6-3-2004

Greenpass RADIUS Tools for Delegated Authorization in Wireless Networks

Sung Hoon Kim
Dartmouth College

Greenpass RADIUS Tools for Delegated Authorization in Wireless Networks

Dartmouth College Computer Science Technical Report TR 2004-510

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Master of Science

in

Computer Science

by

Sung Hoon Kim

DARTMOUTH COLLEGE

Hanover, New Hampshire

June 3, 2004

Examining Committee:

_____
Sean Smith (chair)

_____
Edward Feustel

_____
Christopher Hawblitzel

_____
Carol L. Folt
Dean of Graduate Studies

Abstract*

Dartmouth's Greenpass project extends how public key cryptography can be used to secure the wireless LAN with a RADIUS (Remote Authentication Dial In User Service) server that is responsible for handling authentication requests from clients (called supplicants in the 802.1x authentication model). This thesis describes the design and implementation of the authentication process of Greenpass, specifically what decisions are made in determining who is granted access and how a small modification of already existing protocols can be used to provide guest access in a way that better reflects how delegation of authority works in the real world.

Greenpass takes advantage of the existing PKI to authenticate local Dartmouth users via X.509 identity certificates using EAP-TLS. We use the flexibility of SPKI/SDSI (Simple Public Key Infrastructure/Simple Distributed Security Infrastructure) authorization certificates to distribute the responsibility of delegating access to guests to certain authorized delegators, avoiding some of the necessary steps and paperwork associated with having a large centralized entity responsible for the entire institution. This thesis also discusses how our solution can be adapted to support different methods of guest delegation and investigates the possibility of eliminating the cumbersome central entity and administrative overhead traditionally associated with public key cryptography.

* This thesis is based in part on our preliminary reports on the Greenpass project [SGK+04, GKS+04]. Concurrent theses [Gof04, Pow04] explore other aspects of the project.

Acknowledgments

There are a lot of people I would like to thank for aiding me in this endeavor. I would like to start by thanking the other members of the Greenpass team, starting with my advisor, Professor Sean Smith, for spearheading this project and providing valuable advice and guidance that was indispensable to my efforts. I also would like to thank Nicholas Goffee for his outstanding work on the client-side tools and in developing the other half of the Greenpass system, Punch Taylor for his help and guidance in setting up our network components, Meiyuan Zhao for her work with the SDSI library, Kimberly Powell for her organization of a pilot program to test our solution, and Kwang-Hyun Baek for his research in the recent trends in wireless network authentication. Many thanks also go out to Eileen Ye for her initial investigation of SPKI/SDSI and to Stephen Campbell for setting up our Cisco equipment for the revised model.

I also would like to thank the other members of my thesis committee, Professor Edward Feustel and Professor Chris Hawblitzel, for providing valuable input to my thesis and bring to my attention important issues that needed to be addressed.

I would like to especially thank the Cisco Corporation for providing funding and networking equipment used in our project, and in particular Graham Holmes and Krishna Sankar for coordinating our efforts with Cisco.

This research has been supported in part by Cisco Corporation, the Mellon Foundation, NSF (CCR-0209144), AT&T/Internet2 and the Office for Domestic Preparedness, Department of Homeland Security (2000-DT-CX-K001). This paper does not necessarily reflect the views of the sponsors.

Contents

List of Figures

1. Introduction

1.1 The Problem

Wireless networking is the newest hot topic in networking. As the use of wireless networks becomes more ubiquitous, it is increasingly important to ensure that the wireless network is kept secure. At Dartmouth College, where wireless network access is available across the entire campus, we are developing *Greenpass*, a software-based solution to secure wireless networks that also allows delegation of access to guest users.

Greenpass is designed to enforce an unobtrusive authentication process for registered users (i.e., Dartmouth faculty, students, and staff) and to provide a seamless and decentralized mechanism for non-registered users (guests) to gain access. Once authorized, guests are allowed access to the same access points and resources that local users are given. For example, guest access is not necessarily restricted to localized physical areas (such as a conference room) like in some commercial solutions that are available. Furthermore, the process is decentralized in that pre-selected individuals can delegate temporary access to guests without referring to a central authority. Once the delegator creates the guest's credentials, the guest can immediately get access to the network.

1.2 Motivation

Wireless network access is already prevalent at Dartmouth and is becoming more available on college campuses. With the increased availability of wireless networks, there are several issues that need to be addressed in ensuring that access to the wireless network is at least as secure as access to the wired network. Authorized users must be granted access to the network with little or no additional effort required by the authentication process. Guests must be granted access to the network and the delegation process must be hassle-free and decentralized. The solution must be adaptable: it must scale to large environments, accept a wide variety of access policies, multiple client platforms must be supported, and it must extend to all authorization. Furthermore, the solution needs to be robust enough to defend against a wide range of failures and attacks.

Greenpass adheres to a different definition of guest access than other wireless security solutions do. Guests must be approved by a delegator to qualify for guest access; we are not letting just anyone gain access to the network. We do not want to impose physical restrictions on where guests can get access to the network; guests are not confined to special access points that connect to outside a firewall. Guests are not shuttled outside the firewall, thus preventing access to the interior network. Guests should get access to the inside, as long as they are authorized to do so. With these factors in mind, we sought to create a decentralized solution in which guests can gain meaningful guest access through a user-friendly delegation process.

1.3 My Contribution

The Greenpass project is divided into two key components: (1) the decision process used by an authentication server* in granting or rejecting access and (2) client tools that implement delegation of guest access. My work on the project was focused on the first component, specifically configuring and modifying the RADIUS server that is used to make authentication decisions. My task was to deal with maintaining the server-side components of the system.

Over the past year I familiarized myself with the current wireless and wired network security standards that were used in our solution. I made the decision to use the FreeRADIUS server, an open-source and relatively well-document implementation of a RADIUS server. After learning how the different components of the FreeRADIUS server worked, I configured it to use EAP-TLS authentication for local users and inserted the modified code to check for guest credentials in the authentication process.

In addition to managing the RADIUS server software, I helped in designing the physical setup of our solution and pieced the components together. I was responsible for keeping the components working together and implementing any changes to our setup design. I set up and administered the CA that acted as the trusted root for the RADIUS server and local users, generating key pairs for testing and for participants of our pilot. I set up a

* Actually, the authentication server also acts as an authorization server as authentication grants local users access to the network. This is especially true for guest access where the server checks the guest's authorization certificate to decide whether or not to grant access.

DHCP server that assigns an IP address to unauthorized guests that associate to the guest VLAN and a DNS server that restricts traffic originating from unauthorized guests and redirects them to our delegation main page.

After the initial prototype was completed, my work involved cleaning up the prototype and incorporating new components provided by Cisco into our solution. To this effect I set up the Cisco Secure Access Control Server (ACS) on a machine running Windows 2000 Server and attempted to incorporate the ACS as our authentication server. The design process and authentication procedure described in chapters 3, 4, and 5 refer to our original prototype and the revised model running FreeRADIUS. The work on the Cisco ACS was performed later and is discussed in Section 6.1.

Finally, I played an administrative role in carrying out our pilot [Pow04]. The participants were issued the necessary X.509 certificates used in authenticating as local users. I also provided instructions on how to set up the client's machine to enable EAP-TLS authentication.

A more general discussion of the Greenpass project can be found in [SGK+04] and [GKS+04]. A discussion focusing on the delegation of guest access [Gof04] will be forthcoming. The results and discussion of the pilot [Pow04] will be available as well.

1.4 Roadmap

This thesis discusses the authentication process of Greenpass.

- Chapter 2 examines the wireless security technology we make use of in our solution. This includes discussion of EAP and EAP-TLS, the RADIUS protocol, and an overview of Virtual LANs.
- Chapter 3 describes the physical components of Greenpass, starting with our initial prototype and evaluating the revisions that were made.
- Chapter 4 presents the authentication and authorization from the RADIUS server's perspective, outlining the decision process it undertakes in deciding whether to accept or reject a supplicant. A brief discussion of the guest delegation process is also included.
- Chapter 5 discusses Greenpass from the client's perspective, describing the necessary steps needed to set up the client's machine to support our authentication scheme and evaluating the process the client sees.
- Chapter 6 introduces some prospects for future work and investigates some of the possible extensions of Greenpass.
- Chapter 7 summarizes my work on the project and offers some concluding remarks.
- Chapter 8 provides a glossary of the terms used in this thesis, including the numerous acronyms that are part of the vocabulary of network security.

2. Background

Before discussing the details of the Greenpass project, it is necessary to discuss some of the standards and protocols defined in the field of wireless (and wired) network security. In this chapter, I will introduce the various acronyms that appear in dealing with this field and explain how these standards are incorporated into our solution.

2.1 Wired vs. Wireless

Wireless network security poses additional problems that those involved in keeping a wired network secure. In a wired network, the computer is physically connected to the institution's network. If a machine is connected to a subnet identified with a certain physical location, it is known that the machine must be physically connected to that location. With wireless networking, however, the transmission of the signal via radio waves removes this physical requirement. A machine no longer needs to be inside a certain building to connect to its wireless network; it just needs to be within radio range. This clearly opens the wireless network to a greater range of potential abuses.

In addition to determining whether the supplicant should be granted access, the use of radio transmission also necessitates the need to protect the communication between the client and the access point. Until recently, *wired equivalent privacy (WEP)* was available to protect the session between the supplicant and the access point. Due to major flaws in

the scheme, however, the WiFi Alliance, a vendor consortium formed to certify interoperability of wireless LAN products, changed its certification rules to now require *WiFi protected access (WPA).* WPA is a subset of *802.11i*, the IEEE draft that defines the standard for future 802.11 security. WPA offers a stronger encryption scheme and provides support for a wider variety of authentication techniques, marking a vast improvement over WEP [EA03].

2.2 EAP and EAP-TLS

Both WPA and 802.11i use *802.1x* [IEEE01], a general control mechanism for any Ethernet-based network. 802.1x generalizes the *Extensible Authentication Protocol (EAP),* originally designed as an extension of the *Point-to-Point Protocol (PPP)* used in dialup sessions. While EAP was originally used to authenticate dialup subscribers, it has since been adopted for use in authentication in various settings, including the wireless network. One of the public-key authentication schemes supported by EAP is *EAP-TLS* [AS99, BV98], which uses TLS (*transport layer security*) as its authentication scheme.

TLS is the standardized version of *SSL (secure sockets layer),* the mechanism that secures connections on the Web. SSL/TLS typically allows a Web server to present an X.509 public key certificate to the client and prove knowledge of the corresponding private key, thus proving to the client that she is communicating with the Web server she thinks she is connected to. Furthermore, SSL/TLS allows the server to request an X.509 certificate from the client as well, using the certificate to decide whether to grant the client access.

SSL/TLS, therefore, provides for mutual authentication of both the client and server and also permits the parties to negotiate a cryptographic suite and establish shared secrets to secure the session.

While SSL/TLS is mostly used in a Web setting, the EAP-TLS variant within 802.1x is used to authenticate users over the wireless network. The wireless supplicant plays the role of the Web client and the *authentication server*, working with the access point, fills the role previously held by the Web server.

2.3 The RADIUS Protocol

The authentication server, in our case a RADIUS (*Remote Authentication Dial In User Service*) server [Rig00, RWC00, RWRS00], is responsible for making the decision to accept or reject a supplicant. RADIUS is a protocol that was originally used to authenticate users over dialup connections, but is increasingly used for other authentication scenarios, including the wireless network. The access point, acting as the *network access server (NAS)* in the RADIUS protocol, acts as a middleman in the handshake between the supplicant and the RADIUS server. If the supplicant is granted access, the RADIUS server passes all the necessary configuration information to the NAS so it can provide access to the supplicant. The RADIUS protocol provides the communication between the NAS and the RADIUS server.

2.3.1 The RADIUS Packet

Communication between the RADIUS server and the NAS requires a specified RADIUS packet format. The variable-length packet contains 5 different fields: the *code* specifies what type of packet is being sent; the *identifier* is a serial number to ensure packets are not skipped; the *length* field announces the length of the packet; the *authenticator* field ensures the packet originated from the expected source and has not been corrupted; and the *attribute* field lists the various configuration settings that dictate to the NAS what services should be provided to the supplicant.

Figure 2-1 presents the various fields of the RADIUS packet with their respective sizes. There are a handful of different RADIUS codes defined, but we are only concerned with six of them for our purposes. The Access-Request packet (code #1) is sent from the NAS to the RADIUS server requesting access on behalf of the supplicant. Access-Accept (code #2) and Access-Reject (code #3) packets are sent from the RADIUS server to the NAS once the decision to accept or reject the supplicant is made. Until the decision is made, the RADIUS server will send Access-Challenge (code #11) packets to the NAS to request the appropriate credentials from the supplicant. The RADIUS protocol also provides for accounting and this functionality requires the use of Accounting-Request (code #4) and Accounting-Response (code #5) packets.

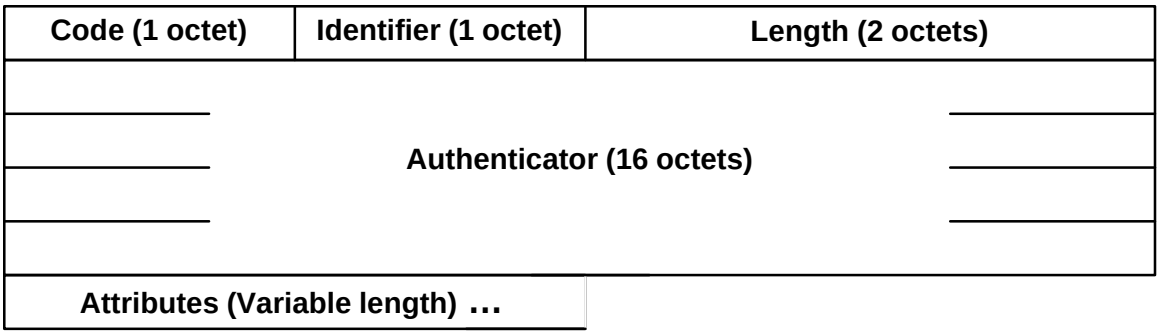


Figure 2-1: The RADIUS packet and a listing of the lengths of each field. The attribute field is of variable length, which causes the length of the RADIUS packet to be variable as well.

2.3.2 The Authenticator Field

The authenticator field confirms to the NAS that it is communicating with the valid RADIUS server. The NAS creates a unique and unpredictable 16-octet random number, known as the request authenticator, and inserts it in the authenticator field of each Access-Request packet. Whenever the RADIUS server responds to an Access-Request packet (with an Access-Challenge, -Accept, or –Reject packet), it must include the corresponding response authenticator in the appropriate field of the response. The response authenticator is computed by taking the hash of the catenation of the RADIUS packet header, request authenticator, response packet attributes, and the shared secret. Packets with invalid authenticators are silently discarded. If an adversary sniffs network traffic between the NAS and the RADIUS server, it could be possible to collect enough packets to potentially crack the shared secret.

2.3.3 RADIUS Attributes

The attribute field comprises a list of the RADIUS attributes that correspond to the supplicant being authenticated. As a whole, the RADIUS attributes define the specific authentication, authorization, information, and configuration details for the request and response. Each attribute is headed by a 1-octet *type* field and a 1-octet *length* field. The *value* of each attribute depends on the attribute in question as is of variable length. A complete list of standard attribute numbers and appropriate values can be found in the appropriate Internet Request for Comments (RFCs) [RWRS00 and ZLR+00].

Of particular relevance are the tunneling attributes that allow the RADIUS server to dictate which *Virtual LANs (VLANs)* [Cis98, Cis99] a supplicant should be placed on.

2.4 Virtual LANs

Virtual LANs are used to partition network resources onto localized networks using software. Resources on a VLAN behave as if they are physically connected on the same LAN even if they are not. VLANs can be used to control what resources users have access to since users placed on a certain VLAN can only access network resources that reside on that same VLAN. Since VLANs are a software solution, they provide much greater flexibility than hardware solutions. A user is assigned to a certain VLAN, so regardless of where on the network she connects from, she can access the same resources.

VLANs, therefore, remove the physical restrictions originally associated with partitioned network resources.

A vital component of VLANs is the IEEE *802.1Q* standard [IEEE98] that addresses the problem of limiting broadcast and multicast traffic within the desired VLAN. Under 802.1Q, a standard method for inserting VLAN membership information into Ethernet frames is provided. Thus packets can be marked with VLAN information. This allows traffic to be limited among the resources on a single VLAN. Furthermore, with 802.1Q *trunking*, a single access point can distribute traffic using the VLAN tag and thus provide service to multiple VLANs.

2.4.1 VLANs in Greenpass

Our current setup of Greenpass partitions the network into two different VLANs. The Native VLAN is accessed by local users and authorized guests and requires authentication to the RADIUS server. The Guest VLAN can be accessed by anyone, but users placed on this VLAN can only access a dedicated Web server housing the guest delegation tool and instructions on obtaining guest access. Once the network is organized into different VLANs, the RADIUS server can easily enforce a policy for each supplicant or group of supplicants and place users on the appropriate VLANs.

3. The Physical Components

This chapter describes the physical components of the Greenpass project and how they are set up. Our original prototype completed in December 2003 was limited by the available hardware resources we had available to us. Using the new resources provided by Cisco allowed us greater functionality and flexibility in our revised model. Section 3.1 describes the original prototype and its limitations and Section 3.2 presents the revised model of the project and discusses the changes in the new model.

