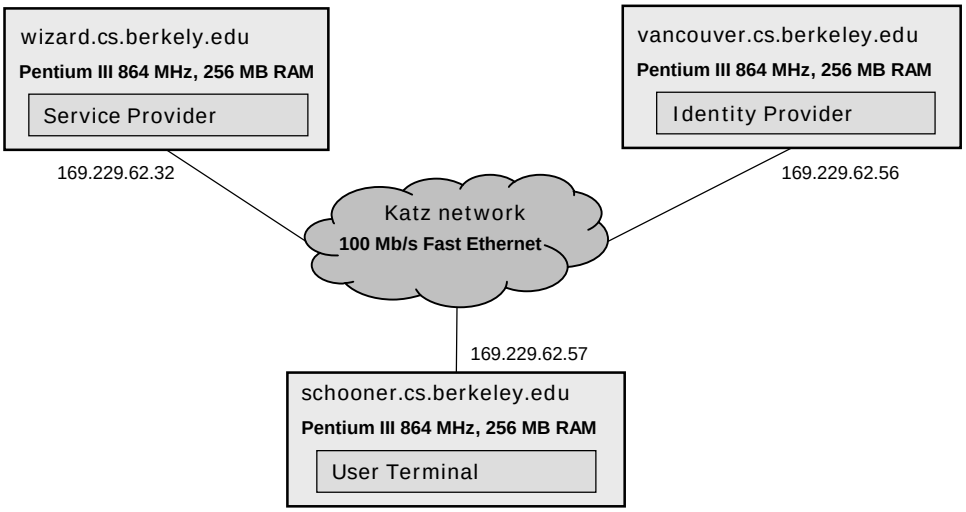


Figure 26: Authentication Latency Testbed

We have also split the delay not attributable to the link layer authentication into its main components, to better understand its root cause. Specifically, we have divided it into

authentication capabilities announcement delay, ANP client delay, policy engine delay and web authentication delay. Authentication capabilities announcement delay and web authentication delay are mainly ascribable to the server. The authentication capabilities announcement delay comprises from the moment in which the ANP client sends the authentication capabilities request to the server until it receives the authentication capabilities response from it. A minimal part of it is due to the client building the actual network packet from the Perl SOAP objects containing its information at an abstract level.

Analogously, the web authentication delay includes from the moment in which the client sends the authentication request until it receives the response from the server notifying it of the final result. In case redirection to another provider must be done, as with Liberty, this delay counts all the redirection and remote authentication process, as well as the evaluation of the remote authentication result by the service provider to decide its final authentication statement. Again, a minimal part of this delay is caused by the client building the network packets from the Perl objects containing their information.

The ANP client and policy engine delays, on the other hand, are originated in their totality by the client modules. The policy engine delay is the time spent in policy engine execution, including the parsing of the authentication capabilities statements received from the server and the selection of the authentication method, charging option and user information to submit depending on the user defined policies, preferences and context. The ANP client delay is the time spent in ANP client execution after discounting the other three delay components. It includes building the authentication capabilities request, constructing the authentication request using the information returned by the policy engine and parsing the authentication statement.

Regarding the web authentication delay, we have analyzed the performance of the two Single Sign-On authentication schemes considered in our prototype, RADIUS and Liberty, as well as that of local authentication through LDAP. For each of these authentication methods we have measured the delay in different situations. In the case of RADIUS we have considered: a) authentication of a legitimate user belonging to the service provider's administrative domain, b) authentication of a legitimate user belonging to a remote identity provider, c) authentication of a user who belongs to the service provider's domain but provides a wrong password, d) authentication of a user who claims to belong to the service provider's domain but does not really

exists there, e) authentication of a legitimate user who belongs to the service provider's domain but does not provide its domain when authenticating. In the last case, the authentication is successfully performed because the RADIUS servers are configured to assume the local domain when a user does not supply any domain. This is to make easier the input of authentication information for local users, who only need to provide their login and password. In the case of Liberty, the cases tested are the following: a) authentication of a legitimate user belonging to a remote identity provider, b) authentication of a legitimate user who belongs to a remote identity provider but provides a wrong password, c) authentication of a user who claims to belong to a remote identity provider but does not, d) authentication of a user who claims to belong to an identity provider with which the service provider does not have trust relationship. For local authentication, three cases have been tested: a) authentication of a legitimate user, b) authentication of a user who provides a wrong password, c) authentication of an inexistent user.

