

# Secure Authentication System for Public WLAN Roaming

Yasuhiko Matsunaga<sup>‡</sup>   Ana Sanz Merino<sup>†</sup>   Manish Shah<sup>†</sup>   Takashi Suzuki<sup>††</sup>   Randy H. Katz<sup>†</sup>  
Computer Science Division   Computer Science Division   Computer Science Division   Multimedia Laboratories   Computer Science Division  
Univ. of California, Berkeley   Univ. of California,   Univ. of California, Berkeley   NTT DoCoMo, Inc.   Univ. of California, Berkeley  
Berkeley, CA, U.S.A.   Berkeley   Berkeley, CA, U.S.A.   Yokosuka,   Berkeley, CA, U.S.A.  
Berkeley, CA, U.S.A.   Kanagawa, Japan

<sup>†</sup>{asanz, manish, randy}@eecs.berkeley.edu   <sup>‡</sup>y-matsunaga@bl.jp.nec.com   <sup>††</sup>suzuki@spg.yrp.nttdocomo.co.jp

## ABSTRACT

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 select the appropriate SSO authentication scheme from the authentication capabilities announced by the WLAN service provider, and can block the exposure of their privacy 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 about 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.

## Keywords

Wireless LAN, roaming, authentication, policy control, link layer security.

## 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, 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, access 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 services 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; 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 provider 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 the *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 offer users the possibility to choose between the available alternatives. It is fundamental to do it in a standard way as to allow automatic processing of the information at the client side. For this purpose we have defined 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, together with a graphical user interface to present the server's authentication capabilities information to users and allow them to select the authentication method in a user friendly way.

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 client, which can request the policy engine to determine the information to be sent to the authentication negotiation 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 also can be used independently, as it provides a generic API for authentication information access. 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 turns on his PDA in his office and is automatically connected to his department's WLAN. The department WLAN uses 802.1X authentication, and the policy engine permits automatic submission of the pre-shared secret between him and the department WLAN.

Then he walks out of the building with his PDA, thus 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 launches a graphical user interface on the user terminal and asks him 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 290 msec, 255 msec and 120 msec, respectively.

The rest of this paper 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 describes the prototype system and our evaluation results. Finally, Section 8 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] is being standardized to provide 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.

### **3. SINGLE SIGN-ON CONFEDERATION MODEL**

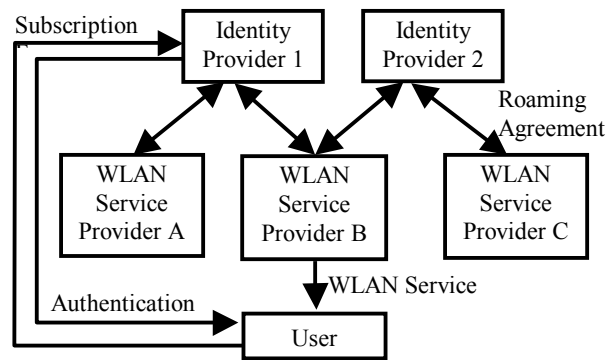
We use single sign-on (SSO) authentication technologies to confederate WLAN service providers via trusted identity providers and we accommodate multiple underlying authentication methods. In this chapter we describe our confederation model and the SSO and authentication technologies selected for our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

- There exists at least one certificate authority trusted by users, service providers, and identity providers.
- 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 Fig. 1, we assume that there is at least one and possibly several identity providers with which 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 Fig.1, 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.