3.1 The Prototype

Among the freely available RADIUS servers, I decided to use FreeRADIUS version 0.9.2 [Free] due to its detailed documentation and its ability to handle several different authentication protocols. The RADIUS server, running on a Dell P4 workstation with Red Hat Linux version 9, handles all authentication requests sent by a Cisco 350 access point. Another Dell P4 workstation houses an Apache Web server to handle the guest delegation and to provide instructions for guests to obtain guest access. All of these components are connected via a Cisco Catalyst 2900 series switch, configured with two different VLANs, which are assigned to physical ports on the switch. Figure 3-1 illustrates how these components are set up.

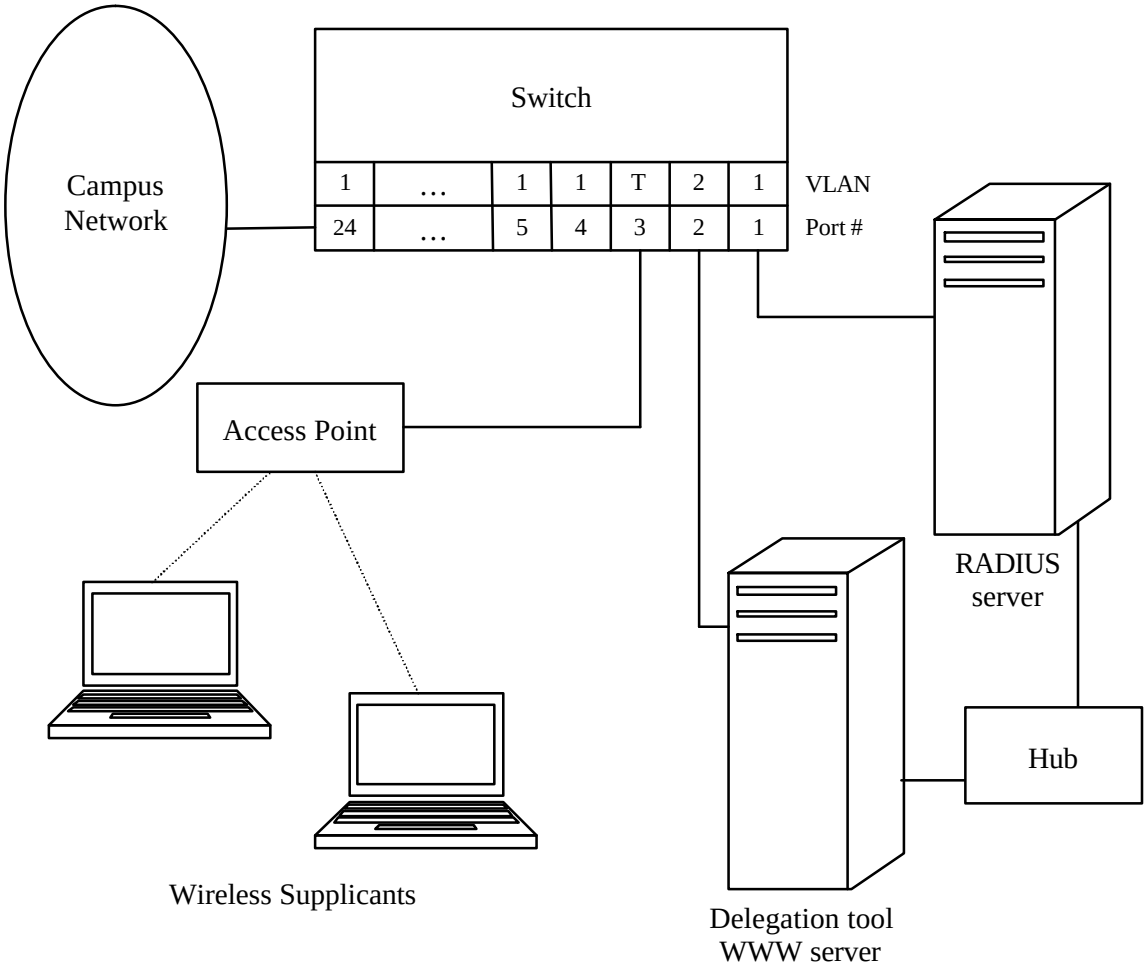


Figure 3-1: The prototype of the Greenpass project. The physical ports on the switch are configured to different VLANs. In the prototype the port connecting the Web server is assigned to VLAN 2, and the port connecting the access point is assigned to a trunking port.

VLAN 1, or the native VLAN, is granted full access to the network and the Internet; authorized users will be given access to VLAN 1. VLAN 2, or the (unauthorized) guest VLAN, allows access only to the Web server for guest delegation; unauthorized guests are placed on VLAN 2, and thus given access only to the guest delegation page. I will refer to the two VLANs as V_1 and V_2 in the following discussion. On the switch, the RADIUS server and a wire to the external campus network are both connected to V_1

ports. The machine running the Web server is connected to a V_2 port on the switch. The access point is connected to a designated *trunking* port on the switch, meaning it can send packets to resources on either VLAN.

The access point is configured with two different *Service Set IDs (SSIDs).* The SSID named "Greenpass Test" is broadcast and is given access to V_2. Any user searching for an available wireless network can find this service set and can connect to it, although they will be placed on a VLAN that allows access only to the guest delegation page. The second SSID, called "Dartmouth user," is not broadcast and is linked to V_1. In order to associate to this service set, the supplicant must enter the name of the SSID and then authenticate to the RADIUS server.

3.1.1 Limitations of the Prototype

The major problem with the prototype was the absence of a router capable of trunking traffic across the VLANs. At some point in the delegation process, the Web server and the RADIUS server need to communicate with one another. The workaround we used was to connect the two machines on a private connection via a network hub. This solution, however, requires physical proximity of the two machines, which decreases the flexibility of the solution. The problem is compounded if there are multiple RADIUS servers utilized to handle authentication in a larger institution.

Furthermore, the delegator would need to access the Web server as part of the delegation process. Placing the Web server only on V_2 would prevent access by authorized users (i.e., non-guests), unless they associated to the guest SSID. This is an odd restriction to enforce. It is possible to grant the delegator access to resources on both V_1 and V_2, but this unnecessarily creates a new class of access policy to enforce. Thus having the Web server reside on both VLANs would be a more flexible solution. The revised model adheres to this solution.

3.2 The Revised Setup

The major change made in the revised model is the addition of new components. The Cisco 2600 series router is capable of routing traffic across different VLANs, providing communication between the RADIUS server and the Web server. The router is configured to separate our setup from the campus network. A subnet on the Dartmouth network was assigned to the components of the Greenpass project and routing entries were added to allow our router to send traffic between the external campus network and our subnet. The access point and switch were replaced by newer models (3550 series switch and 1100 series access point) and configured in the same manner. Figure 3-2 shows the layout of the revised version of Greenpass.

There are a couple of changes to note regarding the WWW server. First, as discussed in the previous section, the Web server uses two network interfaces, one connected to each VLAN. The interface on V_1 is configured with a routable IP address and can be accessed

normally from anywhere on the Web. The interface on V_2 is configured to a private 10.0.0.* network that is used in the guest delegation process. It is through this network interface that unauthorized guests will be able to access the delegation tools that reside on the Web server and complete the guest delegation process.

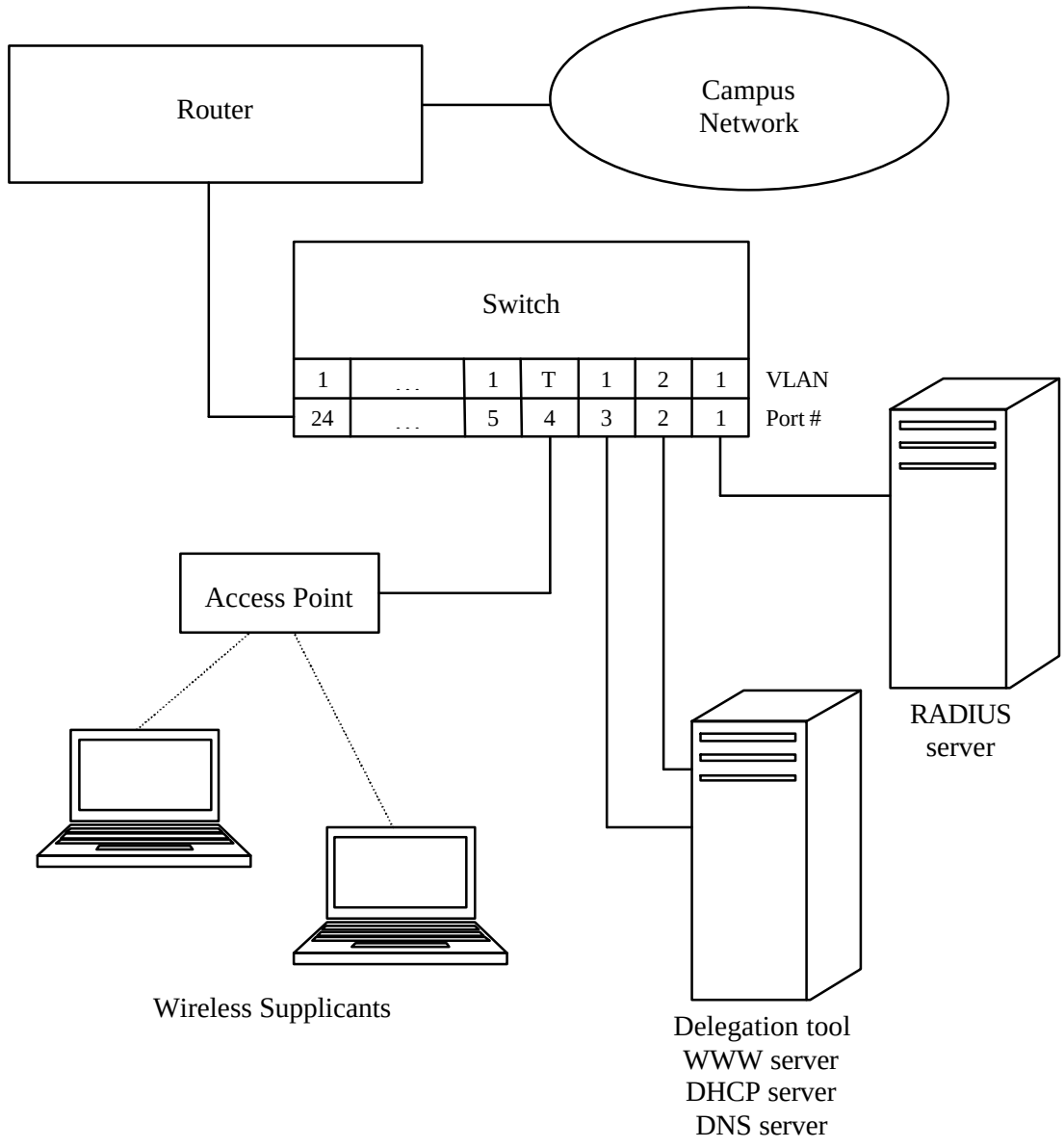


Figure 3-2: The revised setup of the Greenpass project. The router separates the main campus network from the subnet our components reside on. The WWW server has two network interfaces, one connected to each VLAN. The DHCP server assigns a temporary IP address on our private network for use during the delegation process. The DNS server helps redirect all traffic from unauthorized guests to the guest delegation Web page.

Second, the machine housing the Web server now also hosts a DHCP server and a DNS server. When an unauthorized guest arrives and is placed on V_2, the DHCP server will issue the guest a short-lived 10.0.0.* IP address. Since users on V_2 do not have a connection to the external network, they have no way of obtaining an IP address through DHCP unless we provide an IP address on our private subnet. With the IP address on our private network, the guest can talk to the Web server and traffic can be routed to the guest as well.

3.2.1 Capturing and Redirecting Web Traffic

The unauthorized guest also does not have access to the standard DNS servers available on the external network. Thus a DNS server is also necessary on V_2 to provide domain name resolution. To an unauthorized guest residing on V_2, however, domain name resolution in itself is not useful because she does not have access to the external network. Thus even if a standard DNS server were available, it would serve no useful purpose. It is not elegant, however, to have the guest see a "page not found" error for every external website she tried to access. It would be better to redirect all external Web traffic originating from an unauthorized guest to our dedicated Web server.

We can easily modify our DNS server to implement this redirecting of Web traffic. Creating a wildcard entry that resolves all domain names to the private network IP address of our Web server will substitute our Web server domain for whatever domain the guest enters into her browser. Our Apache server is configured to redirect access to

the Web server's root public directory to the location of the delegation tool. Thus when the guest enters the URL "www.google.com," the domain name is resolved to "dupin.dartmouth.edu" (the hostname of the Web server), and Apache redirects the browser to "dupin.dartmouth.edu/Greenpass/cgi-bin/grandcentral.py" (the URL of the delegation tool on the Web server).

There are two details that need to be addressed in this solution. First, the DNS server simply takes the domain name of the URL and replaces it with the IP address of our Web server. Thus any path specified on the desired host would simply be appended to the hostname of our Web server. For example, requesting an article on CNN's website with the URL "http://www.cnn.com/2004/WORLD/meast/05/09/hersh.iraq.abuse/index.html" would result in the DNS parsing out the host name and Web browser returning the page "dupin.dartmouth.edu/2004/WORLD/meast/05/09/hersh.iraq.abuse/index.html," a page that obviously does not exist on our Web server. Thus it is necessary to filter out the requested URL and leave just the hostname of the requested page.

The *RedirectMatch* command in the *alias* module provided by the Apache Web server can address this problem. This command takes a regular expression and redirects any requests to the path or filename that match the regular expression to the specified destination. All of the delegation pages can be found in a folder called "Greenpass," so any requests to the server outside of the path "dupin.dartmouth.edu/Greenpass/" is assumed to be the result of an external request. Thus all such requests are redirected to

the delegation front page while allowing traffic to the delegation pages to pass through unhindered.

The second issue that needed to be addressed was that the delegation front page was cached under the domain name of the original request. Referring to the earlier example of a request to "www.google.com" while the guest's browser would get our delegation front page, it would also cache our page as "www.google.com." Thus after the guest is delegated access to the network, a query to "www.google.com" would result in the cached copy of the delegation front page being fetched until the browser's cache was cleared. The *expires* and *headers* modules in Apache provide a method of inserting a Cache-Control command in the header of each HTTP request made to the Web server that directs the requesting Web browser to not store the pages into its cache.

4. Making the Decision (The Server Side)

While the physical components are the tools of the Greenpass project, it is necessary to discuss how these tools are used in making the decision to authenticate or reject a supplicant. This chapter describes the decision process the RADIUS server makes and explains the process through which a guest is delegated access. Section 4.1 introduces some of the design decisions that went into formulating our solution. Section 4.2 discusses in detail the EAP-TLS handshake that local users use to authenticate. Section 4.3 explains how EAP-TLS is modified to handle guest credentials and Section 4.4 provides a brief description of the delegation process. Finally, Section 4.5 briefly explains how Greenpass supports authentication using VPNs.

4.1 Design Considerations

Recall from the earlier discussion that there were several requirements that we wanted to fulfill with the Greenpass solution. In designing our solution, these factors were considered:

- Local users must be granted access to the network with little or no additional effort required by the authentication process.
- Desired guests must be granted access to the network through a decentralized and hassle-free delegation process.

- The solution must be flexible: it must scale to large institutions, support multiple client platforms, and extend to satisfy various access policies.
- The solution must be secure and robust enough to defend against various attacks.

Using a RADIUS server to handle authentication was a logical choice due to the adaptability of the RADIUS protocol and the availability of open-source implementations. Taking advantage of the pre-existing PKI, having Dartmouth users authenticate through EAP-TLS was the best solution as it did not require passwords and provided mutual authentication. Once local users performed the initial configuration required for EAP-TLS authentication (obtaining a X.509 identity certificate issued by the trusted CA, setting the preferences for the local user SSID, and enabling 802.1x authentication), the authentication process requires no additional effort from the supplicant.

For the sake of simplicity, the RADIUS server must handle the authentication of both guests and local users. The ideal solution would then have the RADIUS server determine whether the supplicant is a local user or a guest. If she is a local user, then the standard EAP-TLS protocol would be used to authenticate her; if she is a guest, then the RADIUS server would automatically make its decision taking that fact into account rather than rejecting the guest for not being a local user. Pursuing the most straightforward approach, I worked on modifying the RADIUS server's decision algorithm to check for guest credentials. If the guest is authorized for access, then the RADIUS server will

grant access automatically; no additional work is required for authorized guests compared to local users. If the guest is not authorized for access, then she will be given instructions on how to obtain guest access. The specific guest credentials and the full authentication decision process are described in detail later in this chapter.

By modifying EAP-TLS to implement guest authentication we take advantage of the inherent benefits of EAP-TLS. EAP-TLS provides the basic communication channel used in the authentication process between the guest and the RADIUS server. In addition to making it easier to implement, the EAP-TLS framework also provides the basic security necessary to prevent attacks.

4.2 Using X.509 Certificates for Local Users

Standards and protocols that use certificates and PKI, including SSL/TLS, usually employ *X.509 identity certificates* to authenticate servers and clients. Typical implementations of a PKI entail a central *Certification Authority (CA)* issuing and signing identity certificates for users. The identity certificate is associated with a generated key pair for the user. The certificate states the user's identity, and certifies that the user owns the public key that is included in the certificate. The user proves her identity by demonstrating ownership of the private key that belongs to the public key in the identity certificate.

Dartmouth has a full PKI in place with its own CA to issue user X.509 certificates for Dartmouth students, faculty, and staff. The use of certificates for authentication purposes is becoming more prominent and new incoming students are issued a certificate by the Dartmouth CA. Students can authenticate themselves using their own X.509 certificate to a Web server in order to register for the term, add and drop courses, and check their grades online.

