Greenpass: Decentralized, PKI-based Authorization for Wireless LANs[*]

Nicholas C. Goffee, Sung Hoon Kim, Sean Smith, Punch Taylor, Meiyuan Zhao, John Marchesini

Department of Computer Science/Dartmouth PKI Lab[†]
Dartmouth College
Hanover, New Hampshire USA

Abstract

In Dartmouth's "Greenpass" project, we're building an experimental system to explore two levels of authorization issues in the emerging information infrastructure. On a practical level, we want to enable only authorized users to access an internal wireless network—while also permitting appropriate users to delegate internal access to external guests, and doing this all with standard client software. On a deeper level, PKI needs to be part of this emerging information infrastructure—since sharing secrets is not workable. However, the traditional approach to PKI—with a centralized hierarchy based on global names and heavy-weight X.509 certificates—has often proved cumbersome. On this level, we want to explore alternative PKI structures that might overcome these barriers.

By using SPKI/SDSI delegation on top of X.509 certificates within EAP-TLS authentication, we provide a flexible, decentralized solution to guest access that reflects real-world authorization flow, without requiring guests to download nonstandard client software. Within the "living laboratory" of Dartmouth's wireless network, this project lets us solve a real problem with wireless networking, while also experimenting with trust flows and testing the limits of current tools.

1 Introduction

Dartmouth College is currently developing *Greenpass*, a software-based solution to wireless network security in large institutions. Greenpass extends current wireless security frameworks to allow guest access to an institution's wireless network and selected internal resources (as well as to the guest's home system).

This project, which enhances EAP-TLS authentication with SPKI/SDSI-based authorization decisions, is a novel, extensible, feasible solution to an important problem.

- Our solution is **seamless.** Guests can potentially access the same access points and resources that local users can. The same authorization mechanism can apply to local users, and can also be used for application-level and wired resources.
- Our solution is also **decentralized**: it can accommodate the way that authorization really flows in large academic organizations, allowing designated individuals to delegate network access to guests.

Although we are initially targeting universities, Greenpass may apply equally well in large enterprises.

This paper. This paper provides a snapshot of the current state of our project. Section 2 reviews the problem we seek to solve, and Section 3 reviews the existing wireless security standards we build upon. Section 4 presents weaknesses in some current attempts to secure wireless networks. Section 5 presents our approach: Section 6 discusses the delegation piece, and Section 7 discusses the access decision piece. Section 9 discusses future directions, and Section 10 offers some concluding remarks.

More lengthy discussions (e.g., [Gof04, Kim04]) of this

[*]A previous version of this paper appeared in Dartmouth Dept. of Computer Science technical report TR2004-484

[†]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. The authors can be reached via addresses of the form `firstname.lastname@dartmouth.edu`

work will appear this Spring.[1]

2 The Problem

Wireless network access is ubiquitous at Dartmouth, and we see a future where a lack of wireless network access at a university—including access for visitors to the campus—is as unthinkable as a lack of electricity. Many institutions, however, want to restrict access to their wireless networks for several reasons: the cost of providing network bandwidth and resources; the credibility or liability hit the institution may incur should an outside adversary use the network to launch an attack or spam; the ability to block users who have not installed critical security patches; and the ability (for reasons of license as well as levels-of-protection) to restrict certain local network resources to local users.

Access to a wired network often depends, implicitly, on the physical boundaries of the network. Most establishments do not install courtesy network jacks on the outside walls of their buildings: a standard door, therefore, fulfills most access-control needs. Wireless network traffic, on the other hand, travels on radio waves, extending the network's physical boundaries. Access control and encryption must be designed into the link layer or higher to prevent unauthorized use and/or eavesdropping.

This future raises some challenges:

- We need to permit authorized local users to access the network.
- We also need to permit selected guests to access the network.
- We must minimize the hassle needed to grant access to guests, and we must accommodate the decentralized ways that authority really flows in large organizations.
- The security should cause little or no additional effort when regular users and guests use the network.
- The type of guests and the manner in which they are authorized will vary widely among the units within an institution.
- We must accommodate multiple client platforms.
- The solution must scale to large settings, more general access policies, and decentralized servers.
- The solution should also extend to *all* authorization—wired or wireless, network or application, guest or intra-institution.
- The solution must be robust against a wide range of failures and attacks.