The validation of the link layer authentication information is successful in all of the previously enumerated test cases. Since no link layer authentication is performed during the tests, we have developed a utility to manually add a {MAC address, session key digest} pair to the server's L2InfoRepository. Before running the experiments, a fake pair is inserted into the server's repository. All the previous tests are configured to send that fake pair during web authentication. To test the effect on performance of the possible variants in link layer information validation, two additional test cases have been included: a) authentication of a legitimate local user but who provides a wrong link layer session key digest, b) authentication of a legitimate local user but who provides a wrong MAC address.

Regarding the policy engine delay, its performance depends highly on how soon a matching rule is found. Therefore, we have measured its delay for the best and worse possible cases. The best case is when the information contained in the policies, preferences and secure information files is the minimum possible to have a functional policy engine and a matching rule is found for the first {authentication method, charging option, IDP} triplet evaluated. The worse case is when the policies and preferences files are quite long and no matching rule is found for any of the evaluated triplets. In the last case, all the triplets must be compared against all the policy rules, requiring a lot of processing. As shown by the experimental results, the delay is quite low in any case, with a difference of 10 msec between the best and worse cases. We have also measured the

delay for a more standard case in which a matching triplet is found after evaluating a few of them. In fact, this is the setting enforced in all of the tests performed for evaluating the web authentication delay.

All the tests have been repeated 50 times for better accuracy. The following tables show the obtained results:

AUTHENTICATION CAPABILITIES DELAY	
Mean	**Std Deviation**
0.139	0.009

Table 2: Authentication Capabilities Delay

		WEB AUTHENTICATION DELAY	
		Mean	**Std Deviation**
LOCAL	**Right login and password**	0.166	0.011
	Right login; wrong password	0.150	0.010
	Wrong login	0.150	0.013
RADIUS	**Right login and password. Local domain**	0.150	0.009
	Right login and password. Foreign domain	0.151	0.009
	Right login; wrong password. Local domain	5.136	0.010
	Wrong login. Local domain	5.136	0.016
	Right login and password. No domain	0.147	0.008
LIBERTY	**Right login and password**	1.452	0.011
	Right login; wrong password	0.915	0.014
	Wrong login	0.918	0.019
	Wrong IDP	0.138	0.010
L2	**Wrong session key digest**	0.155	0.010
	Wrong MAC address	0.154	0.010

Table 3: Web Authentication Delay

	POLICY ENGINE DELAY	
	Mean	**Std Deviation**
Best case: First triplet matches	0.034	0.002
Worse case: No match	0.044	0.000
Standard case: Match after evaluating several triplets	0.035	0.000

Table 4: Policy Engine Delay

ANP CLIENT DELAY	
Mean	**Std Deviation**
0.008	0.000

Table 5: ANP Client Delay

We observe that the higher delay components are the authentication capabilities delay and the web authentication delay, independently of the authentication scheme employed. One of the most important contributions to the latency in those cases comes from the SSL handshake performed between client and server for establishment of a secure connection. To demonstrate this point, we have run the first test case, that is, local authentication with right login and password, without establishing the SSL connection. The results are shown in the table below. We can see how both delays are significantly reduced.

	DELAYS WITHOUT SSL	
	Mean	**Std Deviation**
AUTHENTICATION CAPABILITIES DELAY	0.048	0.007
WEB AUTHENTICATION DELAY	0.078	0.009

Table 6: Authentication Capabilities and Web Delays without SSL

The cases of unsuccessful RADIUS authentication involve an abnormally big web authentication delay. We have performed a detailed analysis of the processing time spent at the server and narrow down the cause of that high delay to the RADIUS server itself. This is an

external component to our system, and, as a consequence, the big delay is not attributable to our system design or implementation.