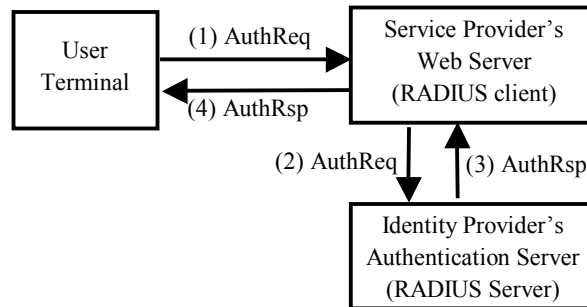


**Fig. 1: Roaming Model**

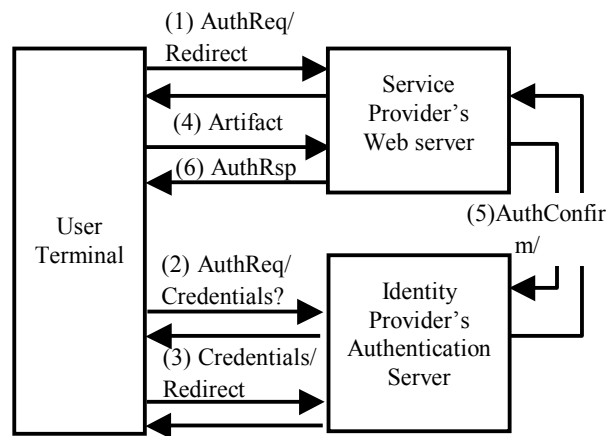
We focus on developing a decentralized authentication framework for inter-service provider roaming where each provider adopts its own authentication methods, with arbitrary roaming agreement relationships between identity providers and WLAN service providers. Our architecture is independent of the authentication 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 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). Liberty-based roaming has the advantage of hiding the user's identity and credentials from weakly-trusted WLAN service providers. 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. Fig. 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.

On the other hand, the Liberty Browser Artifact Profile makes use of a redirect-based authentication model (Fig. 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 credential from untrusted 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.



**Fig. 2: Proxy-based RADIUS Authentication**



**Fig. 3: Redirect-based Liberty Browser Artifact Profile Authentication**

## 4. AUTHENTICATION FLOW ADAPTATION FRAMEWORK

### 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.

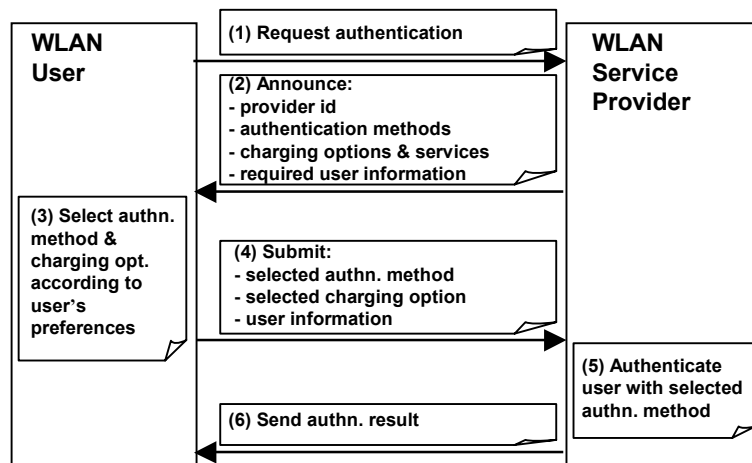
To facilitate roaming, it is useful if the communication of authentication capabilities and the user's selection are accomplished with minimal user intervention. The user should avoid having to make use of a web browser to exchange this information. We have designed a new XML web-based protocol, the *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. As part of this client, we have developed a simple graphical user interface that presents the server's authentication capabilities to the user while allowing users to make their selections. The details of the protocol



are defined in Section 4.2. The internals of the authentication negotiation client with its associated graphical user interface are described 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.



**Fig. 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.

On the server side, analogously to the client side, an 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. The web server is able to detect 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.

The following picture shows the main components of our authentication flow adaptation architecture. 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 will be launched by the link layer roaming client, and will automatically initiate the negotiation with the server. 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.