We have encountered several definitions of "guest access" to a wireless network, many of which differ substantially from our own. Two basic definitions we have seen are as follows:

- **Definition 1**. The trivial solution: the network is open and all passersby, even uninvited ones, can potentially become "guests."
- **Definition 2**. Insiders can connect to a *VPN* (*virtual private network*) or to the inside of a firewall, allowing them to access private resources. Guests have basic wireless access—perhaps with a bridge to the Internet—but remain outside the firewall or VPN.

Our requirements for guest access, on the other hand, are as follows:

- **Definition 3.** We want guests to access the inside; that's the whole point. But we also need to control who becomes a "guest," and we want to permit authorization to flow the way it does in the real world: we don't want a centralized authority (or a central box and rights system purchased from a single vendor) controlling all end-user decisions.

3 Background

Wireless networking comes in two basic flavors. In the *ad hoc* approach, the wireless stations (user devices) talk to each other; in the *infrastructure* approach, wireless stations connect to access points, which usually serve as bridges to a wired network. We are primarily interested in infrastructure networking. Understanding the numerous protocols for access control in infrastructure mode requires wading through an alphabet soup of interrelated standards and drafts. We will provide a brief overview of these standards in this section; Edney and Arbaugh's recent book [EA03] explores them thoroughly.

Rudimentary access control to a WLAN could be implemented by requiring users to enter the correct *SSID* for the access point they are trying to connect to, or by accepting only clients whose MAC addresses appear on an *access-control list* (*ACL*). Both these techniques are easily defeatable, as we discuss below in Section 4.

Wired equivalent privacy (WEP) is a link-layer encryption method offered by the original IEEE 802.11 wireless Ethernet standard. WEP is based on a shared secret between

[1] As of press time, the cited theses have been published as technical reports.

the mobile device and access point. WEP has numerous flaws, which Cam-Winget et al. [CWHWW03] and Borisov et al. [BGW01] discuss in detail.

Wi-Fi Protected Access (WPA) is a stronger authentication and encryption standard released by the Wi-Fi Alliance, a consortium of vendors of 802.11-based products. WPA is, in turn, a subset of *802.11i*, the IEEE draft that standardizes future 802.11 security. WPA provides an acceptable security standard for the present, until 802.11i is finalized and becomes widely supported.

WPA and 802.11i both use *802.1x* [CS01], a general access-control mechanism for any Ethernet-based network. 802.1x generalizes the *Extensible Authentication Protocol (EAP)* [BV98], originally designed for authentication of PPP dialup sessions.

In a wireless context, 802.1x access control works as follows:

- By trying to connect to an access point, a mobile device assumes the role of *supplicant*.
- The access point establishes a connection to an *authentication server*.
- The access point (which, in 802.1x terminology, assumes the role of *authenticator*) relays messages back and forth between the supplicant and the authentication server. These relayed messages conform to the EAP packet format.
- EAP can encapsulate any of a variety of inner authentication handshakes including challenge-response schemes, password-based schemes (e.g., Cisco's LEAP), Kerberos, and PKI-based methods. The supplicant and authentication server carry out one of these handshakes.
- The authentication server decides whether the supplicant should be allowed to connect, and notifies the access point using an *EAP-Success* or *EAP-Failure* message.

EAP-TLS. Authenticating a large user space suggests the use of public-key cryptography, since that avoids the security problems of shared secrets and the scalability problems of ACLs. One public-key authentication technique permitted within EAP is *EAP-TLS* [AS99, BV98].

TLS (*Transport Layer Security*) is the standardized version of *SSL (Secure Sockets Layer)*, the primary means for authentication and session security on the Web. In the Web setting, the client and server want to protect their session and possibly authenticate each other. Typically, SSL/TLS allows a Web server to present an X.509 public key certificate to the client and prove knowledge of the corresponding private key. A growing number of institutions (including Dartmouth) also exploit the ability of SSL/TLS to authenticate the client: here, the client presents an X.509 certificate to the server and proves ownership of the corresponding private key. The server can use this certificate to decide whether to grant access, and what Web content to offer. An SSL/TLS handshake also permits the client and server to negotiate a cryptographic suite and establish shared secrets for session encryption and authentication.

The EAP-TLS variant within 802.1x moves this protocol into the wireless setting. Instead of a Web client, we have the supplicant; instead of the Web server, we have the access point, working in conjunction with the authentication server.