Taking advantage of available resources, it makes sense to use EAP-TLS to authenticate local users to the RADIUS server. As discussed earlier, EAP-TLS is a public key based authentication scheme that provides mutual authentication of the client and server, making it an ideal solution to use with our preexisting PKI. The EAP-TLS module of FreeRADIUS uses OpenSSL to execute the SSL/TLS handshake between the supplicant and the RADIUS server. After changing the appropriate RADIUS configuration files to enable EAP-TLS authentication and linking the OpenSSL libraries [Sul02], the RADIUS server was ready to accept EAP-TLS authentication attempts. The client file was configured to only accept requests sent from an access point with a Dartmouth IP address and the user file was set to only allow EAP (in our case EAP-TLS) authentication.

In order to use EAP-TLS authentication, the RADIUS server needs a trusted root CA so that it knows which certificates to accept. The RADIUS server also needs its own server certificate and key pair issued by the trusted root CA for authenticating itself to the supplicant in the handshake process. All local users are given a key pair and issued client certificates signed by the trusted root CA. OpenSSL can be used to generate key pairs,

create a root certificate, and issue server and client certificates [Ros03]. Although we eventually plan on using the Dartmouth CA as our trusted root, for the purposes of experimentation I created a separate CA to issue certificates for test clients and the RADIUS server. Once the RADIUS server has a trusted root CA to refer to, it can handle authentication requests from the access point.

4.2.1 The EAP-TLS Handshake

When a supplicant associates with the Dartmouth user SSID, EAP-TLS authentication is required for the supplicant to be granted access. An EAP-TLS handshake ensues between the supplicant and the RADIUS server, with the NAS acting as a middleman and forwarding messages between the two. During the handshake, the supplicant provides an X.509 identity certificate and demonstrates knowledge of the private key that goes with the certificate.

Figure 4-1 depicts the EAP and RADIUS packets sent among the three parties during the EAP-TLS handshake. A detailed discussion on the EAP and EAP-TLS handshake process can be found in [AS99, BV98]. The major detail to note in this handshake process is how EAP-TLS works within the RADIUS framework. The NAS uses EAP to communicate with the supplicant, but uses the RADIUS protocol to talk to the RADIUS server. The NAS initiates the handshake with an EAP-Request packet when a wireless supplicant associates to the access point. The EAP-Response packet from the supplicant is encapsulated in a RADIUS Access-Request packet and forwarded to the RADIUS

server. The RADIUS server sends a RADIUS Access-Challenge packet back containing an EAP-Request packet that states it requires EAP-TLS authentication and the necessary server side information for the TLS handshake. The supplicant fulfills its part in the TLS

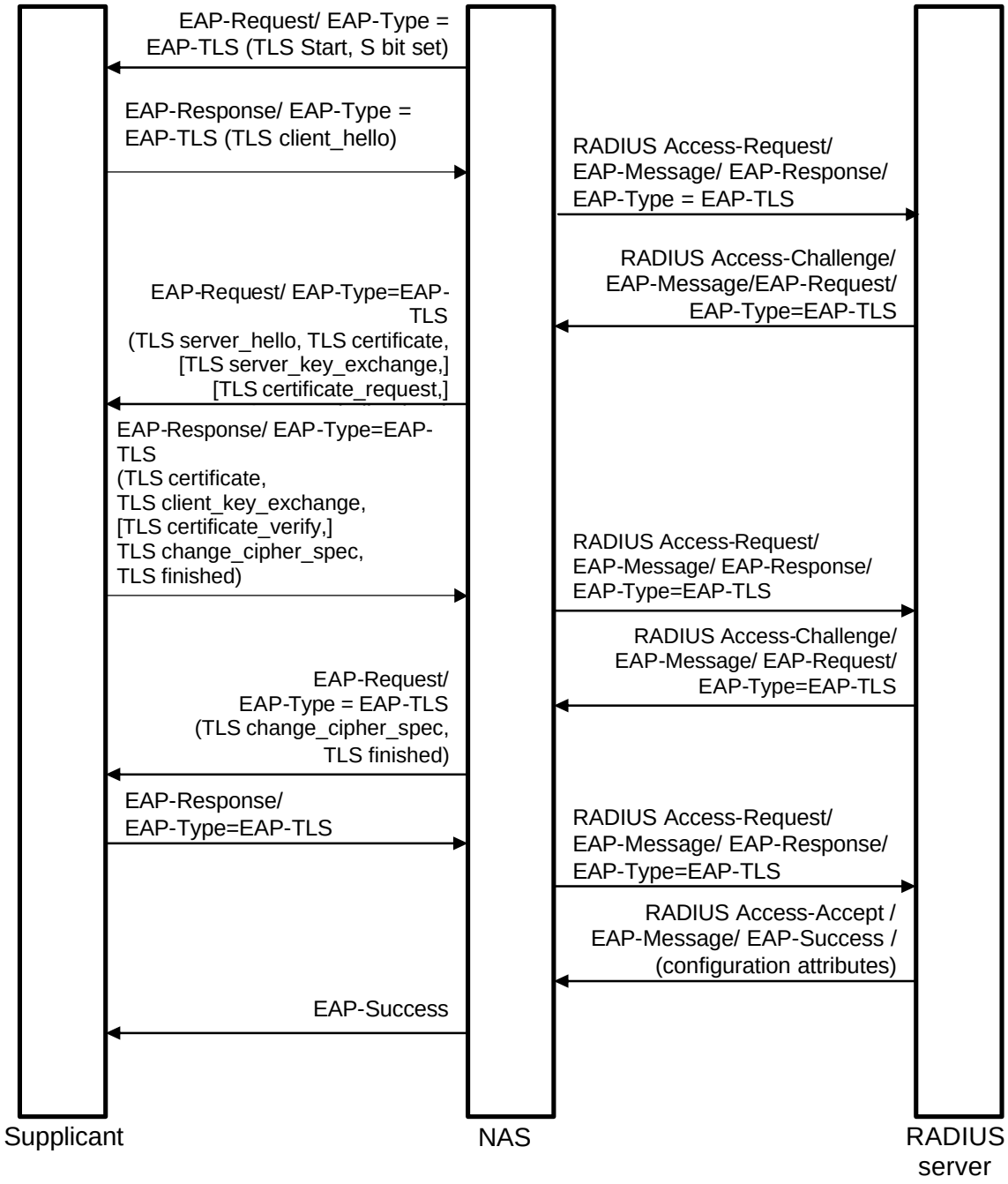


Figure 4-1: A successful EAP-TLS handshake working within the RADIUS protocol. The NAS acts as a middleman between the supplicant and the RADIUS server, using the EAP protocol to communicate with the supplicant and the RADIUS protocol to talk to the RADIUS server.

handshake procedure with an EAP-Response packet (containing the clients X.509 certificate). The process continues until the TLS handshake either succeeds or fails, with the NAS packing and unpacking the necessary EAP packets into the appropriate RADIUS packet formats.

Once the TLS handshake is complete, the RADIUS server has made its decision on whether to accept or reject the supplicant. If the supplicant is accepted, the RADIUS server sends back a RADIUS Access-Accept packet back to the NAS with the necessary configuration options for the supplicant contained in the attribute field of the packet. The NAS will use those RADIUS attributes to provide the level of service to the supplicant as dictated by the user settings for the supplicant. Typical configuration information includes the VLAN to place the supplicant on or a shared secret to encrypt packets. Inside the RADIUS packet is an EAP-Success packet the NAS sends to the supplicant to inform that the authentication process has succeeded. If the supplicant fails authentication, the RADIUS server sends an Access-Reject packet encapsulating an EAP-Failure packet and the supplicant is denied access to the network.

4.3 Modifying EAP-TLS for Guest Access

The problem becomes slightly more complicated for guest users. Ideally, the CA of one institution should be able to easily verify the credentials supplied by a supplicant from a different institution. In actuality, however, this interoperability between arbitrary PKIs has not yet been implemented. (There are current projects, such as the *higher-ed bridge*

CA, that aim to address this issue of incompatibility and attempt to facilitate the cross-authentication of certificates from different institutions.)

It should be noted, however, that the RADIUS server can perform a number of different authorization checks during the EAP-TLS handshake until the RADIUS server makes its decision and hands down an accept or reject packet. Thus we can handle guest access by modifying the RADIUS server to base its decision on some other authorization scheme. Policy languages such as *XACML*, *Keynote* [BFIK99] and *PolicyMaker* [BFL96] *authorization certificates*, X.509 standard *attribute certificates (AC)* [FH02], and *SPKI/SDSI* authorization certificates [EFL+98, EFL+99a, EFL+99b] are some of the options we considered.

4.3.1 SPKI/SDSI

For Greenpass, we settled on SPKI/SDSI *(Simple Public Key Infrastructure/Simple Distributed Security Infrastructure)* to provide the flexibility and operability we require to implement the level of guest access we want. We chose SPKI/SDSI for three main reasons: (1) it focuses specifically on the problem of authorization that we are trying to solve, (2) its emphasis on delegation of authority easily gives rise to the decentralized model of guest access we envisioned, and (3) it is lightweight and easy to process.

A SPKI/SDSI certificate is represented simply as an *s-expression* consisting of five fields: issuer, subject, delegation, validity dates, and authorization (optional). The issuer,

subject, and validity date fields are self-explanatory. The delegation field is a Boolean value stating whether or not the subject is allowed to propagate authority. The authorization field contains an s-expression that describes more detailed authorization properties.

SPKI/SDSI also provides a few major differences from traditional X.509-based PKI, a fact that we take advantage of in our guest delegation scheme. First, SPKI/SDSI uses public keys as unique identifiers as the principals involved are associated with their public key rather than a name. With guests arriving from many different institutions, it is possible that two people with the same name have X.509 identity certificates issued by different institutions. By referring to guests by their public keys, we avoid a potential naming conflict.

Second, a SPKI/SDSI certificate binds authority directly to the public key rather than to a name. In traditional PKI, authority is bound to a name and the public key is also bound to the name. Hence the authentication process requires both bindings to be confirmed before access is granted. A simple example using fingerprints as identifying factors can demonstrate this difference. In the traditional PKI example, Alice is said to have access to the building. When Alice wishes to enter the building, her fingerprints identify her as being Alice (public key bound to the name) and someone checks to see if Alice is allowed access to the building (authority bound to the name). In the SPKI/SDSI case, however, the person with Alice's fingerprints has access to the building. When Alice tries to enter the building, her fingerprints are taken and checked for access (authority

bound to public key) and she is granted access. The fact that fingerprints are a unique identifier (as are public keys) allows authority to be verified without checking for Alice's name.

Finally, any person or entity can potentially issue a SPKI/SDSI certificate. If the user is authorized to do so, she can delegate access to another user by issuing a SPKI/SDSI certificate and signing it. Hence we have a clear mechanism for implementing our decentralized guest access. For example, Dartmouth issues Alice, a professor, a SPKI/SDSI certificate authorizing her to grant access to guests (specifically, the delegation field is set to true). When Gary, a guest, arrives at Dartmouth, he will request access to the network and contact Alice to get guest access. If Alice determines that Gary should be given guest access, she issues and signs a short-lived SPKI/SDSI certificate that grants him access for the duration of the certificate. This guest delegation is done without a Dartmouth central administrator getting involved.

4.3.2 Using SPKI/SDSI to Authenticate Guests

We can use the expressive power of SPKI/SDSI certificates to make an assertion that an authorized delegator granted temporary access to a guest. When a guest wants to get access to the network, she must find a delegator to grant her that access. The guest must prove her identity to the delegator with an X.509 certificate and somehow prove that she is the person who the certificate was issued to. If the guest does not have a X.509 certificate, a temporary one will be issued for the sake of the authentication process.

Once the delegator is satisfied, a SPKI/SDSI certificate is created where the issuer field is the delegator's public key and the subject field is the guest's public key. Since we are granting guest access, the delegation field is set to false as to prevent the guest from further delegating network access to others. The certificate is then signed with the delegator's private key.

Similar to X.509 certificates, we need a chain of SPKI/SDSI certificates that lead up to some recognized trusted root. In order for the SPKI/SDSI certificate issued by the delegator to be sufficient to grant the guest access, there must be a chain from the delegator leading up to some trusted root. For each non-root certificate in the chain, the issuer of the certificate must be validated by a similar certificate in which her public key is the subject. Each certificate in the chain must have the delegation set to be true in order for this chain to be valid. Once such a chain is found, the guest can be granted access.

With SPKI/SDSI certificates, the authentication process needs to be only modified slightly to accommodate guest access. Figure 4-2 shows the decision process the RADIUS server goes through in this endeavor. As in the case of Dartmouth users, a guest associates to an access point and is asked to present credentials to the RADIUS server. The guest submits an X.509 identity certificate issued by some non-Dartmouth CA (or a dummy certificate issued by the delegation tool should the guest not possess a X.509 certificate) that vouches for the guest's identity. The RADIUS server accepts the certificate and notices that the CA that issued the certificate is different from the trusted

root CA that it bases its decisions on. Rather than rejecting the authentication request, however, the RADIUS server attempts to find a SPKI/SDSI certificate chain in the cache that vouches for the public key belonging to the presented X.509 certificate. If such a chain can be found, the guest is granted access. Otherwise, the guest will associate onto V_2 where the Web server will provide directions for obtaining guest access.

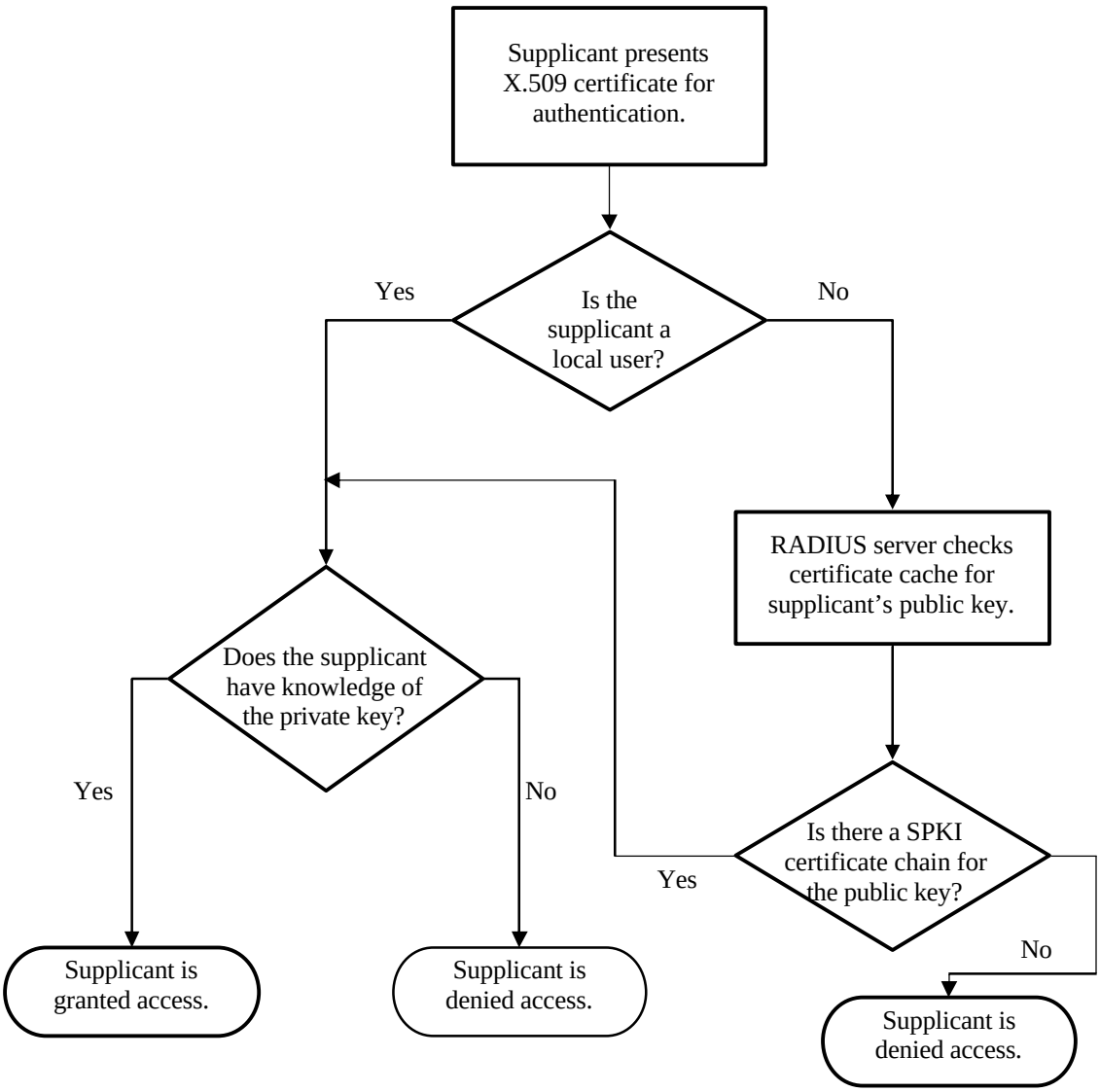


Figure 4-2: Decision flowchart used by the RADIUS server. If the supplicant is a local Dartmouth user (i.e., presents an X.509 certificate issued by the Dartmouth CA), then the supplicant only needs to prove knowledge of the private key associated with the certificate. Otherwise, if the supplicant is a guest, then the RADIUS server checks for a SPKI/SDSI certificate chain vouching for the public key.

Once the guest receives her SPKI/SDSI certificate chain (through the procedure outlined in Section 4.4), she can attempt to authenticate to the RADIUS server. This time, the RADIUS server will find her SPKI/SDSI certificate chain, verify that the public key on the X.509 certificate is the subject of the SPKI/SDSI certificate, check that the guest has knowledge of the private key that corresponds to the said public key, and grant access to the guest.