Comparing between the different authentication methods employed, we can observe that the delay is noticeably higher when Liberty authentication is performed. The only exception is when the user selects an identity provider with which the service provider does not have trust relationship, since no authentication can be carried out is this case. There are two reasons why Liberty authentication implies higher latency, none of them attributable to our system, but to the characteristics of the Liberty authentication scheme. First, the redirect-based approach of Liberty authentication requires more message exchanges with the user than the proxy-based RADIUS approach, as was shown in section 3, and that local authentication, where only a message in each direction is required. Since all the communication between providers and users is protected with SSL, the latency difference is appreciably increased per each of these additional exchanges. Second, upon successful user authentication at the identity provider, Liberty requires the exchange of SAML messages between the service provider and the identity provider to confirm the authentication of the user. These messages are signed and verified using public-private key cryptography, which are costly operations. Each time a message is signed or its signature verified the operation takes around 150 milliseconds.

Parsing of XML files are the main cause of delay in the policy engine processing. The policy engine needs to parse the authentication capabilities statement received from the server, as well as the user-defined policy rules and preferences. Parsing of the secure information is not required, since, as mentioned in Section 7, the parsed information is stored in memory the first time the policy engine is executed. In the same way, generation of the authentication capabilities response message does not contribute to the authentication capabilities delay, since it is only done at server initialization. This optimization is important in keeping the authentication capabilities delay low. The additional overhead of verifying the {MAC address, L2 session key digest} pair at web authentication to perform our compound authentication was smaller than 1 msec.

The following table summarizes the latency results for the success cases of the different authentication schemes employed. The five delay components analyzed, including the link layer delay, are presented. It can be observed that the total delay is below 2 seconds in the worst case, in our view an acceptable authentication latency for WLAN users.

	TOTAL DELAY		
	Local	**Proxy-based (RADIUS)**	**Redirect-based (Liberty)**
Link Layer (802.1X) Authentication	0.124		
Authentication Capabilities Announcement	0.139		
Web Authentication	0.166	0.151	1.452
Policy Engine	0.035		
ANP Client	0.008		
Total	**0.472**	**0.457**	**1.758**

Table 7: Authentication Delay Profile with Authentication Negotiation and Policy Engine

We must take into account that, apart from this delay, the user will experiment the delay due to the detection of the available wireless networks by its wireless network card and the delay due to the assignment of an IP address through DHCP. None of these delays have been considered in our results, since they are totally unrelated to our system, and would be suffered by any user connecting to a wireless network using any authentication system.

We have also studied the case where the user does not use the ANP client for authentication, but performs manual authentication using a standard web browser. The worse case delay is when the user does not access the server's web based authentication interface directly, but instead tries to access protected resources before authenticating. In this case, a firewall will redirect him to the web authentication interface. The delay in this situation can be divided into three components: link layer authentication, firewall redirection and web authentication. The link layer authentication delay is the same as with automatic authentication. The firewall redirection delay includes the detection of an unauthenticated user and his redirection to the web authentication interface using SSL. The web authentication delay is, as in the first scenario, from when the user sends his authentication data to the service provider to when the authentication is completed and the user is notified. In this case, the information is exchanged using HTML pages. Obviously, the user's time inputting the authentication information in the web authentication interface is not considered.

We can observe that the web authentication delay is larger using the Authentication Negotiation Protocol for the exchange of information than using HTML. The main reason is that the establishment of the SSL session is part of the web authentication delay in the first case, whereas it is not in the second. When the authentication is performed using the web browser, the SSL session establishment must be done when the user is redirected to the web authentication interface, so that he submits its authentication information in a secure way. Therefore, the SSL session establishment delay is part of the firewall redirection delay, and not part of the web authentication delay, when the user authenticates using the web browser.