Our approach. WPA with EAP-TLS permits us to work within the existing Wi-Fi standards, but lets the supplicant and access point evaluate each other based on public key certificates and keypairs. Rather than inventing new protocols or cryptography, we plan to use this angle—the expressive power of PKI—to solve the guest authorization problem.

4 Black Hat

As part of this project, we began exploring just how easy it is to examine wireless traffic with commodity hardware and easily-available hacker tools. Right now:

- We can watch colleagues surf the Web and read their email.
- We can read the "secret" (non-broadcast) SSID for local networks.
- We can read the MAC addresses of supplicants permitted to access the network.
- We can tell our machine (Windows or Linux) to use a MAC address of our own choosing (such as one that we just sniffed).

The lessons here include the following:

- We can easily demonstrate that security solutions that depend on secret SSIDs or authenticated MAC addresses do not work.
- The current Dartmouth wireless network is far more exposed than nearly all our users realize; the paradigm shift from the wired net has substantially changed the

security and privacy picture, but social understanding (and policy) lags behind. We suspect this is true of most wireless deployments.

We conjecture that any solution that does not use cryptography derived from entity-specific keys will be susceptible to sniffing attacks and session hijacking.

5 The Overall Approach

We have already built a basic prototype of Greenpass, and are planning a series of pilots in the near future. Our prototype consists of two basic tools. The first automates the process of issuing credentials to a guest by allowing certain local users to issue *SPKI/SDSI authorization certificates* [EFL+99a, EFL+99b] to guests. The second tool is a *RADIUS (Remote Authentication Dial In User Service) server* [Rig00, RWC00, RWRS00] that carries out a standard EAP-TLS handshake for authentication, but has been modified to consult SPKI/SDSI certificates for authorization of non-local users. Neither tool requires users to have software beyond what is typically installed on a Windows laptop (covering most of our user space); other platforms need 802.1x supplicant software, which is provided with recent versions of Mac OS X and is readily available for Linux.

Authorization in real life. In the physical world, a guest gets access to a physical resource because, according to the local policy governing that resource, someone who has the power to do so said it was OK. In a simple scenario, Alice is allowed to enter a lab because she works for Dartmouth; guest Gary is allowed to come in because Alice said it was OK. Gary's authorization is decentralized (Dartmouth's President Wright doesn't know about it) and temporary (it vanishes tomorrow). More complex scenarios also arise in the wild: e.g., Gary may only have access to certain rooms, and it must be Alice (and not Bob, since he doesn't work on that project) who says OK.

For a wireless network in a large institution, the decision to grant authorization will not always be made by the same Alice—and may in fact need to reflect policies and decisions by many parties. PKI can handle this, by enabling verifiable chains of assertions.

Authorization in EAP-TLS. EAP-TLS specifies a way for a RADIUS server to *authenticate* a client, but leaves open the specification of *authorization*. Often, a RADIUS server will allow any supplicant to connect who authenticates successfully—i.e., whose certificate was signed by a CA the RADIUS server has been configured to trust. This approach does not adequately reflect real-life authorization flow as just described. Alice can see to it that Gary, her guest, obtains access to the wireless network, but she must do so by asking a central administrator to issue Gary an X.509 certificate from Dartmouth's own CA. It is possible for the RADIUS server to trust multiple CAs, such as those of certain other universities, but this option remains inflexible if a guest arrives from an institution not recognized by the existing configuration. (Another option that merits further investigation, however, is the possibility of linking RADIUS servers using more advanced trust path construction and bridging techniques. One such implementation is the Trans-European Research and Education Networking Association's (TERENA) [TER] top-level European RADIUS server, a hierarchy of RADIUS servers connecting the Netherlands, the UK, Portugal, Finland, Germany, and Croatia.)

Conceivably, a RADIUS server could perform any of a number of authorization checks between the time that a supplicant authenticates successfully and the time that the RADIUS server transmits an EAP success or failure code. In other words, we can modify a RADIUS server to base its decision on some advanced authorization scheme. Policies could be defined by policy languages such as XACML; or by signed *authorization certificates* as defined by *Keynote* [BFIK99] and its predecessor *PolicyMaker* [BFL96], by the *X.509 attribute certificate* (*AC*) standard [FH02], or by SPKI/SDSI.