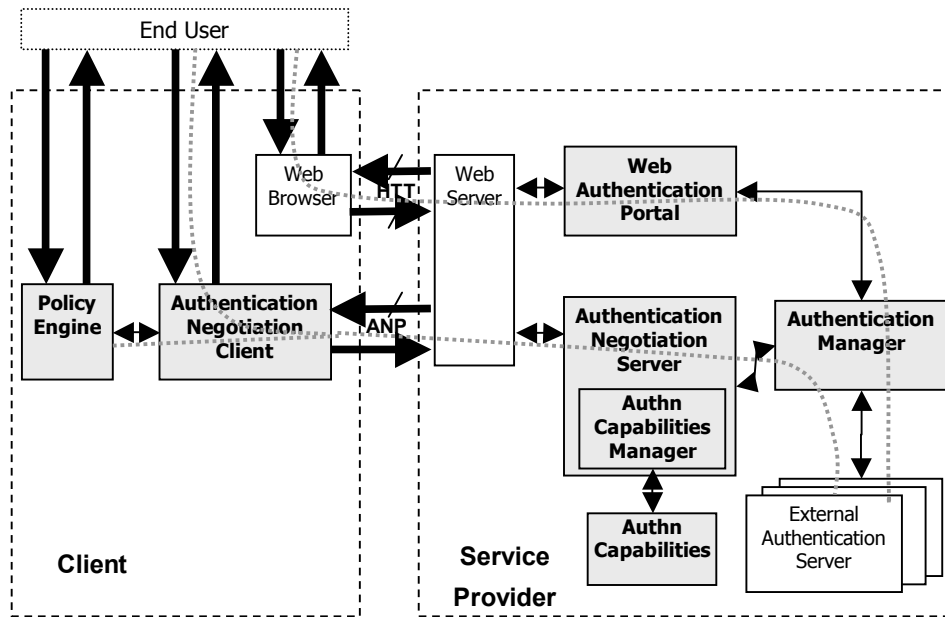


Fig. 5: Authentication Flow Adaptation Architecture

### 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). The content of 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 for authentication messages involving web browsers.

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.
- 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.

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 *AuthnMethodID* 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 39.95 USD per month. 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="1900-01-01T00:00:00Z">
  <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>39.95</anp:UnitPrice>
      <anp:TimeUnit Unit="Month">
        <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>

```

**Fig. 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. 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>
```

**Fig. 7: Authentication Query Example**

- Authentication Statement: used by servers to communicate the result of the authentication to the user.

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

#### 4.4 Client Graphical User Interface

The user interface is a graphical display of the options offered by a particular service provider. These are extracted from the authentication capabilities statement sent by the service provider in Step 2 of the authentication negotiation protocol. It groups the information into categories of identity providers for which the user does or does not carry accounts. Upon selecting one from either the accounts or no accounts boxes, the user interface displays the charging options available for the highlighted identity provider. Upon selecting from the list of charging options, the user interface displays the specific information for that particular charging option. The authentication methods and the required information for each are displayed in a similar fashion. Above each box where a selection is made, the currently selected option is displayed. The 'OK' button chooses the options selected and then prompts the user if he would like to save these for next time. The 'Cancel' button simply exits the user interface. As of now 'Online SignUp' button simply prompts the user to choose the cancel button and open a standard web browser. Lastly, the 'Properties' button is used for displaying the options chosen last time with this service provider and allows the user to modify them. These options can also be modified by the user by launching the user interface program offline.

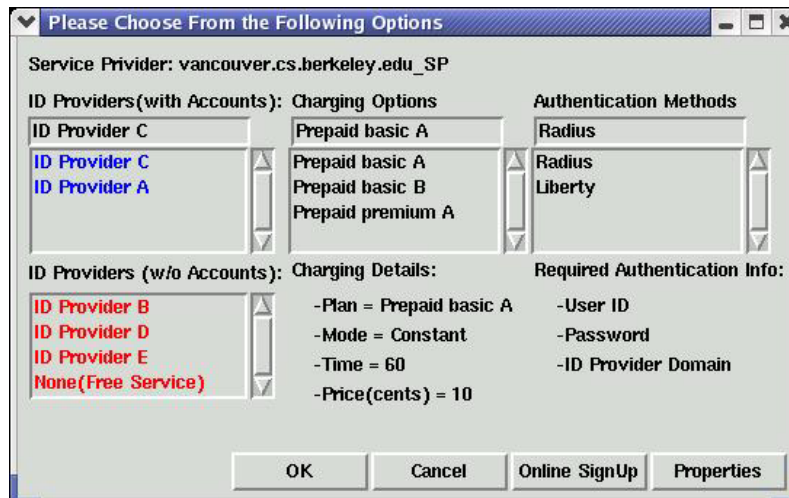


Fig. 8: Graphical User Interface

## 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. In the following subsection, we describe the policy engine component blocks.