The SDSI project at MIT [SDS] provides implementations of SDSI code in Java and C. We make use of a modified version of the Java library code to certify SPKI/SDSI certificate chains for our authentication process. The library uses the SPKI/SDSI certificate chain discovery algorithm proposed by Clark et al [CEE+01]. At the moment, an XML-RPC server executes this Java code and I modified the RADIUS server to use XML-RPC to query the server about the public key of the X.509 certificate it received. If the Java library can find an appropriate chain of SPKI/SDSI certificates that vouch for the guest, the RADIUS server accepts the guest's request for network access and sends the necessary configuration information to the NAS to provide service. Guest access is temporary, so the validity date on the SPKI/SDSI certificate delegating access to the guest will enforce the lifespan of the temporary access.

4.3.3 Alternative Approaches to Guest Authorization

In addition to SPKI/SDSI, we considered other approaches to delegated guest access. Each alternative had its advantages and disadvantages, but we felt that SPKI/SDSI was

the best option. I will discuss the possibility of using a couple of these alternative approaches in Chapter 6.

X.509 attribute certificates work similarly to SPKI/SDSI by binding a short-lived authorization to the holder of a particular X.509 identity certificate. Attribute certificates directly address the issue of authorization, but has some major drawbacks. ACs are meant to be issued by a small number of *attribute authorities (AAs)*, which restricts some of the flexibility in creating a distributed model of delegation. Chains of ACs are difficult to process and administer, and for that reason the attribute certificate profile [FH02] advises against using them for delegation. Furthermore, X.509 ACs are not part of the standard implementations of protocols and cannot be transmitted easily in EAP-TLS and other public key certificate protocols. If standard 802.11 clients could transmit ACs along with X.509 identity certificates as part of the EAP-TLS handshake process, this built-in means to transmit guest credentials would offer a great advantage.

The PERMIS system [COB03, Per] also allows users to issue authorization certificates. This feature, however, was originally designed to be used by a fixed set of authorization certificate issuers and hence is not catered to support a distributed system of delegation. Section 6.3 examines the PERMIS system in more detail.

One of the most basic implementations of guest access could be achieved through an *access control list (ACL)*. Temporary ACL entries in a centralized database would grant a guest short-lived access to the network. Authorized "delegators" could modify parts of

the ACL database to add an ACL entry that grants network access to a guest. This solution requires a closely-guarded centralized database, which is not optimal since we would like to move towards a more decentralized approach to guest access. Maintaining a large database of ACL entries can quickly become a complex task and would cut down on the scalability of the solution.

In the end we chose SPKI/SDSI because it seems to best represent the real-world model of guest delegation. In the real world, authorization tends to be granted by a local authority or policy that governs the desired resource rather than by a central authority that governs all resources. When Alice grants Gary access to Dartmouth's network, it is inefficient to require a central administrator to approve of this access. The characteristics of SPKI/SDSI naturally provide this mechanism for distributing delegation among local authorities.

4.4 A Brief Discussion of the Guest Delegation Process

The following section gives an overview of the process the guest and delegator go through in creating and signing a guest SPKI/SDSI certificate. A more detailed description of the process and the tools used in this process can be found in [Gof04].

Once the guest user is placed on V_2, she is assigned an IP address on the private 10.0.0.* subnet by our DHCP server. The DHCP server is necessary because once the guest is placed on V_2 she cannot communicate with the external network. Without a DHCP

server, the guest will not be able to obtain an IP address and cannot continue the delegation process. Furthermore, the DHCP server informs the guest to use our DNS server for queries. This is necessary because (1) once again the guest cannot access the "real" DNS server on the external network and (2) our DNS server is used to implement our captive portal to redirect all traffic to our Web server. Similar solutions are used for wireless hotspots in cafes and network access in hotel rooms.

Instead of using firewalls to capture packets, however, the guest VLAN allows us to simply use our DNS server to create the same effect. This is implemented by creating a wildcard entry in the default "." zone that resolves all hostnames to the 10.0.0.* IP address of our Web server. Recall that the guest is still isolated from the external network, thus services like e-mail will not function. Attempting to access hosts by their IP address will fail as well since there is no way to route packets to those IP addresses on V_2. Once the guest fires up a Web browser window, she will be greeted by our guest delegation page.

4.4.1 Presenting a Guest Certificate for Delegation

As stated earlier, the guest needs an X.509 identity certificate and a SPKI/SDSI authorization certificate issued and signed by an authorized delegator that vouches for the public key found in the X.509 certificate. So naturally the guest must submit her X.509 certificate in order for a delegator to create an appropriate SPKI/SDSI certificate. This can be achieved easily via an SSL/TLS handshake between the guest and the Web server.

During a SSL handshake, the client presents her X.509 certificate, and a successful handshake proves that the client knows the private key associated with the public key found in that certificate.

The Web server provides for three different scenarios for arriving guests. If the guest has an X.509 certificate issued by some party and her browser supports SSL, then the guest can present her certificate for a SSL handshake (and later to be used for the creation of her SPKI/SDSI certificate). If the guest does not support SSL authentication, then she can upload her certificate from a PEM-formatted file (the file format commonly used when exporting an X.509 certificate from the certificate store) on her local disk. Finally, if the guest does not have an X.509 certificate, she can generate a key pair and obtain a temporary X.509 certificate issued by our "dummy" CA.

Once the guest presents her certificate, it is placed in a temporary repository where it waits for a delegator to retrieve it and create an appropriate SPKI/SDSI certificate. The guest will be assigned an ID number the delegator can refer to. The guest must contact an authorized delegator at this point to obtain access.

4.4.2 Delegating Access to a Guest

Now that the guest has introduced her certificate, a delegator must evaluate the guest's request and create a guest certificate. In order for someone to be an authorized delegator, he must possess (1) a (usually local user) certificate that allows him access to the

network, and (2) a SPKI/SDSI certificate that has the delegate tag set to true and can be traced back to a trusted root of authority. The delegator connects to our Web server and will be recognized as a authorized delegator, thus giving him the added option of delegating access.

The delegator uses the ID number of the guest's request to find the guest's X.509 certificate. Once the delegator selects a guest request, a trusted Java applet that is used to create and sign the SPKI/SDSI certificate will load. The Java applet is signed by a signer vouched for by our CA, thus the delegator knows that the applet can be trusted. This extra level of security is desirable because the delegator's private key is needed to sign the SPKI/SDSI certificate. Thus it is mandatory to ensure that the private key is not being abused by an unknown applet.

Another security issue that arises at this point is that the delegator must ensure that he is delegating access to the correct public key. It is conceivable that guest's delegation request can be intercepted by an adversary. The adversary then can, through a man-in-the-middle attack, substitute his own public key for delegation. Dohrmann and Ellison address a similar problem of safely introducing collaborating parties through the use of a *visual hash* [DE02]. By using the hash of the guest's public key to create a visual representation, the delegator can easily verify that the public key that he received from the guest actually belongs to the guest by checking if the visual hashes match. It is clearly easier to verify that two pictures are the same than it is to do the same for two large hash values in hexadecimal form.

When the guest supplies her X.509 certificate to the Web server, she will see a visual hash of her public key along with the ID number for her request. Before the delegator can create a SPKI/SDSI certificate for the guest, the delegation applet loads a window with 16 different visual hashes, one of which belongs to the guest. If the delegator selects the correct visual hash, then a SPKI/SDSI certificate is created and signed. The certificate chain is then loaded into a HTTP cookie onto the guest's Web browser as well as inserted into a repository where it can be accessed by the RADIUS server.

The visual hash introduction requires the delegator and guest to meet in person so the delegator can check the visual hash on his screen and on the guest's screen. The guest and the delegator typically will not know each other, thus they must meet in person for the introduction. The physical meeting will enable the delegator to check the guest's identification to make sure the name on the supplied X.509 matches the name on her identifying documents.

Once a guest is delegated access, the SPKI/SDSI certificate chain resides on her Web browser as a HTTP cookie. If for some reason the server side repository of SPKI/SDSI certificates is lost, the guest can revisit the delegation front page to reload her certificate chain to the server. The delegation page will grab the cookie from the guest's browser, verify that the certificate chain is valid, and store the chain in the server's repository. On the other hand, if the guest clears cookies from her browser, she can visit the delegation page and submit the same X.509 certificate. As long as the SPKI/SDSI certificate chain corresponding to that certificate is still present in the repository, another copy of the

certificate chain is inserted into a new cookie and the guest does not need to be delegated to again.

4.5 Supporting VPNs

Another popular method of protecting network resources is through the use of *Virtual Private Networks (VPNs).* A VPN restricts access to resources on the network only to users who can authenticate themselves onto the VPN. Its versatility lies in the fact that users can authenticate to the *VPN concentrator* from anywhere, and a secure channel is established between the VPN and the user's current location. The user can, therefore, access restricted resources on the VPN from a remote location in a secure manner. A VPN concentrator can be configured to refer to an authentication server (such as a RADIUS server) to determine whether a user is authorized to access the VPN. In this model, the VPN concentrator will play the role of the NAS instead of the access point. Thus, the mechanism in which the VPN concentrator works with a RADIUS server is already in place.

We obtained a Cisco 3000 series VPN Concentrator to experiment in our Greenpass project. The details of setting up the VPN concentrator can be found in [Gof04]. The VPN concentrator provides a network interface for the public and the private networks. In the VPN model, the public network represents the network resources that all users have access to, while the private network is the portion that is protected and restricted to authorized users. For our setup, we want the public interface to reside on V_2 and the

private interface to be connected to V_1. The VPN concentrator can act as an EAP proxy and pass along the entire X.509 identity certificate to the RADIUS server for EAP authentication.

Once in place, the VPN concentrator simply acts as a NAS: the supplicant uses a VPN client to connect to the concentrator, the concentrator passes the client's certificate to the RADIUS server, and the RADIUS server sends its decision back to the concentrator. Authorized guests can be authenticated in the same manner as well since the RADIUS server can look at the public key in the guest's X.509 certificate and verify a SPKI/SDSI certificate chain. The VPN concentrator either accepts or rejects the supplicant based on the decision made by the RADIUS server.

5. Authenticating to Greenpass (The Client Side)

In order to evaluate the usability of our solution, it is necessary to examine the entire process from the client's perspective. This chapter will describe the necessary steps a local user and a guest running either Windows or Mac OS must complete in order to use Greenpass. I will argue for the usability of Greenpass at the end of the chapter and support my claims with results of the pilot [Pow04].

5.1 Local Users

As discussed earlier, local users authenticate to Greenpass through an EAP-TLS handshake. To this end, local users need a X.509 identity certificate issued by the root CA and a machine that supports 802.1x authentication. The first requirement is easy to fulfill as institutional CAs tend to have an enrollment page that generates a key pair and creates a certificate for authorized users. The enrollment process typically stores the generated certificate in the user's keystore, so installation of the certificate is usually unnecessary. For our experiments, however, we use a created CA to act as the root authority, hence the user needs to manually install the X.509 certificate, as well as install the CA's certificate into the browser's store of trusted root certificates. This additional step is straightforward and does not add to the complexity of the scheme.

Configuring 802.1x authentication, on the other hand, is a more involved process. The latest versions of operating systems have implemented support for 802.1x. Specifically,

Windows XP (from service pack 1), Windows 2000 (from service pack 4), and Mac OS X Panther (version 10.3) all support 802.1x and represent a large majority of our user base. Users running older versions of these operating systems must use a third-party 802.1x client in order to authenticate using Greenpass. Linux users can download the open-source XSupplicant 802.1x client.

5.1.1 EAP-TLS Authentication for Windows

Once the user has the necessary X.509 certificate for authentication, the Internet and network settings must be configured to allow EAP-TLS authentication. Windows users must use the Internet Options of Internet Explorer to select the option to allow the use of TLS 1.0. This option is located under the *Advanced* tab of the Internet Options screen (see Figure 5-1). Even if Internet Explorer is not the Web browser of choice, client authentication is controlled by the Windows operating system, and changing the settings on IE appears to carry over to the "overall" Windows settings. The default setting has TLS not enabled, so this option must be selected to initiate a TLS handshake.

Since we are dealing with the case of a local user, we need to direct our wireless card to associate to the non-broadcast SSID "Dartmouth User." In order to use 802.1x authentication for a network, the network must use WEP encryption. Under the properties (see Figure 5-2) for the Dartmouth User SSID, the **Network Authentication** field should be set to *Open* and **Data Encryption** should be set to *WEP*. The WEP key for the connection is provided by the access point, so the box corresponding to this option

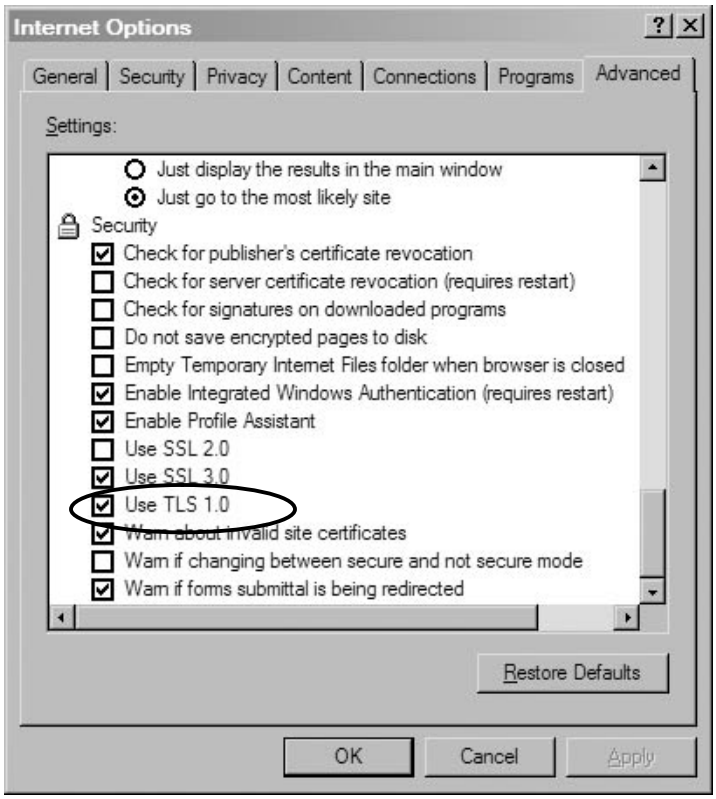


Figure 5-1: Internet Options window for Microsoft IE with TLS 1.0 enabled.

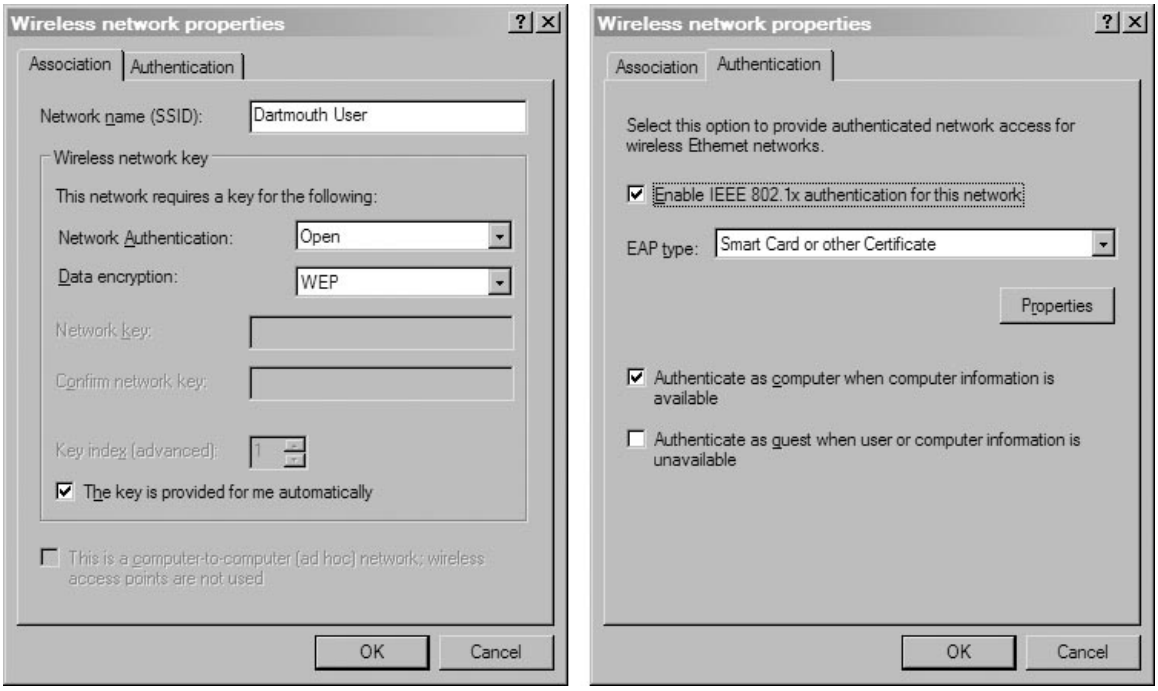


Figure 5-2: Wireless network properties window for the Dartmouth User SSID.

should be checked. Under the **Authentication** tab, the box that reads, "Enable IEEE 802.1x authentication for this network" should be checked and the **EAP type** should be set to *Smart Card or other Certificate.*

The wireless card should attempt to authenticate to the RADIUS server and associate to our network. If more than one possible authentication certificate is found, a window will prompt the user to select which certificate to submit for authentication. A message should appear about processing server information for the network, which will pop up a window asking if the server certificate issued by the CA should be trusted (Figure 5-3). Viewing the certificate verifies that the certificate belongs to the Greenpass RADIUS server and was issued by the Greenpass CA (Figure 5-4). Once the user is satisfied with the server's certificate, the authentication process will continue now that the server has verified itself to the client.