SPKI/SDSI. For our Greenpass prototype, we settled on SPKI/SDSI for three main reasons: (1) it focuses specifically on the problem we are trying to solve (authorization), (2) its central paradigm of *delegation* gives us precisely the decentralized approach to guest access we desire, and (3) its lightweight syntax makes it easy to process and to code for.

SPKI/SDSI differs from a traditional X.509-based PKI in three important ways:

- SPKI/SDSI uses public keys as unique identifiers: people and other *principals* are referred to as holders of particular public keys, rather than as entities with particular names.
- A SPKI/SDSI certificate binds an authorization directly to a public key, rather than binding a name to a public key (as an X.509 certificate does) or an authorization to a name (as an ACL entry or attribute certificate typically does).
- Any person or entity, not just a dedicated CA, can potentially issue a SPKI/SDSI certificate. In particular,

the recipient of a SPKI/SDSI authorization can optionally be authorized to *delegate* his privilege to further users.

SPKI/SDSI therefore solves some of the problems with guest authorization. First, even if a guest's home organization issued him an X.509 certificate, we cannot use it to authenticate the guest (i.e., bind the guest to a unique identifier) if the issuer of the certificate is not trusted. A SPKI/SDSI authorization certificate, however, binds an authorization to a particular keyholder without an intermediate naming step: therefore, we can bind credentials to a guest's public key. The public key acts as the sole identifying information for the guest ("authentication," then, means proving ownership of the key).

Additionally, SPKI/SDSI delegation provides a straightforward way to implement guest access. Dartmouth can issue Alice, a professor, a SPKI/SDSI certificate granting her the ability to delegate access to guests.[2] If Alice invites Gary to campus to deliver a guest lecture, he will probably request access to the network. Alice can simply issue Gary a short-lived SPKI/SDSI certificate (vouching for the public key in his X.509 certificate) that grants him access to the network while he is on campus. No central administrator need be contacted to fulfill Gary's request.

Alternative approaches to authorization. Other approaches to delegated guest access are available, and each has its own balance of advantages and disadvantages.

An X.509 AC can grant a short-lived authorization to the holder of a particular X.509 identity certificate, in much the same way as we use SPKI/SDSI. Attribute certificates address the problem of authorization, but are intended to be issued by small numbers of *attribute authorities* (*AAs*); the attribute certificate profile [FH02] states that chains of ACs are complex to process and administer, and therefore recommends against using them for delegation. Doing so would amount to emulating SPKI/SDSI's functionality using attribute certificates. If standard WPA clients were able to transmit attribute certificates along with identity certificates as part of the EAP-TLS handshake, we might have chosen ACs instead of SPKI/SDSI certificates, as the former would have provided a convenient means to transmit authorization information to a RADIUS server.

Two other authorization certificate systems worth considering are *PERMIS* [COB03, PER] and X.509 *proxy certificates* [TWE$^+$03, WFK$^+$04]. The PERMIS system uses attribute certificates to specify roles for various users; members of some roles are able to delegate their role, or a subordinate role, to other individuals. PERMIS is worth investigating as a means of wireless guest access, although it might require modification to eliminate its reliance on global names. X.509 proxy certificates provide a different means of delegation than SPKI/SDSI does, along with a concept of temporary, relative identities that local users could provide to their guests. Proxy certificates conform closely enough to the X.509 name certificate format that they might work directly in an EAP-TLS handshake.

We also could have implemented guest access by placing temporary ACL entries in a central database. "Delegation" could be implemented by allowing authorized delegators to modify certain portions of the ACL. Ultimately, however, would like to support a "push" model of delegation where guests carry any necessary credentials and present them upon demand, allowing us to further decentralize future versions of Greenpass (see Section 9). Decentralizing authorization policies using signed certificates also eliminates the need for a closely-guarded machine on which a central ACL is stored.

We chose SPKI/SDSI because it reflects, in our minds, the most straightforward model of real-world delegation. A thorough comparison of alternative approaches would provide a worthwhile direction for future work.

6 Delegation

Assume that a new guest arrives and already holds an X.509 identity certificate containing a public key. In order to obtain wireless connectivity, the guest must obtain a SPKI certificate that conveys the privilege of wireless network access directly to his own public key.