### 5.1 Overview

In federated public WLANs, a user roams across networks operated by different providers. Because 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. As output, it returns an XML file formatted according to the XML schema definition for the authentication query of the Authentication Negotiation Protocol containing the requested information.

An XML-based access control language is used to define access control rules for the user information. Examples of information elements and rules are shown in Fig. 9. The policy rules specify entities to be authorized to access authentication information elements using the entities identifiers or attributes (roles). The authentication information elements are specified with XML *authn\_info* elements, and the entities, with *subject* elements. The policy rules may include provisional actions to be fulfilled such as user notification and acknowledgement. The provisional actions are indicated with *provisional\_action* elements. In case the user terminal always demands user input or acknowledgment, it is the end user who makes the access control decision according to his security policy and context. The policies may also 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 are stored in the elements *chosen\_idp*, *chosen\_charging\_option* and *chosen\_auth\_method*. Regarding the authentication capabilities, these are organized based on the available identities providers. These can be split into two groups, those with which the user has accounts and those with which he does not. Each available identity provider with which the user has an account, together with its supported authentication and charging options, is stored in a *Have\_accounts\_with* element. Each available identity provider with which the user does not have an account, and the authentication and charging options supported for it, is stored in a *Do\_not\_have\_accounts\_with* element. When first logging into a service provider's network with authentication capabilities announcement functionality, the user will specify which options they want. After having chosen which options they want, they will be asked whether or not that service provider is to be trusted and they want to save those options for future automated selection. If so, the policy engine will save the current options to a service provider specific file indicating that the user will allow automatic sign on when once again roaming into this service provider's network. For subsequent times that the user roams into that provider, the policy engine will compare the saved options from the last sign on to this service provider and the options currently available. If the previous options match the current options, then the policy engine will proceed with the automatic selection. However, if the options are mismatched, the policy engine alerts the user and the user can use the user interface to select new options.

### Policy Rule

```

<policy>
  <rule>
    <authn_info href="..." />
    <authn_info href="..." />
    <subject>
      <id> ... </id>
      <attribute> ... </attribute>
    </subject>
    <provisional_action name="..." />
  </rule>
  <options>
    <chosen_idp>...</chosen_idp>
    <chosen_charging_option>...</chosen_charging_option>
    <chosen_auth_method>...</chosen_auth_method>
    <Use_Next_Time>...</Use_Next_Time>
    <Last_Update_Time>...</Last_Update_Time>
    <Have accounts with>
      <IDP_Name> ... </IDP_Name>
      <Charging_Option>...</Charging_Option>
      <Auth_Method>...</Auth_Method>
    </Have accounts with>
    <Do not have accounts with>
      <IDP_Name>...</IDP_Name>
      <Charging_Option>...</Charging_Option>
      <Auth_Method>...</Auth_Method>
    </Do not have accounts with>
  </options>
</policy>

```

### Information Element Examples

Authn_info	Authentication information file path <ul style="list-style-type: none"> <li>login ID, password</li> <li>credit card type, card number, name, expiration date</li> </ul>
Subject	id: provider identifier attribute: <ul style="list-style-type: none"> <li>role: <ul style="list-style-type: none"> <li>identity provider</li> <li>service provider</li> </ul> </li> </ul>
Provisional_action	<ul style="list-style-type: none"> <li>user notification</li> <li>user acknowledgment</li> <li>user input</li> </ul>
Options	<ul style="list-style-type: none"> <li>authentication methods</li> <li>charging characteristics</li> </ul>