Figure 5-3: Confirmation window asking whether to trust the RADIUS server's certificate.

Once our client certificate is verified and we pass authentication, we are now connected to the Greenpass network. Our subnet on the Dartmouth network is assigned the IP address range 129.170.253.64/27, so the associated local client should be assigned an IP address in that range (see Figure 5-5). This will verify that the client did indeed pass authentication and has been placed on V_1. Checking the association table on the access point (Figure 5-6) will also confirm that the client has been associated onto V_1.

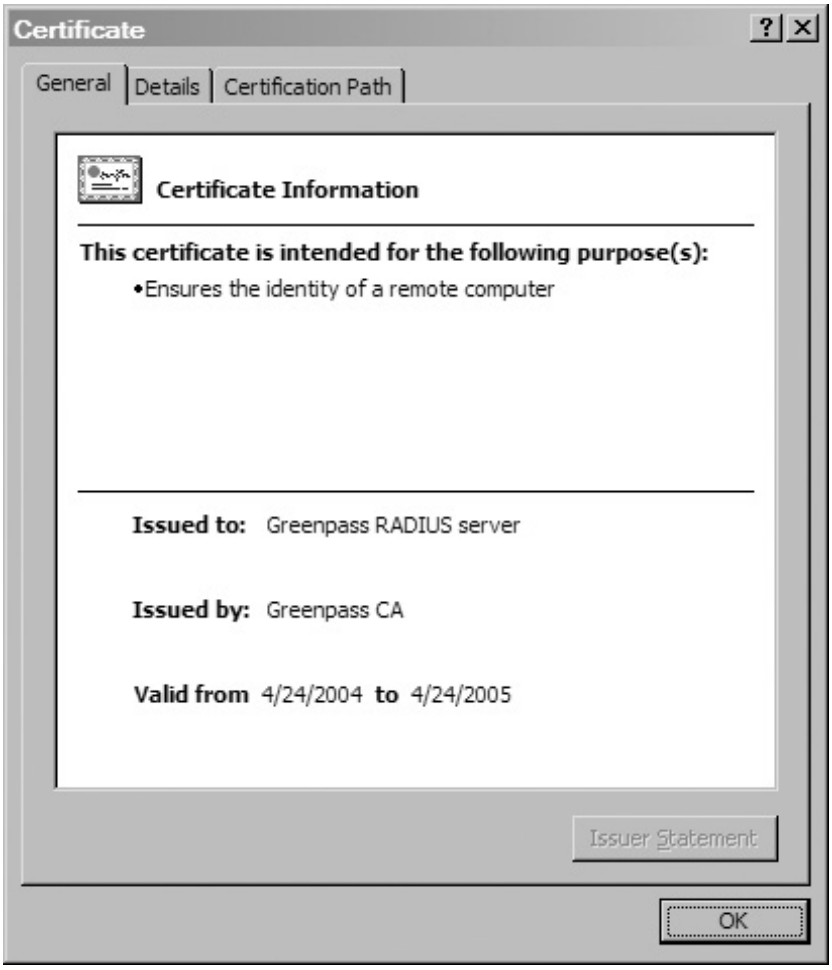


Figure 5-4: The RADIUS server's certificate, signed by the Greenpass CA.

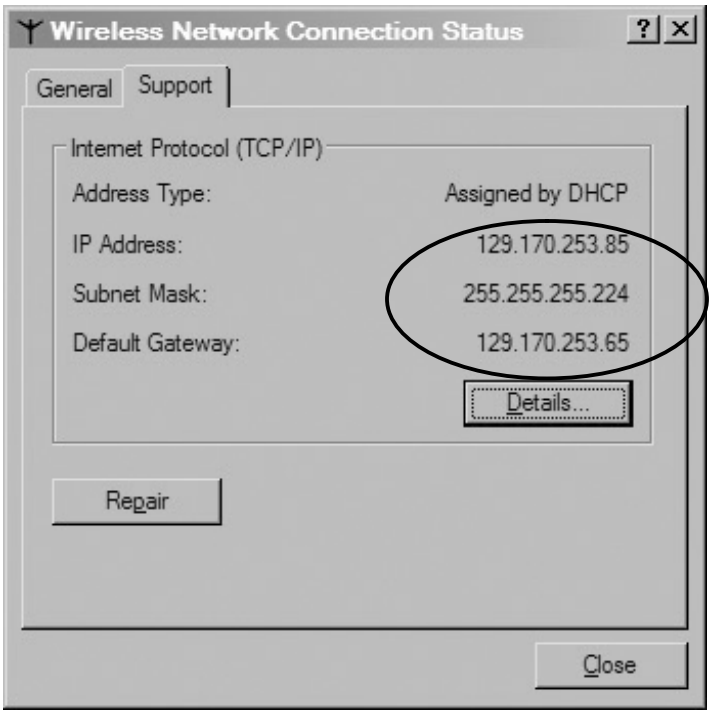
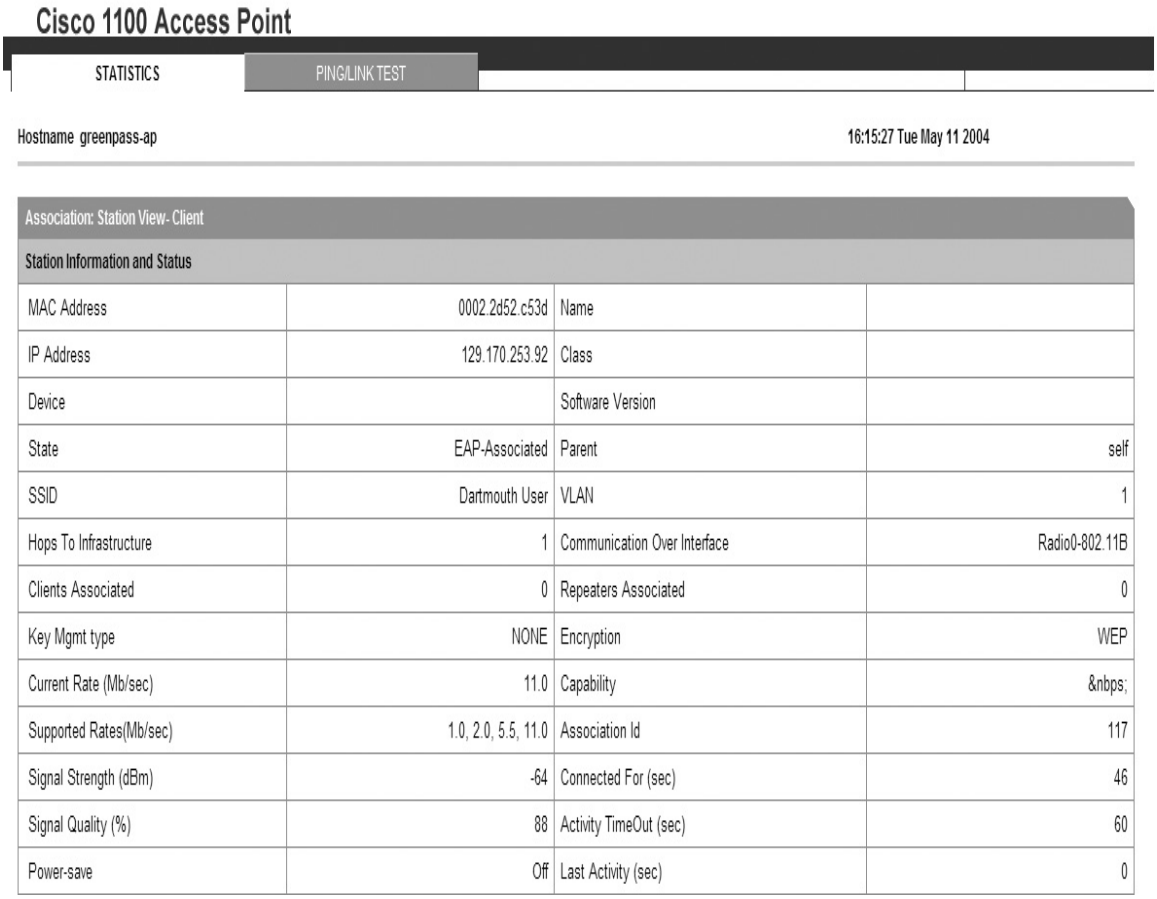


Figure 5-5: Wireless network connection status of an authenticated local user. Note that the supplicant has been assigned an IP address in the subnet of the Greenpass network.

It should be noted that configuring the client to use EAP-TLS authentication for Greenpass is a one-time setup. Once the wireless network (i.e., Dartmouth User SSID) is set up and the X.509 certificate used to authenticate to the network is selected, the authentication process is automatically performed on subsequent associations. Windows remembers which certificate to submit to the RADIUS server. Depending on the security settings, the server certificate needs to be verified only on the first attempt. Thus from the client's perspective, authentication is automatic after the initial attempt, with a short pause (during which the EAP-TLS authentication is taking place) being the only noticeable difference. For local users, authentication to Greenpass is seamless once the initial configurations are complete.



Cisco 1100 Access Point

STATISTICS | PING/LINK TEST

Hostname greenpass-ap 16:15:27 Tue May 11 2004

Association: Station View- Client

Station Information and Status

MAC Address	0002.2d52.c53d	Name	
IP Address	129.170.253.92	Class	
Device		Software Version	
State	EAP-Associated	Parent	self
SSID	Dartmouth User	VLAN	1
Hops To Infrastructure	1	Communication Over Interface	Radio0-802.11B
Clients Associated	0	Repeaters Associated	0
Key Mgmt type	NONE	Encryption	WEP
Current Rate (Mb/sec)	11.0	Capability	&nbps;
Supported Rates(Mb/sec)	1.0, 2.0, 5.5, 11.0	Association Id	117
Signal Strength (dBm)	-64	Connected For (sec)	46
Signal Quality (%)	88	Activity TimeOut (sec)	60
Power-save	Off	Last Activity (sec)	0

Figure 5-6: Status of an authenticated local user from the access point's association table. Note the values for IP address, SSID, and VLAN.

5.1.2 EAP-TLS for Mac Users

Authentication via EAP-TLS is somewhat different under the Mac platform due to implementation differences in the operating system. We tested 802.1x authentication under Mac OS X Panther (version 10.3) using Airport Software (version 3.3 or later). Panther is the first version of the Mac OS that has built-in support for 802.1x.

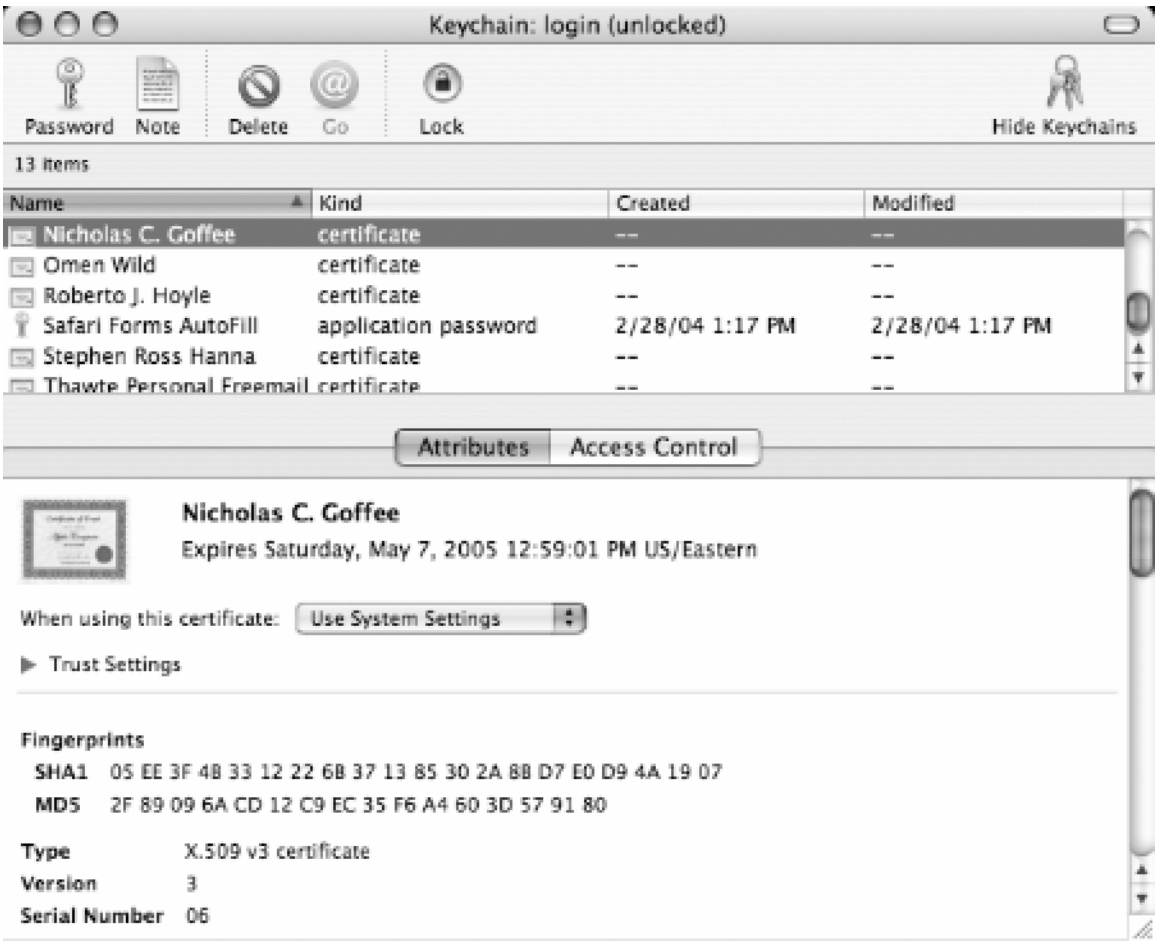


Figure 5-7: The Keychain Access window for the *login* keychain. The certificate we are interested in is the one issued to Nicholas C. Goffee. The other certificates must be moved to a temporary keychain for 802.1x authentication to use the appropriate certificate.

Mac users must manage their user certificates through the **Keychain Access** application that is included with Mac OS. With Keychain Access, users can add certificates and store passwords for a variety of applications. In particular, all users on a Mac have their own user keychain called *login* that is the default keychain used in authentication. To use a X.509 identity certificate for authentication, the certificate must be installed into the login keychain, and it must be the only certificate installed in that keychain (see Figure 5-7). Apple's 802.1x support is still in its early stages and hence not very elegant.

Once the X.509 certificate is installed in the keychain, Airport must create an 802.1x connection. Under the configuration settings, select *Airport* as the **Network Port**, "*Dartmouth User*" as the name of the **Wireless Network**, and uncheck all authentication types except for *TLS*. Figure 5-8 shows the configuration window for the 802.1x connection. Another implementation quirk in Airport is that users must supply a username and password even when using a certificate-based authentication method such as EAP-TLS. The username and password have no bearing on the authentication process since the X.509 certificate is used to verify the supplicant. Thus, supplying any values for the username and password will allow the EAP-TLS process to take place.

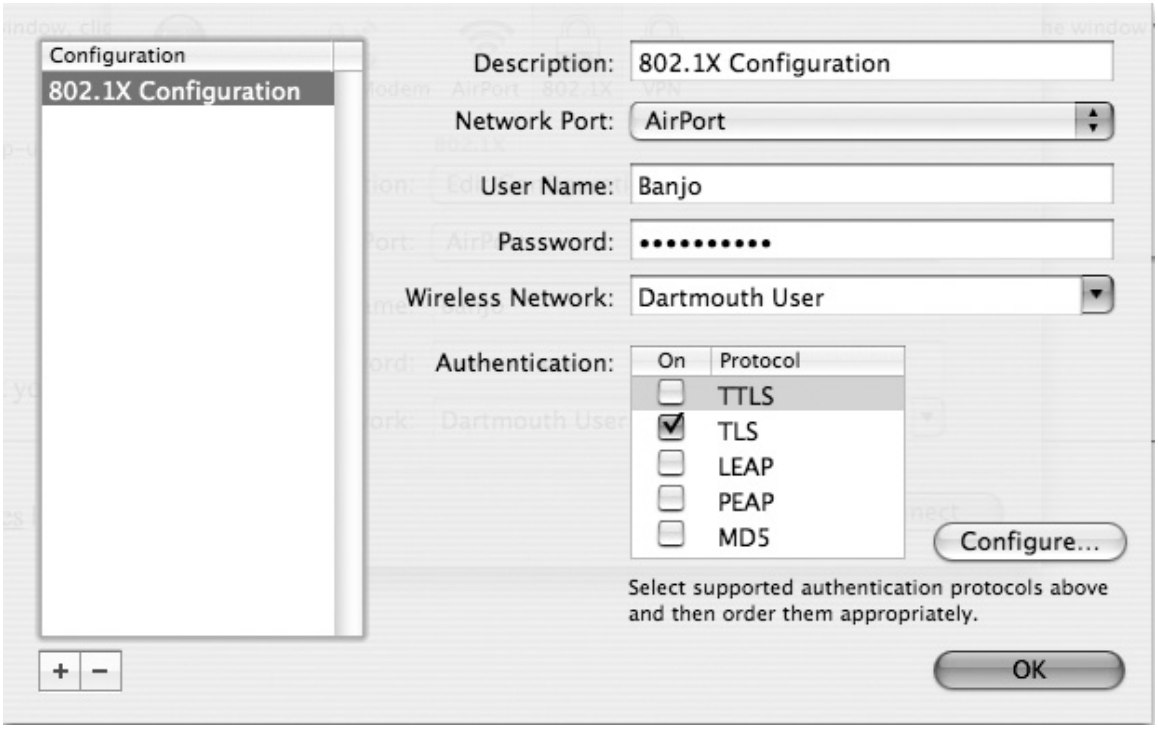


Figure 5-8: The 802.1x connection configuration screen. The wireless network is set to our Dartmouth User SSID and TLS is selected as the authentication method. Note that a dummy username and password must be included as well.