	TOTAL DELAY		
	Local	**Proxy-based (RADIUS)**	**Redirect-based (Liberty)**
Link Layer (802.1X) Authentication	0.124		
Firewall Redirection	0.086		
Web Authentication	0.088	0.102	1.364
Total	**0.298**	**0.312**	**1.574**

Table 8: Authentication Delay Profile with Web Browser

8.3 Authentications per Second

To analyze the maximum number of authentications per second supported by our system, we need to drive the servers to their maximum load capacity. For this purpose, we added another client machine to our testbed, so that authentication requests are sent simultaneously from two clients to the same service provider. Additionally, we developed a client stub that builds a particular authentication request at initialization and then sends it multiple times in a loop, and a test program that creates multiple threads, entrusting each of them the execution of the client stub. This way, from a same machine, numerous concurrent requests can also be sent. The measurement methodology consists on sending a particular kind of authentication request a certain amount of times with concurrency, so that the server always has authentication requests waiting to be served and it is never idle, and measuring the time passed since the first request was sent until the authentication answer for the last one is received. The number of authentications per

second achieved by each client will be equal to the division of both quantities. The two client machines used have the same hardware and software characteristics, so that the results obtained by both clients when running concurrently can be directly added. To avoid propagation delays interfering with the time measurements, the two client machines are linked to the Ethernet interconnecting the servers.

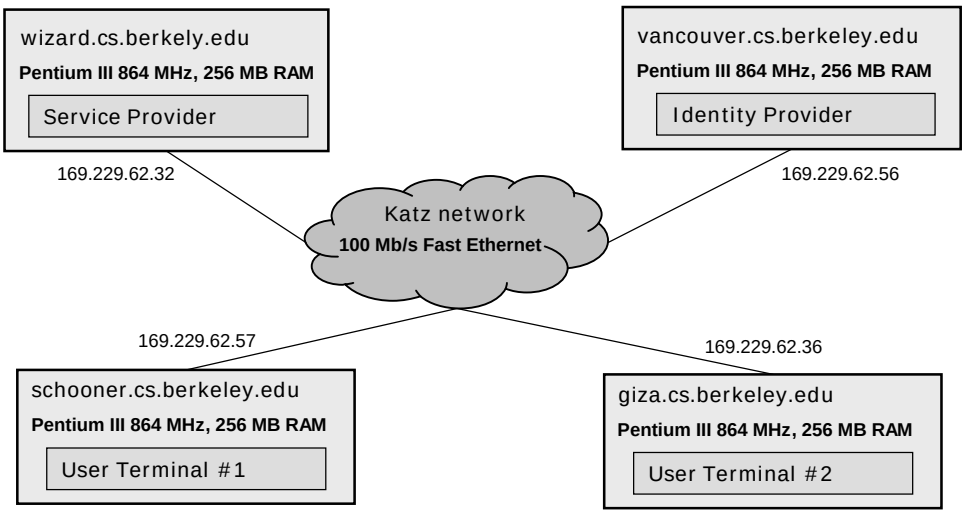


Figure 27: Load Testbed

To avoid artificial bottlenecks in the service provider, we have set the maximum number of connections per second allowed by the firewall to 10000 and the maximum number of threads permitted by the web server to 500. The test cases that we have analyzed are those of successful authentication for each of the possible authentication methods supported in our prototype, considering remote authentication for both RADIUS and Liberty Alliance. In the same way as we did when measuring the authentication latency, all the tests are configured to send certain fake {MAC address, session key digest} pair during web authentication, and this pair is manually inserted into the server's repository before running the experiments.

Our client stub supports authentication requests for the three web authentication cases previously mentioned. The test program accepts as parameters the name of the authentication method to test, the number of concurrent threads and the number of request per thread. We have run our Local and RADIUS experiments with 20 threads per client and 20 requests per thread,

and the Liberty one with one client with 15 threads and 30 requests per thread. Those numbers of threads and the fact of using only one client for Liberty authentication have been decided to prevent the service provider from running out of memory. In a real deployment of our system, each service provider should determine the maximum number of concurrent requests it can serve and fix to it the maximum number of threads allowed by its web server, so that memory problems are never encountered. Each test has been repeated 20 times for accuracy. The results obtained are showed in Table 9.