**Fig. 9: Policy Rule and Information Element Examples**

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 Component Blocks

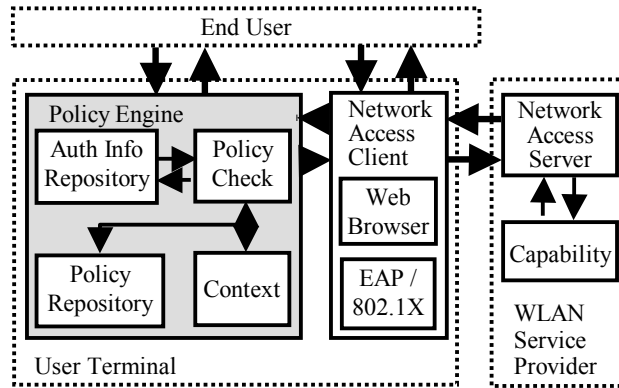
The policy engine major components are shown in Fig.10. The policy check component performs a policy compliance check based on the input parameters and user-specified access control rules, and outputs granted information along with any provisional actions. The provisional action is what must be performed before the corresponding information is sent to the information requestor. In our current implementation, we support two: user authorization and user manual input. For example, if the access control component returns user identity along with the “authorization” provisional action, the policy engine launches a new pop-up window to give the option to send the identity to the server. The identity can be sent out only if the user selects the “accept” button. If the access control component returns granted information without provisional action, the information is automatically sent to the server. The policy check component is divided into three subcomponents: root component, secure component, and specific component.

The root component is the only component that has direct interaction with the accessor and is where the XML file is input. Once given a properly formatted XML file, the root component does some initial parsing to gather information before it proceeds. It is mainly checking the service provider's identity (and verifying the service provider). Once it has the name of the service provider (SP), it executes a “decision tree.” First it checks if a service provider specific policy exists for this SP. If so, it interacts with the specific component to verify if the user desired the stored options (saved as context information) to be automatically evoked when roaming into this service provider's network. If this is true it compares that saved information against the options announced to verify if these options are still valid (e.g., identity providers, price identifiers, rates, and authentication methods are still available). If this is true, then the root element interacts with the secure component and creates an XML file that contains the options chosen for this SP and the required information for these options (from the secure component). If any of the decision questions returned false, the root element notifies the authentication negotiation client. It then invokes its graphical user interface so the user can choose options manually and decide whether to save these for next time.

The specific component contains service provider (SP) specific rules. It stores the options the user chose for this SP when last signed on to the SP's network. It also stores whether the user wants these options to be used when roaming back into this SP's network (i.e., automatic sign on). Next it stores information from the announcement that provided these options. This is done so that the user can change the saved options offline and use them instead for automatic sign on. The specific rules for a SP can only exist if the user has previously signed on to this network using the manual user interface.

The secure component stores the critical information for the user to be authenticated anywhere. It stores username, password, and identity provider for each account the user holds. It also stores the user's preferred credit card for each account and accordingly stores the

credit card specific information. This component is only read by the root component and is thus hidden from the SP. It is only accessed if the criteria are met in the root component. To protect the stored information from unauthorized access, this information must be cryptographically protected.