To obtain this certificate, the guest will find a local user who can delegate to him (e.g., the person who invited him in the first place). This *delegator* must then learn the guest's public key. This step requires an information path from the guest's machine to the delegator's. Once the delegator learns the guest's key, he can issue a SPKI certificate with the guest as its subject.

We also need a way to ensure that the key the delegator authorizes to use the wireless network is really the key held by the guest. Otherwise, an adversary might inject his own public key into the communication channel between guest and delegator, tricking the delegator into authorizing the adversary instead of the intended guest. Dohrmann and Ellison describe a nearly identical problem in *introducing* the members of a collaborative group to one another [DE02]. Their

[2] In our current scheme, Alice uses an X.509 certificate, signed by the local CA, to gain access to the network herself; she must obtain a SPKI/SDSI certificate only if she needs to delegate to a guest without a locally-signed X.509 certificate.

solution was to display a *visual hash* of the public key being transferred on both the keyholder's device and the recipient's device: this allows the recipient to quickly compare the two visual images, which should appear identical if and only if the recipient received a public key value identical to the one stored on the originating device. We adopted this same approach; further details are given below.

Guest interface. We chose to use a Web interface to allow a guest to introduce his public key to a delegator. Web browsers are ubiquitous: we can safely assume that any user who wishes to access our network will have a Web browser installed. This technique gives us an advantage over, e.g., infrared transfer, wired tranfer, or passing of some storage medium, all of which might be incompatible with certain client devices.

Both our delegation tool and our modified RADIUS server rely on the observation that standard X.509 certificates, and standard SSL/TLS handshakes (including EAP-TLS) perform three functions that we need:

- An X.509 certificate contains the value of its owner's public key.
- An SSL/TLS handshake *presents* an X.509 certificate (and thus the owner's public key value).
- An SSL/TLS handshake, if it succeeds, also *proves* that the authenticating party owns the private key corresponding to the subject public key in the presented X.509 certificate.

With this observation, it becomes clear that, if the guest's Web browser supports SSL/TLS client authentication, then he can present his public key value to a Web site using this functionality.

When a guest arrives he must connect to our Web application to present his existing X.509 certificate. Therefore, he must obtain *some* wireless connectivity even before he is authorized. We are experimenting with various ways to enable this by creating an "unauthorized" VLAN (for newly-arrived guests) and an "authorized" VLAN (for local users and authorized guests); we present this approach in more detail in Section 7.

When a guest connects to our Web application, he will see a welcome page helping him through the process of presenting his certificate. We handle three situations at this point:

- If the guest's Web browser presents an SSL client certificate, we allow the option of presenting it immediately.
- We also allow the guest to upload his certificate from a PEM-formatted file on his local disk. (Browser and OS keystore tools usually allow a user to export his X.509 certificate as a PEM file. Since the purpose is to transfer the certificate to another user, a PEM file typically does *not* contain the user's private key.)
- If the guest does not already have a keypair and certificate, he can connect to a "dummy" CA page (separate from the main Dartmouth CA) that lets him generate a keypair and obtain a temporary X.509 certificate. (This should not be a standard approach, because a proliferation of client keypairs impairs usability. Note that the sole purpose of the dummy CA is to get the guest a keypair—we are therefore exempt from standard CA worries such as securing the registration process and protecting the CA's private keys.)

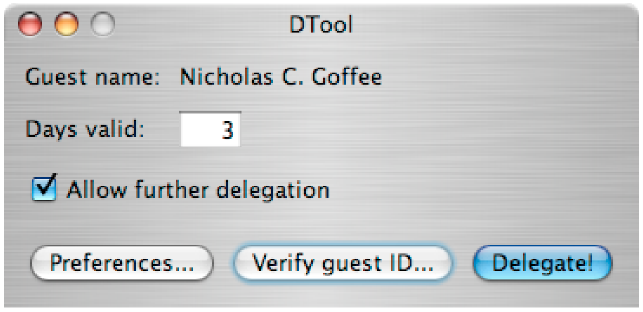
We implemented the guest interface using simple CGI scripts served by an Apache Web server. Our installation of Apache includes *mod_ssl*, which we configure to request (but not require) SSL client authentication. (We had to set a seldom-used option in *mod_ssl* that forces Apache to accept the guest's certificate even if it was signed by an unknown CA. Our purpose here is to learn a stranger's public key, not to authenticate a known user.) Therefore, if the guest has installed a client certificate in his Web browser, it will present it to our Web server. Our CGI scripts use OpenSSL to process the guest's X.509 certificates.