The supplicant should now be connected to Dartmouth User, signifying that they have passed authentication to the RADIUS server. This can be verified by checking that the assigned IP address is in the 129.170.253.64/27 range that is assigned to the Greenpass components.

5.2 Guest Users

As stated earlier, guest users must be delegated access to the network by an authorized delegator before connecting to the network. The delegation tools are available on the Web server residing on V_2 of our network. In order to access the delegation front page, the supplicant must associate to the V_2 SSID of our wireless network. Unlike the local user SSID, the guest SSID name is broadcast as "Greenpass Test." When in range of our access point(s), the supplicant will see Greenpass Test as an available network. No authentication is required for this SSID, and the supplicant needs only to select this network to connect. If the supplicant is already on a wireless connection, it may be necessary to disable or remove the current connection in order to associate to the Greenpass Test SSID.

Once the connection to Greenpass Test is made, the supplicant should be assigned an IP address on our 10.0.0.* private network by our DHCP server. It can be verified that along with the IP address, the DHCP server and the DNS server provided for the connection reside on 10.0.0.1, which is the private network IP address of our machine

running these services. Figure 5-9 shows the connection status and wireless network details for a Windows user associated onto V_2.

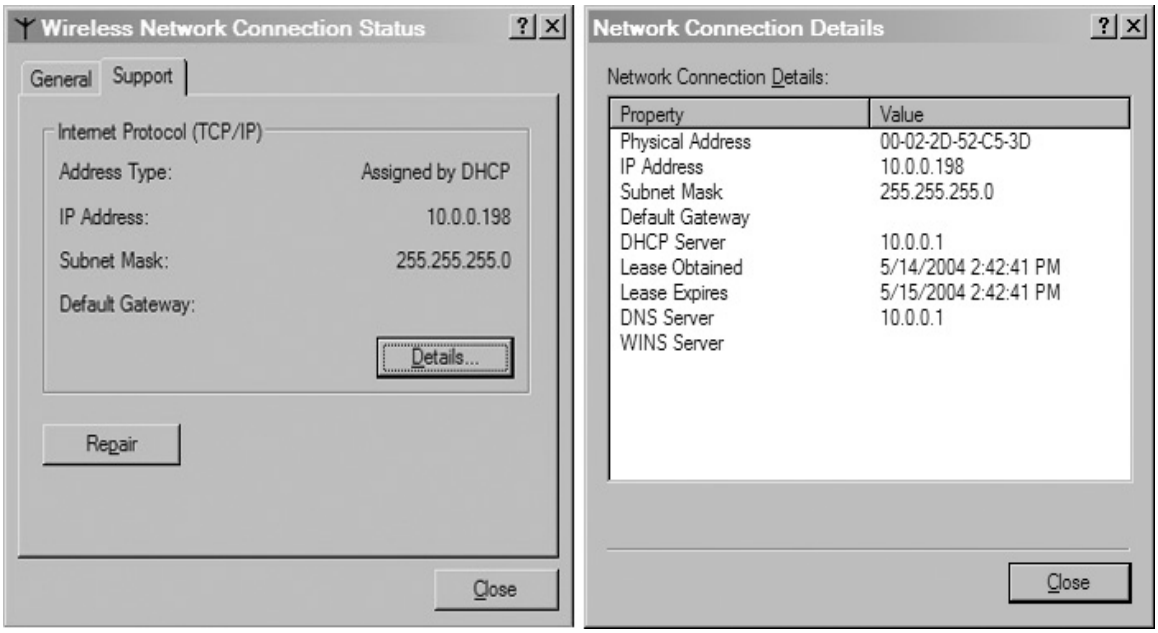


Figure 5-9: Network connection status and wireless network properties for a guest associated onto the guest VLAN SSID.

With the guest now associated onto V_2 and assigned an IP address on our private network, the guest can be delegated access to the network. Due to our modified DNS server, any hostname will be redirected to our Web server. When the guest opens up a Web browser page, she will automatically be sent to the delegation front page. Figure 5-10 shows a screenshot of the delegation front page. As explained in the previous chapter, the guest supplies her public key certificate as part of an SSL handshake, or a temporary certificate issued by our dummy CA is issued and used if she does not have a certificate.

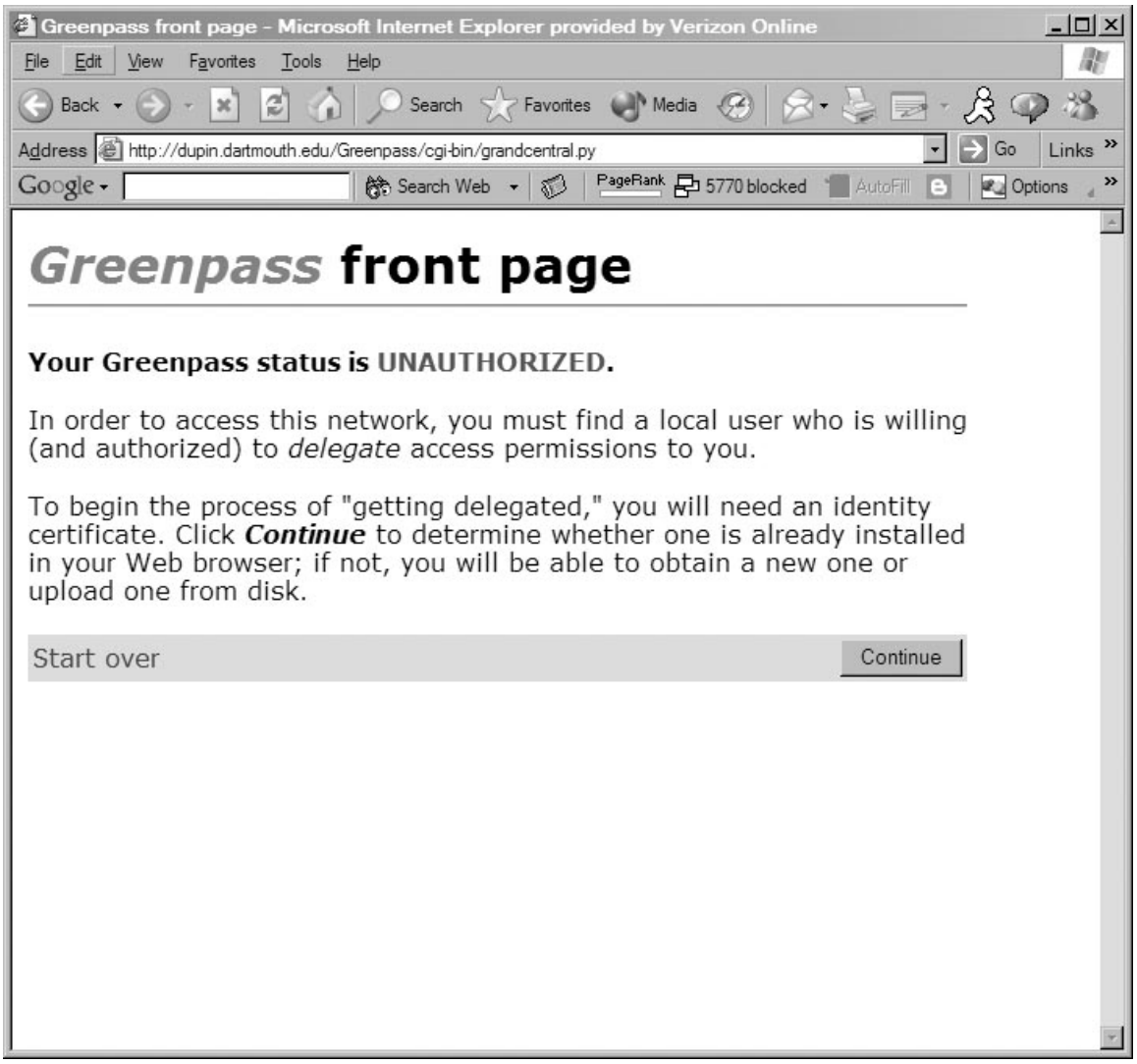


Figure 5-10: The delegation front page. When the guest first arrives at the site, her status will be "Unauthorized."

The guest uploads her public key certificate and receives an ID number for her request and sees a visual hash of her certificate's public key (Figure 5-11). The guest then must find a delegator who will check the guest's credentials, compare the visual hashes, and create and sign a SPKI/SDSI certificate for the guest. The entire SPKI/SDSI certificate chain is recorded as a HTTP cookie and placed in the guest's Web browser. Figure 5-12 displays the delegation page once the guest has been delegated access.

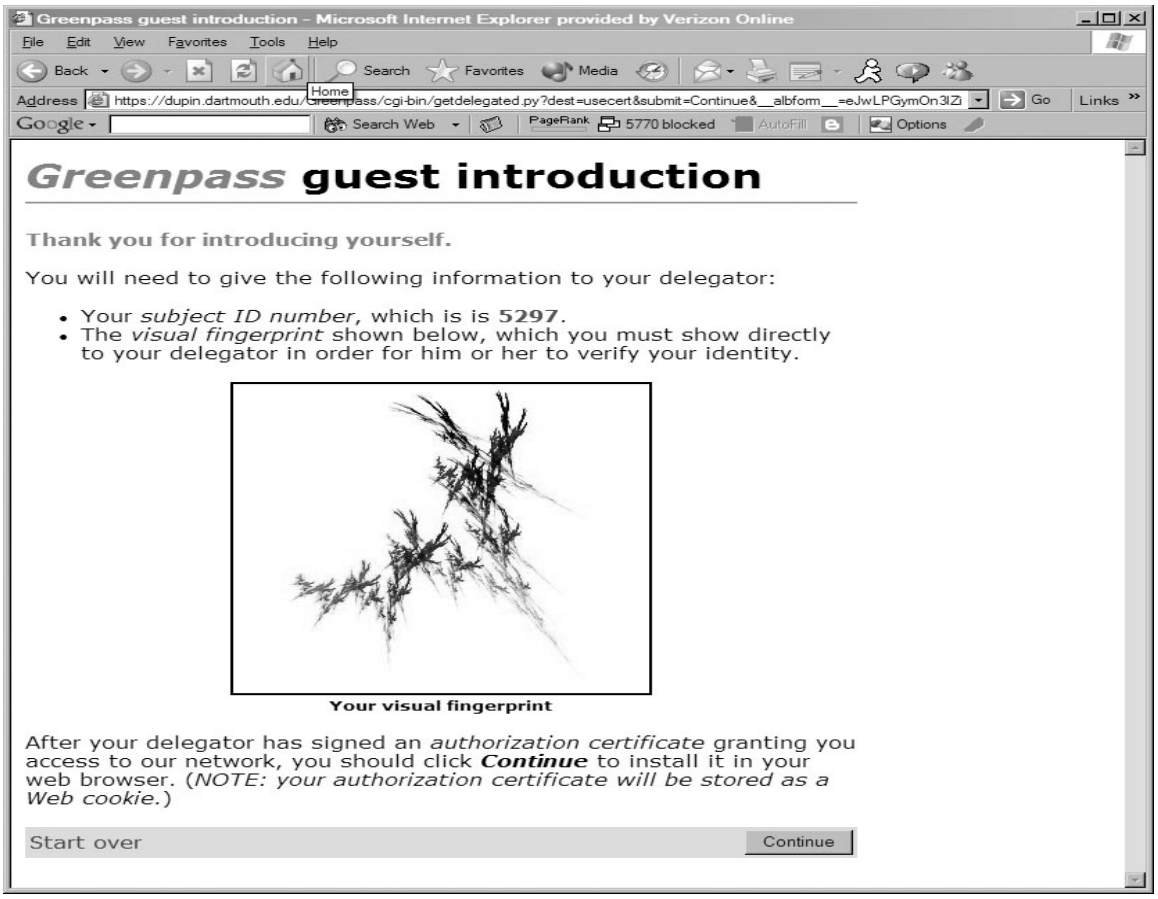


Figure 5-11: The visual hash of the guest's public key. The guest must now contact a delegator to create a SPKI/SDSI certificate for network access.

Figure 5-12: The delegation front page where the guest's status is now "Authorized User."

5.3 The Greenpass Pilot

In order to test the usability of Greenpass, we ran a small pilot where users unfamiliar with the system worked with the various parts of the project. In the first part of the pilot, participants were provided with X.509 identity certificates and authenticated to the system as local users. For the next phase, participants were directed to remove the local user certificates and authenticated to the RADIUS certificate after being delegated guest access. The final portion had some of the participants acting as delegators and granting temporary access to the other participants. A more detailed description of the procedure and results of the pilot can be found in [Pow04].

The majority of the problems occurring during the pilot occurred while setting up 802.1x authentication on the user machines. We did not have any external 802.1x client software available, thus users running older operating systems needed to use different machines. While most Windows users will have Windows 2000 or later, Mac OS X Panther (version 10.3) is still relatively new and not as widely used. This problem will diminish with time as new users will have updated operating systems and old users will eventually upgrade to a new operating system. Mac users had problems managing the Keychain application to install and move around their local user certificates. Users with multiple certificates had to remove all other certificates to a temporary keychain, which proved to be a task the participants were unfamiliar with. The Windows users in our pilot did not have as much trouble setting up 802.1x authentication, although some found the process confusing. Once 802.1x authentication was set up, the participants did not have any further problems authenticating to the RADIUS server.

The second potion of the pilot yielded few problems. The participants had already set up 802.1x authentication and succeeded in authenticating as local users. Some participants were unclear on how to connect to the guest SSID, but understood when the step was explained to them. Delegating access was a straightforward procedure as well, and the visual hash aided in verifying the guest certificates. In general, the participants seemed to understand the overall delegation process.

The pilot itself was not as comprehensive a test as we would have liked. We did not have enough participants to test the full range of operating systems and hardware. However, the results were promising in that the participants were able to authenticate to the RADIUS server as both a local user and an authorized guest. The majority of the problems that occurred could be attributed more to the client operating system and how they handle 802.1x authentication. Once the initial configurations were completed, the authentication and delegation processes were fairly straightforward. The pilot demonstrated that the system itself was fairly easy to comprehend, and the usability of the system was affirmed to us.

6. Future Work

Up to this point, our focus on the creation of Greenpass has been on getting a working model to experiment with. In the course of our discussion, several interesting ideas and prospects came to light, but unfortunately were placed on the back burner due to time restrictions. This chapter describes some of the modifications and applications for Greenpass that have been discussed and, for some applications, suggests how these ideas can be incorporated into Greenpass.

6.1 Integrating the Cisco ACS

In addition to the FreeRADIUS version of Greenpass, I spent time working with the *Cisco Secure Access Control Server* (version 3.2) running on a Windows Server 2000 machine. The Cisco ACS is a closed-source, proprietary version of a RADIUS server developed by Cisco. Under the Cisco ACS paradigm, the ACS provides authentication, authorization, and accounting, hence making it an *AAA server*. In this model, the NAS is considered to be an *AAA client*. Working with closed-source software was more difficult since the changes that could be made were severely limited. This section will explain how I set up the Cisco ACS to authenticate local users and discuss the alterations that need to be made to support our delegated guest access.

Authenticating local users using the Cisco ACS is not much different than using FreeRADIUS. The ACS is configured to accept EAP-TLS authentication, and the access point is told to consult the ACS for EAP authentication. The only additional requirement that the ACS enforces is that each local user must have a user account set up, and the name of the account must match the common name field of the X.509 identity certificate issued to the user. There is a protocol for handling unknown users, but this must be done through an external database, such as the Windows User Database. In order to maintain the local users internally (which seems to be the more desirable choice), a user account must be added to the internal Cisco database. This adds an extra administrative step when providing credentials to a new local user.

Authenticating guests will require changing our original model, however. In our FreeRADIUS version, I simply intercepted the EAP-TLS authentication process to run an extra check for a SPKI/SDSI certificate chain if the supplicant had a non-local X.509 identity certificate. With a closed-source piece of software, such a direct solution was not possible. I could not come up with an implementation of authenticating delegated guests analogous to what occurs in the FreeRADIUS model in the time I had left. Instead, I will present a blueprint for a possible solution to authenticate delegated guests.

The Cisco ACS provides a mechanism where an external database can be consulted in determining whether a supplicant should be granted access. If we could utilize one of the external databases to complete our external check for a SPKI/SDSI certificate chain, we can simulate the same procedure we use with FreeRADIUS. Among the external

databases supported by the Cisco ACS, the most promising possibility appears to be Generic *LDAP (Lightweight Directory Access Protocol)*. According to the Cisco ACS User's Guide [Cis03], the Cisco ACS forwards the username and password to the LDAP database, which then either passes or fails the authentication request. The ACS then takes the decision made by the LDAP database and instructs the AAA client to grant or deny access to the supplicant.

One advantage is that the hierarchical nature of LDAP can be used to retain the hierarchy between delegators and the guests they delegated to. Thus, any guest delegated to by a certain delegator would appear as a child node of that delegator in the LDAP directory. As the user base grows and the system is scaled to organize a larger number of delegators into separate departments, this solution is very appealing. It would be easy to keep track of which delegator granted access to which guest, a fact that will be very useful in accounting and revocation of permissions (see Section 6.4). We can have the LDAP directory maintain the public keys of the guests and delegators, and have a method of verifying ownership of the private key that is associated with the public key stored in the database. We can use SPKI/SDSI certificates or one of the alternate means of representing delegation discussed in Section 6.3 to implement the LDAP hierarchy of users.