**Fig. 10: Policy Engine Block Diagram**

## 6. SECURING WEB-BASED AUTHENTICATION AND ACCESS CONTROL

### 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.

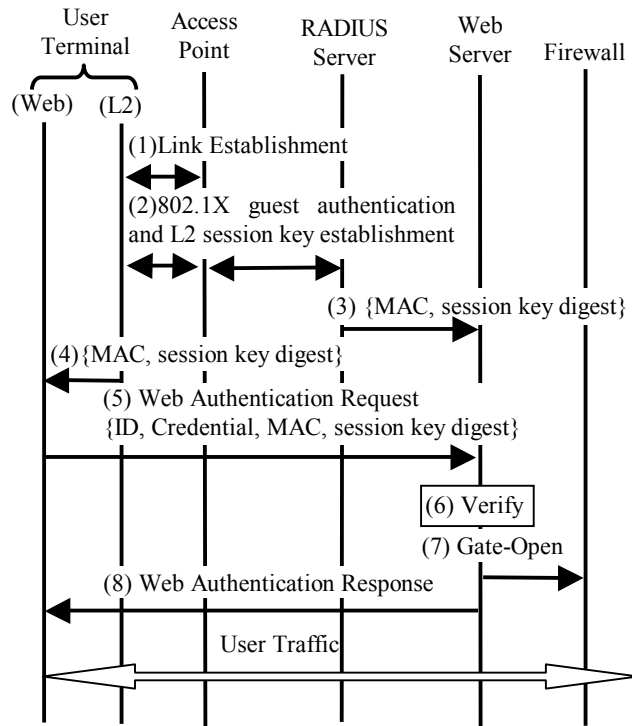
The key to avoiding these security threats is to have a cryptographic binding between the user and the network. As explained in Section 2, IEEE 802.1X port-based network access control is being deployed in corporate wireless LANs, and it uses such a cryptographic method for user authentication and network 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.

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

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

## 6.2 Compound Layer 2 and Web Authentication

To ensure cryptographically protected access in public wireless LANs, we have developed a compound Layer 2 and Web 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.



**Fig. 11: Compound L2 and Web Authentication Message Sequence**

The compound authentication message sequence diagram is shown in Fig. 11. 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 can't 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. After Step 2, the encryption key derived from the L2 session key encrypts packets transmitted over the air. The RADIUS 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). The inter-process communication in Step 4 can be accomplished by using a customized web client or the policy engine described in the previous section. Then the user terminal sends a Web authentication message to the Web server, with {ID, credential, 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. Finally, the Web server verifies the quadruplets (Step 6), changes firewall rules so that the user can access to the external network if the verification succeeds (Step 7), and returns an authentication response to the user terminal (Step 8).

### 6.3 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. 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.

### 7.1 Testbed

Our testbed consists of five authentication servers, a WLAN access point, and two user terminals. Two of the authentication servers act as service providers and other two act as identity providers, and the last one acts as an 802.1X authentication server. To avoid wireless-specific delay variance, link layer authentication delay and other delay were measured separately. Each server was connected via 100Mb/s Fast Ethernet and link delay was minimal. All of them are implemented on standard Linux or Windows PCs using open-source software.

The hardware specifications of components in the testbed are listed below.

- Identity Provider #1: Pentium III 864 MHz (Linux)
- Identity Provider #2: Pentium II 266 MHz (Linux)
- Service Provider #1: Pentium III 864 MHz (Linux)
- Service Provider #2: Pentium II 266 MHz (Linux)
- User Terminal #1: Pentium IV 1.6 GHz (Windows XP)
- User Terminal #2: Pentium III 1.1GHz (Linux)
- 802.1X Authentication Server: Pentium III 1.1GHz (Linux)
- WLAN Access Point: Cisco AIR-AP352