	AUTHN/SEC	
	Mean	**Std deviation**
Local	6.390	0.120
Remote RADIUS	6.592	0.177
Liberty	0.977	0.059

Table 9: Authentications per Second

It can be observed that the number of authentications per second is much lower when Liberty authentication is performed. This is mainly because of the signing and verification of SAML messages, which are processing consuming operations and must be executed a couple of times per Liberty authentication. The number of remote RADIUS authentications per second is slightly higher than the number of local authentications per second because the actual verification of the {login, password} pair is done at the remote identity provider, saving some processing time. In all cases the number of authentications per second is low, since even local or RADIUS authentication are quite processing consuming due to establishment of SSL secured connections. Nevertheless, it is high enough for the expected rate of incoming users into a specific network in the WLAN world, where mobility is lower than in cellular networks.

9 Future work

In the development of our prototype we have focused in its functionality. However, its usability has been left aside in some occasions. Before the system is deployed in a real environment we recommend the development of some graphical user interfaces to facilitate the management of user information. In particular, a GUI to handle the secure information would be desirable. This GUI would take care, among other tasks, of encrypting and decrypting the user information, guaranteeing that it is always protected before being stored. Graphical user interfaces for management of policy rules and preferences would also be useful. Additionally, the GUI for manual selection of identity provider, authentication method and charging option is not implemented yet. A first version was developed in the initial stages of the prototype. However, the client code was later upgraded, but not the GUI, which became useless due to its high dependency on the data structures used to store the user information at the client.

Another area for future work is the incorporation of the Authentication Negotiation Protocol in the interaction between client and identity provider, so that the latter can announce its authentication capabilities to the client and user authentication can be performed in an automated and flexible fashion, just as it is done between client and service provider. At the moment, standard HTML forms send over HTTP are used by IDPs for user authentication. We assume the names of the required fields to be fixed to certain values, and the ANP client always submits the user information using those field names. This approach has he limitation that automated authentication using the ANP client will only be successful as long as the identity provider adjusts to the expected format. As soon as one of the required fields changes its name or a new field is required, the current automated authentication will fail. The Authentication Negotiation Protocol is flexible enough to accommodate the new usage model without requiring any modifications to the currently supported message types.

Client extension to incorporate evaluation of the service provider trustworthiness is also an important element to be considered in further improvements of the system. Two options are

proposed for it. The first one is the use of nonces in authentication capabilities request messages, so that servers return signed capabilities statements. The policy engine should try to verify the authentication capabilities statement's XML signature using the public key of the service provider and, if successful, check that the replied nonce matches the nonce originally sent in the request to avoid replay attacks. The second option is determining whether the service provider is trusted or not at the establishment of the SSL connection with it. The secure SSL connection should always be created, independently of whether the server's certificate can be successfully verified or not, but the authentication result should be notified to the policy engine, so that it acts accordingly.

We also propose the extension of the system to support periodic automatic reauthentications when the user desires to keep his connection alive for a long time. The reauthentication process could be initiated by the client or the server. In the first case, the implementation is much easier, since basically the only modification required is the usage of a reauthentication timer at the client. The server driven approach, on the other side, requires that the server keeps track of the session length for all of the clients connected at a given time. It also involves the definition of a new notification message, which would be sent by the server to the clients to warn them when their session is about to expire.

Finally, there is much room for improvement regarding the accounting part of the system. We have focus of authentication, and, although some support for charging has already been implemented, most of that functionality has been postponed for future versions of the system. The Authentication Negotiation Protocol, however, is already prepared for carrying all the charging related information.