Using a different external database to authenticate and authorize guests adds to the system that we have designed. It would be nice to maintain a single mechanism for authenticating both local and guest users. At the very least, the guest does not realize

there are two different authentication mechanisms at work. The guest presents credentials to the AAA client (NAS) and is either granted or denied access; the guest does not really care who made the decision, as long as the correct decision is made. And while maintaining two separate databases could be extra work, it could be advantageous to maintain local users and guests in separate databases.

6.2 Support for Alternate Authentication Methods

Our current version of Greenpass requires the supplicant to authenticate to the RADIUS server using EAP-TLS. This decision was made because EAP-TLS seemed to be the logical choice for the environment at Dartmouth, where a full-fledged PKI is already in place and X.509 identity certificates are already fairly prevalent among members of the Dartmouth community. At other institutions, however, lack of the necessary resources may make EAP-TLS an infeasible method of authentication.

Password-based authentication schemes are simpler to implement, but tend to be less secure due to the vulnerabilities inherent in the username-password model and have the added requirement of keeping shared secrets between the user and the server. Nevertheless, password-based authentication is still commonly used and support should be provided. Supporting password-based authentication schemes such as EAP-MD5 ([BSK04] provides a more detailed description of the various authentication protocols) is a simple matter of configuring the RADIUS server to accept such authentication methods. In FreeRADIUS and other available RADIUS servers, databases of usernames

and passwords are maintained. When a supplicant arrives and enters a username, the RADIUS server checks against its list of users and verifies that the password (or in the case of MD5, the MD5 hash of the password) matches the one listed for the username. While having the RADIUS server check a username-password is not difficult, this method requires a greater administrative role by some central authority because the database of users must be created and maintained. Furthermore, delegation using username-passwords is not as natural as in the certificate-model. In order to avoid placing all the work of granting guest access to a central administrator, some clever method of deriving guest passwords must be developed. It would be easy to simply add and delete accounts as guests arrive, but since this method requires direct access to some centralized database, distributing this privilege to too many delegators would create security holes.

A more recent trend in authentication protocols is the use of *tunneling* (creating a secure channel between the client and server) to protect the traffic created during the authentication process. To this effect, *EAP-TTLS (Tunneled TLS)* and Cisco's *PEAP (Protected EAP)* have gained popularity. Tunneled protocols use a two-step approach in which the client and server create a secure channel for communication in the first stage, and then use one of the standard authentication protocols in the second stage. Support for tunneled protocols is also implemented by most RADIUS servers by allowing the secure channel to be established between the supplicant and the server. The RADIUS server must then be configured to handle the authentication method used within the tunnel in the same manner as it is without the tunnel. Tunneled protocols are still a recent

development and there are only a few actual implementations of PEAP and EAP-TTLS. Most of the existing implementations use a password-based authentication method within the tunnel rather than a certificate-based one.

6.3 Alternative Methods of Guest Delegation

As mentioned earlier, SPKI/SDSI is by no means the only method of implementing delegated access. Certain characteristics of SPKI/SDSI made it an optimal choice for our initial implementation, but there are other methods for delegated authorization that merits closer investigation. I will discuss two of these alternative approaches in more detail and provide an outline for using these models to implement an authentication system with delegated guest access.

6.3.1 PERMIS and X.509 Attribute Certificates

PERMIS [COB03, Per, Cha02] is an example of a *Privilege Management Infrastructure (PMI)* that uses X.509 attribute certificates to bestow authorization onto the holder of the certificate. In this model, each user is assigned a role and the privileges that are associated with that role. Users are assigned to roles through *role-assignment ACs* while privileges are tied to roles through *role-specification ACs*. The PERMIS implementation consists of three main components: the authorization policy, the privilege allocator (PA), and the PMI application programming interface (API).

The authorization policy specifies the resources that each user has access to under what conditions. In other words, this is the overall blueprint that reflects the access control to resources and the assignment of roles to users. Policies are expressed through XML, and the authorization policy is implemented through a *document type definition (DTD)* that specifies the details, such as what roles are available, how roles are allocated to users, and the *source of authority (SOA)* that oversees this procedure.

The PA is used by the SOA to allocate privileges to users through the use of role-assignment ACs. Role-assignment ACs are stored and maintained in a LDAP directory, relying on the fact that ACs are signed by the issuing AA to provide tamper resistance.

The API provides the interface between the *access control enforcement function (AEF)* and *access control decision function (ADF)*. The AEF authenticates the user and queries the ADF if the user has the privileges to perform the requested actions on the requested target. If the user has the appropriate role ACs defined in the authorization policy, the ADF grants the request.

Implementing a PERMIS-based model of authentication and guest delegation would require a completely different setup. Instead using a RADIUS server and 802.1x authentication for local users, each local user would be defined a role that grants them access to the network. Since PERMIS uses hierarchical role-based access control, the roles are structured in such a way that a parent role has all the privileges granted to the child role. Following this model, we could define a *delegator* role and a subordinate *user*

role. Users are given the privilege to access the network and delegators are given the additional privilege of delegating user roles. Due to the hierarchy, delegators also have access to the network. The AEF and ADF defined by the API will carry out the role of the RADIUS server in authenticating users and authorizing requests (in this case, accessing the network). In the simplest model, a local user will have a role-assignment AC that states she is a "user" and a role-defining AC will state the users can access the network.

When a guest wants to gain access to the network, she will contact a delegator and use some method of supplying credentials. The delegator would then need to delegate a role-assignment AC to the guest, and have the AC signed by some AA. In order to make this model decentralized, each delegator would have to be delegated the same authority as an AA. Otherwise, the created AC must be sent off to a central AA who would sign it and send it back to the delegator.

There are foreseeable problems with this model of delegation. First, PERMIS is designed to adhere to a more centralized model, which conflicts with our goal of a decentralized solution. It may be possible to decentralize the mechanism by expanding the powers afforded to the delegators, but this could open up security holes. Second, there is no clear distinction between local users (non-delegators) and authorized guests. Even if we have distinct roles for local users and authorized guests, there is no practical difference between the two roles. We would need to somehow enforce a limited lifespan on the role that is assigned to authorized guests or implement some manner of revocation. Finally,

storing all the ACs for the entire user base into a database can lead to performance issues. As the size of the user base increases, a larger database must be searched whenever a user tries to authenticate to the network. While it is uncertain the extent of the performance hit that would occur, there seems to be a concern about scalability under this model.

6.3.2 KeyNote and PolicyMaker

KeyNote [BFIK99, Key] (and its predecessor PolicyMaker [BFL96]) is a trust-management system that utilizes signed assertions rather than public-key certificates to grant access to certain resources. A trust-management system contains five basic components: a language for describing *actions*, which are operations that are controlled by the system; a mechanism for identifying *principals*, the entities that can be authorized to perform actions; a language for specifying *policies*, which state what actions which principals are allowed to perform; a language for specifying *credentials*, which provide a mechanism for delegation of authorization; and a *compliance checker*, which makes the decision on the result of a request considering a policy and a set of credentials.

Actions are represented by name-value pairs organized into action attribute sets. The specifics of the name-value pairs or not set and can vary from application to application. Principals are identified by any string value, including public keys; thus we have a system in which a supplicant's public key can be the principal rather than a name. An assertion can be thought of as a statement of the conditions under which the asserting principal authorizes actions requested by others. A principal identified by its public key

can sign an assertion and the resulting credential performs a role analogous to public key certificates in the traditional PKI model.

In this model, the compliance checker acts as our decision-making entity. Applications consult the compliance checker by issuing it a query that contains a proposed action attribute set and the requesting principal. The compliance checker consults an ordered set of possible responses to select the appropriate response (called the "policy compliance value"). This value can either be a Boolean value that simply accepts or rejects the request or a range of pre-determined values (such as "delegator," "local user," "authorized guest," and "unauthorized guest").

It seems possible to leave the RADIUS server to authenticate local users via EAP-TLS, while deferring decisions on guests to the compliance checker. In order to implement this model, we would have the public keys of delegators and unauthorized guests to be the principals involved. We would need to establish a policy where delegators (or actually, their public key) can sign assertions that grant guests (via the holder of the public key in question) access to the network. The action involved here would be access to the network. Once the guest authenticates to the delegator, the delegator would sign an assertion that authorizes the guest's public key to access the network. Meanwhile, we would require the pre-defined "POLICY" principal that acts as the root of trust to issue assertions that allow delegators to grant access to guests. When the RADIUS server is contacted by a guest user, the RADIUS server would send a query to the compliance checker containing the appropriate access attribute set (that establishes the chain of

assertions from the root of trust to the supplicant), along with the guest's public key (as the requesting principal). The compliance checker can respond with a simple Boolean accept/reject response or with a more sophisticated set of answers (such as which VLAN to place users on).

Clearly the KeyNote model ties in better to our original Greenpass model since local users are unaffected. There are, however, a few potential problems with this implementation. A lot of work needs to be done to implement the model explained above: the initial policy and access attribute set must be created, the semantics of the access attribute name-value pairs must be defined, and the class of compliance value must be decided upon. While the assertion/action model provides greater flexibility than SPKI/SDSI certificates, the assertions can grow to be quite complex and difficult to manage. Furthermore, revoking assertions in KeyNote can become a major hassle.

6.4 Accounting and Revocation

An important aspect of maintaining a secure authentication system is keeping track of who does what and allowing for the revocation of privileges. As stated earlier, the RADIUS protocol includes accounting packets that allow the NAS to request the RADIUS server to log client activity. Each client is assigned a session ID number and the traffic concerning a certain session is tagged by that ID number. The accounting functionality records useful information about the client such as the username, SSID and

VLAN, and the IP address of the NAS that initiated the request. Figure 6-1 shows an example of the accounting entries created for an authenticated local user.

```
Tue May 11 16:09:42 2004
        Acct-Session-Id = "00000D83"
        Called-Station-Id = "0040.96a1.328e"
        Calling-Station-Id = "0002.2d52.c53d"
        Cisco-AVPair = "ssid=Dartmouth User"
        Cisco-AVPair = "nas-location=unspecified"
        Cisco-AVPair = "connect-progress=Call Up"
        Acct-Authentic = RADIUS
        User-Name = "Sung Hoon Kim"
        Acct-Status-Type = Start
        NAS-Port-Type = Wireless-802.11
        Cisco-NAS-Port = "373"
        NAS-Port = 373
        Service-Type = Framed-User
        NAS-IP-Address = 129.170.253.67
        Acct-Delay-Time = 0
        Client-IP-Address = 129.170.253.67
        Acct-Unique-Session-Id = "9388ceebcc31d7f4"
        Timestamp = 1084306182

Tue May 11 16:14:05 2004
        Acct-Session-Id = "00000D83"
        Called-Station-Id = "0040.96a1.328e"
        Calling-Station-Id = "0002.2d52.c53d"
        Cisco-AVPair = "ssid=Dartmouth User"
        Cisco-AVPair = "nas-location=unspecified"
        Cisco-AVPair = "vlan-id=1"
        Cisco-AVPair = "auth-algo-type=eap-tls"
        Acct-Authentic = RADIUS
        Cisco-AVPair = "connect-progress=Call Up"
        Acct-Session-Time = 263
        Acct-Input-Octets = 111899
        Acct-Output-Octets = 415652
        Acct-Input-Packets = 828
        Acct-Output-Packets = 1052
        Acct-Terminate-Cause = Lost-Carrier
        Cisco-AVPair = "disc-cause-ext=No Reason"
        User-Name = "Sung Hoon Kim"
        Acct-Status-Type = Stop
        NAS-Port-Type = Wireless-802.11
        Cisco-NAS-Port = "373"
        NAS-Port = 373
        Service-Type = Framed-User
        NAS-IP-Address = 129.170.253.67
        Acct-Delay-Time = 0
        Client-IP-Address = 129.170.253.67
        Acct-Unique-Session-Id = "9388ceebcc31d7f4"
        Timestamp = 1084306445
```

Figure 6-1: Accounting entries provided for a successfully authenticated local user. Note that the client is placed on the "Dartmouth User" SSID and is placed on VLAN 1.

Although the functionality of accounting is already implemented in the RADIUS protocol, there are some adjustments that need to be made to work effectively with our Greenpass modifications. First, the source of the username field provided in the accounting logs differs depending on which operating system the client is running. Under Windows, the username is supplied by the X.509 identity certificate provided for EAP-TLS authentication to the RADIUS server. Mac users, however, need to input any username and password in order to initiate 802.1x authentication, even when using a public-key certificate scheme such as EAP-TLS. I observed that for Mac users, the username recorded in the accounting logs is the username the client enters to initiate the authentication process. Since this username-password has no meaning whatsoever, Mac users can supply any username to be recorded in the logs. We want Mac users to have the username in their X.509 certificate recorded in the accounting files as well.

In the case of authorized guests, there is also a problem with logging the username. Even if the username taken from the guest's X.509 identity certificate is logged, this information is not necessarily important. The guest's certificate was issued by another institution, so the guest's common name entered in the certificate does not necessarily have much meaning at our institution. Furthermore, the guest's authority comes from possessing knowledge of the private key that corresponds with the public key provided in the certificate, not from the name contained in the certificate. For guests, it is more meaningful to keep track of the public key of the guest's certificate and the identity of the delegator that granted the guest access. The delegator has met the guest in question in order to complete the delegation process, and presumably verified the guest's identity and

credentials in some manner. Thus having this information recorded would be more useful in tracking down an authorized guest than simply having the entry listed in the common name field of the guest's identity certificate.

Revocation is another interesting problem that needs to be addressed. Local users are taken care of through standard *Certificate Revocation Lists (CRL)* that provides a list of certificates that have been revoked. The authentication server is provided with an updated CRL and checks against it to make sure the supplicant's certificate has not been revoked. On the guest side, certificates have short expiration periods since guest access is short-lived. If a guest's network access needs to be revoked, the SPKI/SDSI certificate vouching for the guest's public key can simply be removed from the repository. Again, there needs to be some record maintained of the map between a guest's identity and her public key. We can look up the guest's public key and search for the SPKI/SDSI certificate speaking for that key. This step needs to be performed manually, but revocation should be entrusted to some sort of administrative figure and hence this is an acceptable solution. There must also be some sort of revocation list that prevents the guest from being delegated to again after her first credentials are revoked.

Revoking the credentials of a delegator is a more complex problem. If a delegator leaves the institution, it is important to revoke the delegating authority that was entrusted to him (revoking network access can be handled by a CRL since the delegator is a local user). Removing the SPKI/SDSI certificate vouching for the delegator's public key would prevent further delegation of network access from that public key. Once that certificate is

removed, however, all the guests who he delegated to will also have their access revoked since there is no longer a chain of valid certificates leading up to a trusted root. Since the delegator's SPKI/SDSI certificate is not a leaf on the chain, removing the certificate will have an effect on all the guests that he delegated to.

The obvious solution is to have all the guests delegated to by the revoked delegator resubmit their request for network access to be handled by another delegator. This is not the most desirable option, but it is the simplest and not too difficult to manage. Revocation of a delegator's credentials should be a very rare occurrence, so the inconvenience caused to the guests will occur infrequently enough to be excusable. It is important to balance the delegation load placed on each of the delegators. Thus if a delegator's guests must be reauthorized, only a proportional number of the guests will be affected. Again, guest access is short-lived, thus at any time there should not be too many guests affected by this procedure. In order to keep the system scalable, the number of delegators should increase with the number of authorized guests at any given point. Keeping a proportional number of delegators and balancing out the load evenly across the delegators will make this scenario less cumbersome. Also, it is again vital to keep track of which delegator granted network access to each guest.

Note that a problem arises only if the delegator is stripped of the authority to delegate. A temporary restriction on network access due to the delegator's machine being compromised does not pose any additional problems in terms of revocation. We can temporarily place the delegator's local certificate on the CRL to prevent access to the

network. If the security breach did not affect our guest delegation process, the delegator's SPKI/SDSI credentials can remain in the system so that his guests still have access. Since the delegator is denied access to the network, he will not be able to delegate to new guests. Guests previously delegated to should not have been affected, and thus their credentials are untouched. If the nature of the security breach compromised the guest delegation process, we can remove the delegation credentials as well to invalidate the appropriate guest credentials.

6.5 Supporting Other Devices

Currently, laptop computers constitute the majority of access to wireless networks. This fact is rapidly changing as new technology offers a variety of devices that utilize the wireless network. PDAs with wireless network access are the most common example of alternative devices. More cutting edge products such as Cisco's new Voice *over IP (VoIP)* handset device and Vocera's device for WiFi voice communication are being tested at Dartmouth.

Naturally we want to be able to support authentication for these devices as well. In the case of PDAs, simpler methods of authentication are already supported. Thus in theory authenticating PDAs via Greenpass to authenticate to the wireless network is a simple process. There is a concern caused by the fact that PDAs possess less processing power and are subject to power consumption restrictions due to battery life. To deal with these limitations, decreasing the processing load required to authenticate to the network would

be a worthwhile cause. Furthermore, PDAs tend to be more mobile, and thus a supplicant using a PDA is more likely to associate to several different access points within a single session. It would be necessary to devise an elegant way of handing off an authenticated client among access points so that the supplicant does not need to authenticate to each new access point she associates to.

There are some interesting ways to use these other devices for delegation of authority as well. One of the applications discussed involved a "delegation stick" (RFID tag reader) that can read the identity off a student's ID card (with embedded RFID tag). A delegator then can create a SPKI/SDSI certificate granting a student access to a certain resource (such as entry into a building).

6.6 Finely Grained Definition of Authority

Our current version of Greenpass takes an all-or-nothing approach in granting or rejecting access to supplicants; the supplicant is either allowed or not allowed to connect to the network. There are a few ways in which we can define a more finely grained definition of network access.