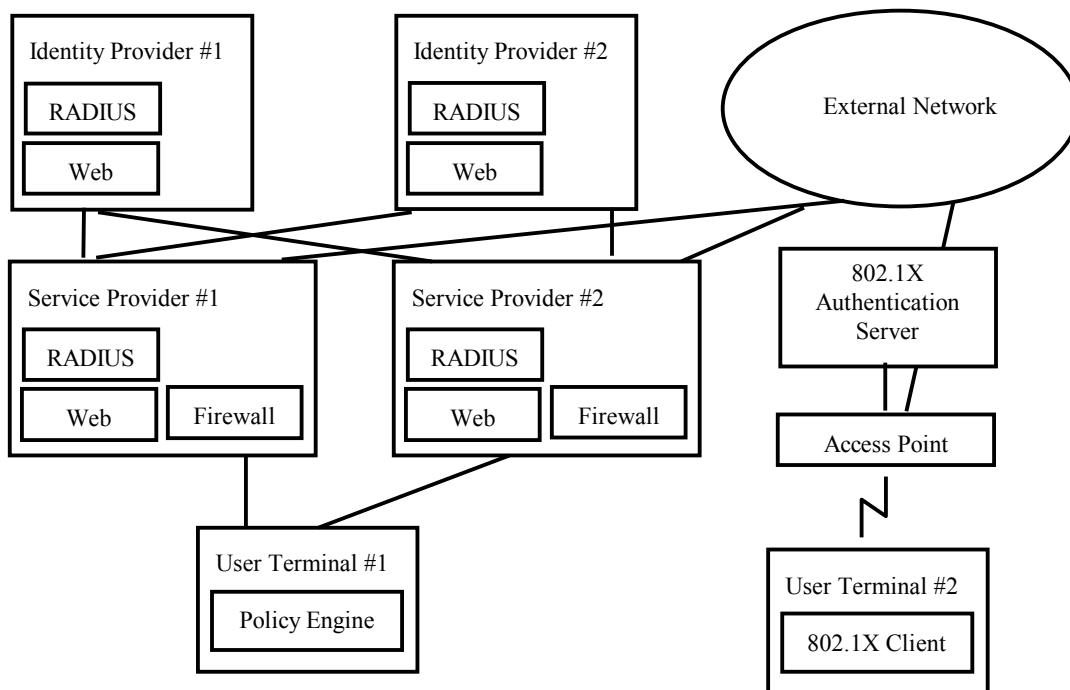


Fig. 12: Testbed Schematics



The open-source software components used in our testbed are as follows.

- GNU RADIUS v0.96.4 (ID/password authentication server)
- FreeRADIUS v0.8.1 (802.1X authentication server)
- XSupplicant v0.6 (802.1X client)
- ForgeNet RADIUS Client 0.9c
- Sun Interoperability Liberty prototype v0.1
- OpenLDAP LDAP Server v2.1.12
- iptables v1.2.6a (Firewall)
- libwww-perl 5.64 (Web client)

## **7.2 Authentication Latency**

To analyze the performance of our solution, we have measured the authentication delay for each of the components participating in the authentication process, and this for the following two possible alternatives in the client side: 1) the user terminal has an authentication negotiation client and a policy engine, 2) the user terminal does not have these components and a web browser is required to perform the authentication. In addition, we have analyzed the performance of the two authentication schemes considered in our prototype, evaluating both of them in two different authentication scenarios: local and remote. In the former, the service provider also plays the role of identity provider, directly authenticating the user. In the latter, a third party plays this role. No transmission delays have been included, as they vary with the distance between entities and are not directly attributable to our implementation. The following table shows the results obtained for roaming authentication using the authentication negotiation client to request the authentication capabilities information to the server and the policy engine to submit the authentication data. The total delay was about 2 seconds in the worst case, in our view an acceptable authentication latency for WLAN users.

The delay can be divided in four components: link layer authentication, authentication capabilities announcement, policy engine and web authentication. The link layer authentication delay is due to the establishment of a secure Layer 2 communication using the IEEE 802.1X protocol. The authentication capabilities announcement delay is from when the authentication negotiation client is invoked to request the authentication capabilities from the server until it finishes parsing the authentication capabilities it received and launches either the graphical user interface for manual authentication or the policy engine for automatic authentication. The policy engine delay comprises the selection in the client side of the authentication method to use and the information to submit depending on the user defined policies and the communication context, including the authentication capabilities information received from the server. Finally, the web authentication delay is 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 first alternative, it is the authentication negotiation client that sends the authentication data to the server and receives the authentication result using the Authentication Negotiation Protocol. The most significant delay in the web authentication segment is the Liberty remote case. This is due to the exchange of SAML messages between a service provider and an identity provider

to confirm the authentication of the user, which are signed and verified using public-private key cryptography. This is not needed when the authentication is local, since the same entity is both service and identity provider.