The dummy CA uses the standard enrollment functionality included in Web browsers that support SSL client authentication. The guest visits the CA page and enters (possible fake) identifying information. The page includes code that, when he submits the identifying information, causes his Web browser to generate a keypair, store the private key, and submit the public key to our Web server. The dummy CA then issues a new X.509 certificate back to the guest's Web browser, which stores it in its keystore. We support both Internet Explorer and Netscape/Mozilla enrollment methods.

After the guest presents his X.509 certificate by one of the above methods, our Web server generates a visual hash of it using the *Visprint* [Gol, Joh] program. (This program tranforms the MD5 hash of an object into an image using IFS fractals.)

After the guest uploads his certificate using one of the above methods, our Web server stores it in a temporary repository from which the delegator can retrieve it.

Delegator interface. A delegator first visits the same Web server as the guest, and searches for the guest's X.509 certificate by entering pieces of identifying information such as



Guest name: Nicholas C. Goffee
Days valid: 3
☑ Allow further delegation

Figure 1: A screenshot of our delegator tool (a trusted Java applet, shown running under Mac OS 10.3). Before delegation, the delegator has to verify the identity of the guest's public key using a visual hash comparison. Also notice the inputs for validity interval and whether or not to allow further delegation.

the guest's name and organization. After this step, the delegator verifies the certificate's authenticity (by comparing visual hashes) and constructs and signs a SPKI certificate.

Signing a SPKI certificate is problematic, because it requires access to the delegator's private key. A private key must be well-protected so that adversaries cannot use it to sign data that did not actually originate from the owner. Software usually signs data of a very specific type (email, Word documents, authentication challenges, certificates) to prevent misuse of the key.

We therefore needed to build a special software tool for signing SPKI certificates. We considered a number of alternative ways to implement this, including a custom application which delegators would have to download, but for the prototype, we settled on using a trusted Java applet (screenshot shown in Figure 1). Trusted applets are hashed and signed by an entity that the user of the applet trusts, ensuring that the applet has not been modified to do anything the signing entity did not intend. Sun's Java plugin for Web browsers, by default, gives trusted applets greater privileges than standard applets, including the ability to access the local filesystem on the client machine. (It is not unreasonable to have local users install a trust root certificate for the applet signer ahead of time.) Our applet can, therefore, load the delegator's private key from a local file[3] and, after prompting for a password to decrypt the key, use it to sign a SPKI certificate.

Our Web server generates a page with a reference to the delegation applet, and provides the guest's PEM-encoded certificate as an argument to the applet. The applet uses standard Java cryptography functionality to extract the public key from this certificate, and uses a Java SPKI/SDSI library from MIT [Mor98] to construct and sign a SPKI certificate that delegates wireless access privileges to the guest. The applet allows the delegator to specify a validity interval for the new certificate and choose whether or not the recipient should be able to delegate further. We have ported the Visprint code from C to Java so we can build the visual hash verification step into the applet as well.

[3]The applet prompts the delegator to choose an appropriate keystore file the first time it is run, and saves its location to a local preferences file for future signing sessions. We currently support PKCS12 keystore files. In the future, we would like to support various platforms' OS keystores.

7 Making the Decision

We now consider the process by which our modified RADIUS decides whether to admit users.

7.1 The decision process

Local users. In the initial case, local users show authorization (via EAP-TLS) by proving knowledge of a private key matching an X.509 identity certificate issued by the local CA. Once the TLS handshake succeeds, the supplicant is granted access. On most platforms, the supplicant must choose which certificate to submit only on the first successful attempt; the machine will remember which certificate to use on subsequent attempts, making the authentication process transparent to local users.

Guests. Authorized guests also authenticate via EAP-TLS using an X.509 certificate. (In this case, "authentication" consists only of proving knowledge of the private key, since we cannot trust the certificate's naming information.) The RADIUS server uses a different process, however, to decide whether the user is authorized. It must find a valid SPKI/SDSI certificate chain originating from a principal it trusts that ultimately grants access privileges to the supplicant's public key.

In preliminary sketches, we also involved the delegator's X.509 certificate, but that does not seem to be necessary. As a consequence, the delegator doesn't necessarily need to have a centrally-issued X.509 identity certificate; we consider this further in Section 9.