10 Conclusion

Dynamic selection of the authentication method and the identity provider is essential for enabling the confederation of public wireless LAN service providers under different trust levels and with alternative authentication schemes. Our proposed authentication adaptation framework accommodates multiple authentication methods. To demonstrate our approach, we confederated public wireless LANs using two industry-standard single sign-on authentication schemes: RADIUS and Liberty Architecture. A client-side policy engine enables the user to automatically select which of the alternate single sign-on authentication schemes to use. The policy engine also protects the user‘s sensitive information by forcing him to manually input his authentication information when he roams into a service provider he weakly trusts. In addition, we developed a compound Layer 2 and Web authentication scheme to prevent theft of service, eavesdropping, and message alteration in public wireless LANs. To demonstrate the feasibility of our approach, we developed a single sign-on prototype system. The measured authentication delay values ranged from 0.457 to 1.758 seconds depending on the authentication method employed. They are small enough for practical use.

11 Acknowledgments

I would like to thank my research advisor, Prof. Randy Katz, for his support and wise advice during the execution of my master's project. I would also like to thank Prof. David Wagner for valuable suggestions on trust relationship, Yasuhiko Matsunaga and Takashi Suzuki for their ideas, Nick Neely for his important help on implementing the client code and Fred Archibald for giving us access to the wireless equipment.

12 References

[1] HotSpotList.com, http://www.hotspotlist.com/

[2] IETF, RFC 2865 “Remote Authentication Dial In User Service (RADIUS)”, June 2000.

[3] Liberty Alliance Project, “Liberty ID-FF Architecture Overview”, Version 1.2, November 2003.

[4] Wi-Fi Alliance, “Best Current Practices for Wireless Internet Service Provider (WISP) Roaming”, ver. 1.0, 2003.

[5] S. Hada and M. Kudo, “Access Control Model with Provisional Actions”, IEICE Trans. Fundamentals, Vol. E84-A, No.1, Jan. 2001.

[6] OASIS, “eXtensible Access Control Markup Language (XACML)”, Version 1.0, February 2003.

[7] IEEE Std 802.1X-2001, “Port-Based Network Access Control”, June 2001.

[8] IEEE Std 802.11i/D7.0, ”Medium Access Control (MAC) Security Enhancements”, October 2003.

[9] IETF, RFC 2716, “PPP EAP TLS Authentication Protocol”, Oct. 1999.

[10] Internet-Draft, “EAP Tunneled TLS Authentication Protocol”, draft-ietf-pppext-eap-ttls-03.txt, work in progress.

[11] IETF RFC 2402, “IP Authentication Header”, Nov. 1998.

[12] D. Jablon, “Strong Password-Only Authenticated Key Exchange”, Computer Communication Review, Vol.26, 1996.

[13] http://srp.stanford.edu/

[14] V. Bahl, A. Balachandran, S. Venkatachary, “The CHOICE Network: Broadband Wireless Internet Access In Public Places”, Microsoft Technical Report, MSR-TR-2000-21, Feb. 2000.

[15] OASIS, “Assertions and Protocol for the OASIS Assertion Markup Language (SAML)”, Committee Specification 01, May 2002.

[16] http://www.open1x.org/

[17] N. C-Winget, R. Housley, D. Wagner, J. Walker, “Security flaws in 802.11 data link protocols”, Communications of the ACM, 46(5), May 2003, pp. 35-39

[18] J. Bellardo and S. Savage, “802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions”, Proceedings of the USENIX Security Symposium, August 2003.

[19] IETF, RFC2759 “Microsoft PPP CHAP Extensions, Version 2”, Jan. 2000.

[20] B. Schneier, “Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)”, Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.

[21] R. Angal, C. Kaler et al., “Web Single Sign-On Metadata Exchange Protocol”, MSDN Library, April 2005.

[22] R. Angal, C. Kaler et al., “Web Single Sign-On Interoperability Profile”, MSDN Library, April 2005.

[23] E. Griffith, “802.11i Security Specification”, Wi-Fi Planet, June 25 2004.

[24] P. Galli, “Microsoft, Sun Update Their Technical Cooperation Work”, eWeek.com, May 13 2005.