Parsing of XML files are the main cause of the delays in the authentication capabilities announcement and the policy engine processing. In the latter case, the client parses the user-defined policy rules. In the former, the client parses the authentication capabilities statement received from the server. The generation of the authentication capabilities information in the server usually takes almost no time. Since the server authentication capabilities do not change often, the authentication capabilities information is generated at server start-up and kept in memory. This information will only be updated when a change in the authentication capabilities occurs.

The link layer (802.1X) authentication delay was as small as 0.12 sec, and the additional overhead of verifying {MAC, L2 key digest} at web authentication was smaller than 1 msec.

**Table 2: Authentication Delay Profile with Authentication Negotiation and Policy Engine**

	Proxy-based (RADIUS)		Redirect-based (Liberty)	
	Local	Remote	Local	Remote
<b>Web Authentication</b>	0.295	0.296	0.276	1.545
<b>Policy Engine</b>	0.255			
<b>Authentication Capabilities Announcement</b>	0.250			
<b>Link Layer (802.1X) Authentication</b>	0.124			
<b>Total</b>	0.924	0.925	0.905	2.174

In the case that the user does not use the authentication adaptation client to request the authentication capabilities information to the server, he will authenticate by opening a web browser and accessing the server web based authentication interface. The worse case delay is when the user does not access this authentication interface directly, but instead tries to access protected resources before authenticating. In this case, a firewall will redirect him to the server 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 above. 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. 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. One of the reasons 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. The other reason is the currently suboptimal generation and processing of authentication negotiation messages on the client side. This should be improved in future versions of the client.

**Table 3: Authentication Delay Profile with Web Browser**

	Proxy-based (RADIUS)		Redirect-based (Liberty)	
	Local	Remote	Local	Remote
<b>Web Authentication</b>	0.091	0.102	0.088	1.364
<b>Firewall Redirection</b>	0.086			
<b>Link Layer (802.1X) Authentication</b>	0.124			
<b>Total</b>	0.301	0.312	0.298	1.574

## 8. 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 select which of the alternate single sign-on authentication schemes to use. The policy engine also protects the user's privacy information by forcing him to input 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 0.9 to 2.2 sec depending on the authentication types. They are small enough for practical use.

## 9. ACKNOWLEDGMENTS

The authors would like to thank David Wagner for valuable suggestions on trust relationship. Yasuhiko Matsunaga was supported by NEC Corporation, as a visiting industrial fellow at University of California, Berkeley, until December 2003. Takashi Suzuki was

supported by NTT DoCoMo, as a visiting industrial fellow at University of California, Berkeley, until May 2003. Funding for this work was provided in part by the California MICRO Program, with matching industrial support from Sprint and Ericsson.

## 10. 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.

## 11. AUTHORS

### **Yasuhiko Matsunaga**

Affiliation: Computer Science Division. University of California, Berkeley, USA.

E-mail Address: y-matsunaga@bl.jp.nec.com

Postal Address: 475 Soda Hall, Berkeley, CA 94720-1776, USA.

### **Ana Sanz Merino**

Affiliation: Computer Science Division. University of California, Berkeley, USA.

E-mail Address: asanz@eecs.berkeley.edu

Postal Address: 473 Soda Hall, Berkeley, CA 94720-1776, USA.

### **Manish Shah**

Affiliation: Computer Science Division. University of California, Berkeley, USA.

E-mail Address: mshah@uclink.berkeley.edu

Postal Address: 473 Soda Hall, Berkeley, CA 94720-1776, USA.

### **Takashi Suzuki**

Affiliation: Multimedia Laboratories. NTT DoCoMo, Inc. Yokosuka, Kanagawa, Japan

E-mail Address: suzuki@spg.yrp.nttdocomo.co.jp

Postal Address: 3-5 Hikarinooka, Yokosuka, Kanagawa, 239-8536, Japan.

### **Randy Katz**

Affiliation: Computer Science Division. University of California, Berkeley, USA.

E-mail Address: randy@eecs.berkeley.edu

Postal Address: 637 Soda Hall, Berkeley, CA 94720-1776, USA.