The algorithm. Putting it all together, the modified RADIUS server follows the following procedure, illustrated by the flowchart in Figure 2:

- The supplicant initiaties an EAP-TLS authentication handshake.

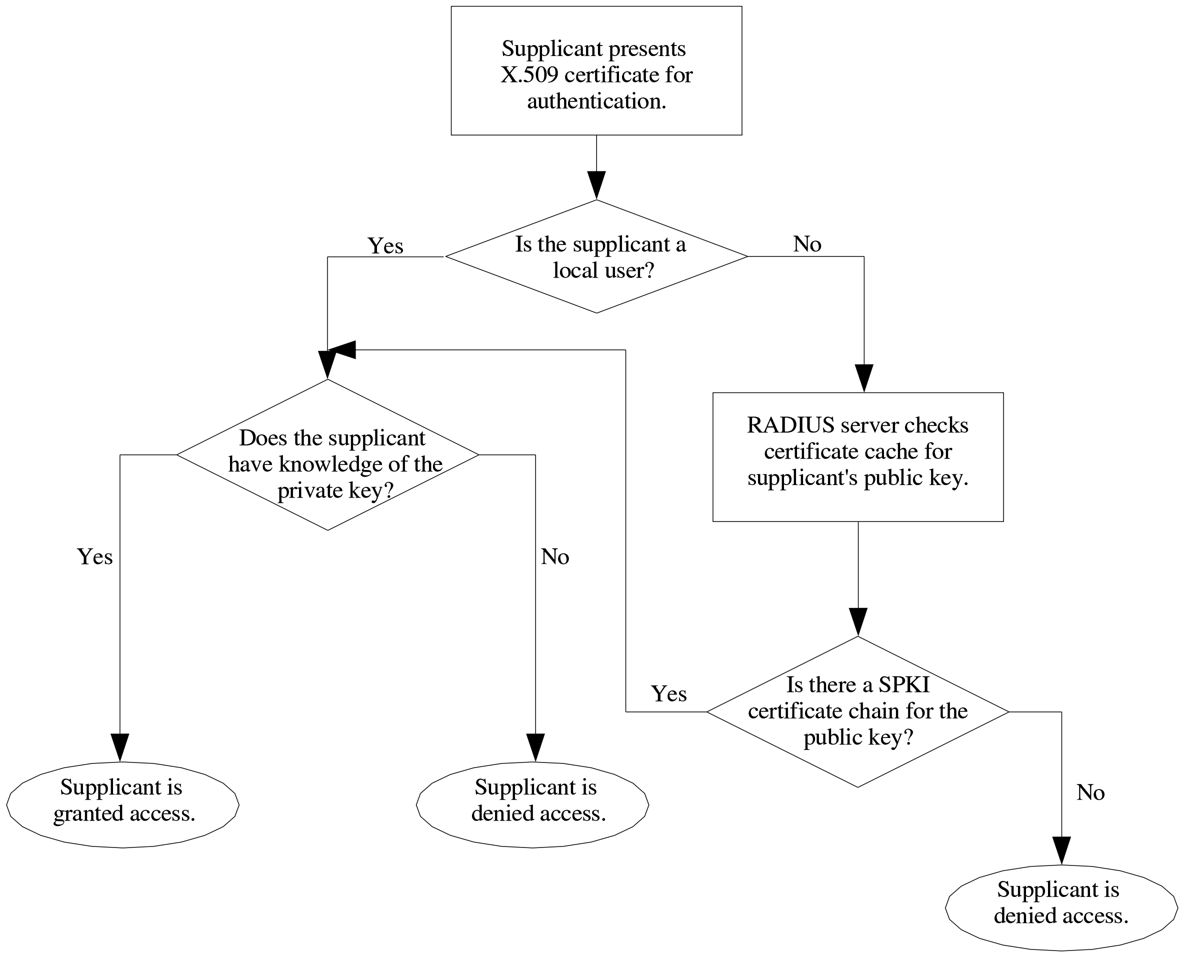

Supplicant presents X.509 certificate for authentication.
Is the supplicant a local user?
Yes
No
Does the supplicant have knowledge of the private key?
RADIUS server checks certificate cache for supplicant's public key.
Yes
No
Yes
Is there a SPKI certificate chain for the public key?
No
Supplicant is granted access.
Supplicant is denied access.
Supplicant is denied access.