Recall how the current setup involves two VLANs, one for authorized users and one for unauthorized guests. It is a simple matter of configuring the user profiles to associate a supplicant onto different VLANs. If the campus network is organized in a manner where different classes of resources are placed on different VLANs, we can simply assign users

to the appropriate VLAN to grant them the desired level of access. The different resources can be organized in a hierarchical structure (such as administrators and students) or partitioned into non-intersecting groups (as in different departments). In terms of the RADIUS server, these modifications are trivial. The real work lies in setting up the physical network to place the appropriate resources onto the desired VLANs.

Another method of finer-grained access control is the use of a policy-based scheme. For example, KeyNote can be used to have different applications (such as e-mail clients, Web browsers, and FTP clients) each containing different access restrictions. If we want guests to access only e-mail, we can establish a credential that allows principals who are guests to check e-mail. We prevent access to the Internet by not defining a similar credential to Web traffic. Policy-based authentication requires no hardware modifications to be made, but requires a lot of effort on the software side. The desired policies must be defined through numerous assertions and credentials. Furthermore, more finely-grained access control in this model requires the policy to be checked every time the user requests access to a different type of resource.

6.7 Push Authorization

In the push model of authorization, the supplicant pushes the necessary credentials to the authentication server for evaluation. Storing the guest's certificate chain into a HTTP cookie and uploading it into the Web browser is a good step towards supporting the push

model. Whenever the guest visits the delegation main page after being delegated access, the Web browser pushes the cookie with the certificate chain. The Web server then evaluates the contents of the cookie and determines the status of the guest.

Continuing this approach, we would like to make the process adhere completely to the push model. Instead of the RADIUS server pulling up the SPKI/SDSI certificate that corresponds to the guest's public key, it would be more elegant if the guest could submit both the X.509 identity certificate and the SPKI/SDSI chain. An alternative method could be inducing the guest's Web browser to automatically authenticate to the local user SSID once the delegation front page reads the cookie and determines that the guest is authorized. In this scenario, the guest pushed her credentials to the Web server via the HTTP cookie, and the request is initiated with the RADIUS server without any action on the part of the guest (the RADIUS server would still need to refer to its repository of SPKI/SDSI certificates to verify the chain).

Alternatively, having authorized guests skip authentication to the RADIUS server altogether gets us closer to a push model of authorization. Local users can still use EAP-TLS with their X.509 certificate to authenticate to the RADIUS server. Authorized guests can submit their SPKI/SDSI chain through a HTTP cookie to the Web server. If the RADIUS server could contact an outside authority to verify the guest, then we could have the external authority grab the SPKI/SDSI chain and make the decision. The Cisco ACS provides for this mechanism where the RADIUS server defers to an external database to check for the supplicant's credentials. The drawback here is that local users

are authenticated through one method, while authorized guests are judged through a different process.

6.8 No PKI

One of the obstacles to public-key based authentication schemes is the fact that such schemes require the institution to have a PKI in place. For most institutions with small user populations, this is not a feasible solution; establishing a PKI is expensive and inefficient for small user bases. Removing the requisite centralized component could redefine the concept of public-key authentication and allow such methods to be a more attainable solution.

In our Greenpass model, the division between the local users possessing X.509 identity certificates and the authorized guests having SPKI/SDSI authority certificate chains is fairly arbitrary. Both classes of users are granted the same access to the same network, are granted the authority to do so from some "higher" authority, and are not allowed to propagate this authority. We can abstract this model to the scenario where there is only one local user (and delegator) and everyone else is an authorized guest. Theoretically, PKI-based certificates follow this model: the CA is the local user whose authorization does not expire (or at least for a very long time) and can grant authority to others, while the users are granted the authority to resources for as long as they are part of the institution.

We can distribute the load of delegation by granting the authority to delegate access to certain individuals. So now we still have the one local user (who acts as the CA), a few authorized guests whose SPKI/SDSI certificates allow them to propagate authority (the delegators), and the rest of the authorized guests who cannot propagate authority (the normal local users). We do not need a centralized component to vouch for the identity of a supplicant if there is already an alternate method of authentication. For example, all new students at Dartmouth could be issued a key pair and signed SPKI/SDSI certificate by one of the delegators. The student's presence and participation during orientation provides authentication that the student is part of the Dartmouth population. From this point, the decision to grant a student access to the network is determined by the fact that she is a student and not by the student's name. The fact that there is a SPKI/SDSI chain that leads to the student's public key and that the student has the private key that corresponds to that public key is enough to grant the student access. A situation similar to this scenario already occurs at Dartmouth when incoming freshmen register for classes. Each student is assigned to an advisor who possesses a PIN number for the student. The advisor meets with the student, discusses course options, and gives the PIN number to the student, which enables the student to sign up for courses.

In this scenario, we do not need a highly centralized PKI to be in place. A stripped-down dummy CA like the one that is currently used to generate a key pair for guests without a X.509 certificate can be used to create key pairs for all the local users. Delegators can be organized into more complex hierarchical structures to further distribute the role of delegation in a manner that is more representative of the institution in question. All this

is achieved through the use of SPKI/SDSI authorization certificates, which are simpler and easier to process than traditional PKI certificates.

There are a few flaws in this scenario, however. There is no concept of delegation depth inherent in the SPKI/SDSI model. The propagation of authority is an all-or-nothing choice and the depth of this propagation cannot be specified. For example, you cannot create a SPKI/SDSI certificate that states that the subject may propagate authority, but the subject of that delegated authority may not propagate authority. In order to create a structured hierarchy of delegation, it is important to closely guard how delegation of authority can be carried out.

Revocation of authority can be a hassle as well. Some sort of revocation list must be maintained for end users. When a student is expelled, her SPKI/SDSI certificate chain should no longer grant her access to the network. If the certificates are maintained in a repository, this simply involves the removal of the appropriate certificate. Using a push model involving HTTP cookies, however, the student has control over the credentials, so there must be some way of invalidating her certificate chain before the expiration specified in the certificate. Furthermore, if a delegator leaves the college, then the delegator's certificate should no longer be valid. If we remove the delegator's certificate, however, all users that he delegated to will have their authority revoked as well. Without the delegator's SPKI/SDSI certificate, there is no longer a chain of certificates that lead to the trusted root. Hence there needs to be some method of reauthorizing the affected users in this case.

Despite the flaws in this scenario, the concept of removing the PKI is an interesting one that merits further exploration. This could provide an alternative for institutions that do not have the resources to set up and maintain a full-fledged PKI and thus could make public-key based authentication a more feasible option.

7. Conclusion

In this thesis I presented the Greenpass project, focusing on the process of authenticating users to the network. We added a step to the EAP-TLS authentication process to perform an external authorization check based on SPKI/SDSI certificates for authorized guests. Local users can be authenticated and authorized via traditional EAP-TLS. SPKI/SDSI is a lightweight solution that removed the need for a cumbersome central authority. Adding to the existing X.509-based PKI made the solution more flexible and prevented the need for users to install additional client software than what is provided by updated versions of the common operating systems.

The primary goal of the Greenpass project was to implement a method of delegated guest access that naturally and accurately depicts how the propagation of authority flows in the real world. SPKI/SDSI is the best option in this endeavor and had the added benefit of resulting in a more decentralized model of delegation. Local users are unaffected by the guest delegation aspect of the project, and authentication to the system is mostly automatic after the initial configuration is complete. Guests are afforded a similarly unobtrusive authentication process once a simple delegation process is complete. In this manner I feel that our goal was accomplished.

My work on the Greenpass project involved working with the server-side components of the system, focusing on modifying the RADIUS server to handle our check for authorized guests. Maintaining the various components of the project entailed setting up network

equipment, configuring various network services, and familiarizing myself with the current wireless and wired network security technology.

There are many directions this work can take in the future. As technology evolves and newer devices and authentication standards are introduced, we want our solution to evolve along with the technology. We can use software to enforce more finely-grained authorization policies. By experimenting with different definitions of the delegator and guest roles and redefining the responsibilities of the trusted root, it is possible to break away from the monolithic PKI and to create a more dynamic and decentralized alternative that can adapt to the requirements of the current environment. Greenpass introduces a method of using lightweight public-key schemes along with traditional PKI and investigates the possibility of redefining PKI into something that is decentralized, flexible, and feasible to implement.

8. Glossary of Terms

802.11 – family of IEEE specifications developed for wireless LAN technology.

802.11i – IEEE draft that defines the standard for future 802.11 security.

802.1Q – IEEE standard that addresses limiting multicast and broadcast traffic within a desired VLAN. This standard introduces inserting VLAN membership information into Ethernet frames.

802.1x – standard designed to secure wireless LANs that follow the IEEE 802.11 standard. 802.1x generalizes *EAP* and provides an authentication framework or wireless LANs.

Authentication Server – server responsible for authenticating (determining the identity of) supplicants. Some common examples of authentication servers are RADIUS and Kerberos.

CA (Certification Authorities) – root of authority responsible for issuing certificates that vouch for the identity of local users.

Delegator – special class of user that is authorized to grant temporary access to guests in our Greenpass model. Delegators possess a SPKI/SDSI authorization certificate that has the delegation field set to true that allows propagation of that authority.

EAP (Extensible Authentication Protocol) – an extension of the Point-to-Point Protocol that acts as a general protocol for authentication. EAP supports multiple authentication methods and provides the framework within which a supplicant and an authentication server communicate.

EAP-TLS – variation of the EAP protocol that uses *TLS* as the authentication method. Local users in Greenpass authenticate via EAP-TLS, and guests will use a modified version of this protocol.

KeyNote – a trust-management system that uses signed assertions to grant access to desired resources.

NAS (Network Access Server) – component in 802.1x authentication that is responsible for initiating the authentication request to the authentication server and provides service to the supplicant upon approval. Most commonly, the access point will act as the NAS.

Permis – a privilege management infrastructure that uses X.509 attribute certificates to assign roles to users. Privileges are assigned to roles, thus a user assigned to a certain role will gain the privileges specified to that role.

RADIUS (Remote Authentication Dial In User Service) **server –** the authentication server used in the Greenpass system. As the name suggests, the RADIUS protocol was originally intended for authenticating dialup users, but more recently used in other settings (including the wireless LAN).

SPKI/SDSI (Simple PKI/Simple Distributed Security Infrastructure) **authorization certificate –** Lightweight certificate format used to authorize the subject to access resources. SPKI/SDSI certificates are expressed as s-expressions of five fields (subject, issuer, delegation, validity dates, and authorization). SPKI/SDSI certificates can propagate the ability contained in them to the subject of the certificate.

SSID (Service Set ID) – the identifying name of the wireless network. A wireless network can broadcast its SSID so that users can see its existence or not broadcast the SSID and require only those who know the name to connect to the network.

Supplicant – the client in the 802.1x authentication model. The supplicant must be authenticated by the authentication server in order to gain access to desired resources.

TLS (Transport Layer Security) – a public-key authentication scheme that provides for mutual authentication of the client and server. TLS can be embedded within the EAP protocol and is analogous to SSL.

VLAN (Virtual LAN) – a software solution to protecting resources on a network. Resources residing on a VLAN behave as if they were located on the same physical network even if they are not. VLANs provide great flexibility as users assigned to a particular VLAN can gain access to resources on the VLAN regardless of where the connection is made.

WEP (Wired Equivalent Privacy) – a basic scheme designed to encrypt the session between the access point and the wireless supplicant. Several design flaws make WEP an incomplete solution to secure wireless traffic.

WPA (WiFi Protected Access) – a standard designed by the WiFi Alliance to address the weaknesses of WEP. WPA offers stronger data encryption and also provides for client authentication through the use of EAP.

X.509 attribute certificate – a standard certificate format that grants authorization to the holder of the appropriate X.509 identity certificate.

X.509 identity certificate – a standard certificate that vouches for the identity of the holder of the certificate. X.509 certificates are issued by a CA that certifies that the holder of the certificate belongs to a certain institution. The holder of the certificate proves ownership of the certificate by displaying knowledge of the private key associated with the public key contained in the certificate.

9. References

[AS99] Bernard Aboba and Dan Simon. PPP EAP TLS Authentication Protocol. IETF RFC 2716, October 1999.

[BFIK99] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The Keynote Trust-Management System Version 2. IETF RFC 2704, September 1999.

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1996.

[BSK04] Kwang-Hyun Baek, Sean Smith, and David Kotz. A Survey of WPA and 802.11i RSN Authentication Protocols. Computer Science Technical Report (to appear), Dartmouth College.

[BV98] Larry J. Blunk and John R. Vollbrecht. PPP Extensible Authentication Protocol (EAP). IETF RFC 2284, March 1998.

[CEE+01] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos and Ronald L. Rivest. Certificate Chain Discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4): 285-322, 2001.

[CG02] Oscar Canovas and Antonio F. Gomez. A Distributed Credential Management System for SPKI-Based Delegation Scenarios. In *Proceedings of the 1st Annual PKI Research Workshop*, April 2002.

[Cha02] David W. Chadwick. An X.509 Role-based Privilege Management Infrastructure. In *Business Briefing: Global InfoSecurity 2002*, World Markets Research Centre, Ltd.

[Cis98] Cisco IOS VLAN Services, January 1998. `http://www.cisco.com/warp/public/614/11.html.`

[Cis99] Cisco VLAN Roadmap, April 1999. `http://www.cisco.com/warp/public/538/7.html.`

[Cis03] User Guide for Cisco Secure ACS for Windows Server Version 3.2. `http://www.cisco.com/univercd/cc/td/doc/product/access/acs_soft/csacs4nt/acs32/user/32winug.pdf.`

[Cla01] Dwaine E. Clarke. SPKI/SDSI HTTP Server / Certificate Chain Discovery in SPKI/SDSI. Master's thesis, Massachusetts Institute of Technology, September 2001.

[COB03] David W. Chadwick, Alexander Otenko, and Edward Ball. Role-Based Access Control with X.509 Attribute Certificates. In *IEEE Internet Computing*, March-April 2003.

[DE02] Steve Dohrmann and Carl M. Ellison. Public-Key Support for Collaborative Groups. In *Proceedings of the 1st Annual PKI Research Workshop*, April 2002.

[EA03] Jon Edney and William A. Arbaugh. *Real 802.11 Security.* Addison-Wesley, 2003.

[EFL+98] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. SPKI Examples. IETF Internet Draft, draft-ietf-spki-cert-examples-01.txt, March 1998.

[EFL+99a] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. Simple Public Key Certificate. IETF Internet Draft, July 1999.

[EFL+99b] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. SPKI Certificate Theory. IETF RFC 2693, September 1999.

[FH02] Stephen Farrell and Russell Housley. An Internet Attribute Certificate Profile for Authorization. IETF RFC 3281, April 2002.

[Free] The FreeRADIUS Server Project. `http://www.freeradius.org`.

[GKS+04] Nicholas C. Goffee, Sung Hoon Kim, Sean Smith, Punch Taylor, Meiyuan Zhao, and John Marchesini. Greenpass: Decentralized, PKI-based Authorization for Wireless LANs. In *Proceedings of the 3rd Annual PKI Workshop*, May 2004.

[Gof04] Nicholas C. Goffee. Greenpass Client Tools for Delegated Authorization in Wireless Networks. Master's thesis, Dartmouth College, June 2004 (expected).

[IEEE98] IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks. IEEE Std 802.1Q-1998, December 1998. `http://standards.ieee.org/getieee802/download/802.1Q-1998.pdf`.

[IEEE01] IEEE Standard for Local and Metropolitan Area Networks: Port-based Network Access Control. IEEE Std 802.1X-2001, October 2001. `http://standards.ieee.org/getieee802/download/802.1X-2001.pdf`.

[Key] Keynote home page. http://www.cis.upenn.edu/~keynote/.

[Mor98] Alexander Morcos. A Java Implementation of Simple Distributed Security Architecture. Master's thesis. Massachusetts Institute of Technology, May 1998.

[Naz03] Sidharth Nazareth. SPADE: SPKI/SDSI for Attribute Release Policies in a Distributed Environment. Master's thesis. Dartmouth College, May 2003.

[Per] Permis home page. http://www.permis.org/.

[Pow04] Kimberly Powell. Testing the Greenpass Wireless Security System. Undergraduate senior thesis. Dartmouth College, June 2004 (expected).

[Rig00] Carl Rigney. RADIUS Accounting. IETF RFC 2866, June 2000.

[Ros03] Ken Roser. HOWTO: EAP-TLS Setup for FreeRADIUS and Windows XP Supplicant, February 2003. http://3w.denobula.com:50000/EAPTLS.pdf.

[RWC00] Carl Rigney, Ward Willats, and Pat R. Calhoun. RADIUS Extensions. IETF RFC 2869, June 2000.

[RWRS00] Carl Rigney, Steve Willens, Allan C. Rubens, and William Allen Simpson. Remote Authentication Dial In User Service (RADIUS). IETF RFC 2865, June 2000.

[SDS] Java Implementation of SPKI/SDSI 2.0. http://theory.lcs.mit.edu/~cis/sdsi/sdsi2/java/SDSI_Java_Intro.html.

[SGK+04a] Sean Smith, Nicholas C. Goffee, Sung Hoon Kim, Punch Taylor, Meiyuan Zhao, and John Marchesini. Greenpass: Flexible and Scalable Authorization for Wireless Networks. Computer Science Technical Report TR2004-484, Dartmouth College, January 2004.

[Sul02] Adam Sulmicki. HOWTO on EAP/TLS authentication between FreeRADIUS and XSupplicant, April 2002. http://www.missl.cs.umd.edu/wireless/eaptls/.

[ZLR+00] Glen Zorn, Dory Leifer, Allan Rubens, John Shriver, Matt Holdrege, and Ignacio Goyret. RADIUS Attributes for Tunnel Protocol Support. IETF RFC 2868, June 2000.