Figure 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, the RADIUS server checks for a SPKI certificate chain vouching for the supplicant's public key.

- If the supplicant cannot present an identity certificate, we shunt them to a special VLAN on which the supplicant can only connect to our delegation tool's "welcome" page.
- If the supplicant *can* present an identity certificate, we then evaluate it as follows:
  - If the certificate is valid and issued by the local CA, then we accept it.
  - Otherwise, if we can obtain and verify a valid SPKI/SDSI chain supporting it, we accept it.
  - Otherwise, we reject the certificate and shunt the supplicant to our "welcome" page.
- If we accept the certificate, and the supplicant proceeds to prove knowledge of the private key, then we let him in.
- Otherwise, we shunt the supplicant to our "welcome" page.

This procedures modifies standard EAP-TLS implementations only by changing how the server decides to accept a given supplicant certificate.

Getting the certificates ("pull" approach). To carry out the guest user case, the RADIUS server needs to know the X.509 identity certificate, the public key of whatever source-of-authority SPKI/SDSI chains will originate from, and the SPKI/SDSI certificate chain itself. EAP-TLS gives us the first, and we can build in the second. But how do we find the relevant SPKI/SDSI certificates?

One solution would be to have the delegation process leave the authorization certificates in a reliable, available directory where servers can access them; since the data is self-validating, maintenance of this directory should be automatic. When the RADIUS server needs to verify a guest's SPKI/SDSI credentials, it can "pull" up the credentials it requires from the directory. We can organize these certificates as a forest: guest authorization certificates are children of the delegation certificates that signed them.

- The source-of-authority tool needs to write new delegator certificates to this directory.
- The delegator tool needs to read delegator certificates from this directory, and write new guest authorization certificates back.
- The RADIUS server needs to be able to ask for delegator-authorization chains whose leaves speak about a given public key.

The directory itself can perform time-driven checks for expiration.

Our initial implementation used the "pull" approach just described: SPKI/SDSI certificates were maintained in a cache that the RADIUS server can query via XML-RPC. The RADIUS server queries the cache about a particular public key; the cache itself finds a chain, if it exists, verifies it, and returns it. (To make our prototype more secure, we need to use authenticated XML-RPC messages or move the decision procedure onto the same machine as the RADIUS server.)

Getting the certificates ("push" approach). The centralized solution above is somewhat unsatisfying, because it introduces a centralized component (even if this component does not have significant security requirements). It would be slicker to find a way for the delegator and guest themselves to carry around the necessary certificates, since the necessary information paths will exist. When necessary, the guest can "push" the necessary credentials to the RADIUS server for validation.

We note that HTTP cookies provide most of the functionality we need. (We will add a message to the guest welcome page notifying users of what browser features will need to be enabled, including cookies and Java, in order to use our services.)

- The delegator will be interacting with the source-of-authority signing tool when their delegation certificate is created; the delegation certificate could be saved at the delegator machine as a cookie.
- At delegation time, both the delegator and the guest will be interacting with the delegation tool. The tool can read the delegator's certificate chain as a cookie, concatenate it with the new authorizatiion certificate, and then store the resulting chain as a cookie in the guest's Web browser.

The only remaining question would be how to get this cookie from the guest machine to the RADIUS server, when an authorized guest connects. One approach would be to add a short-term SPKI/SDSI store to the RADIUS server. When deciding whether to accept an X.509 certificate not issued by the Dartmouth CA, the server looks in this store for a SPKI/SDSI certificate chain for the public key in this X.509 cert. If none can be found, the supplicant is routed to a special Web page, that will pick up the guest's certificate chain from an HTTP cookie (this step requires that the guest have a browser running) and save it in the store.[4]

In this decentralized approach, it also might make sense to have the delegation tool save newly created SPKI/SDSI chains in the short-term store at the RADIUS server, since the guest will likely want to use the network immediately after being delegated to.

Changing VLANs. We now have two scenarios—when first receiving delegation, and in the above decentralized store approach—where a supplicant will be connected through the access point to the special VLAN, but will want to then get re-connected to the standard network. In both scenarios, the guest will be interacting with the Web server we have set up on the special VLAN.

One way to handle this would be for our server to display a page telling the guest how to cause their machine to drop the network and re-associate. However, this is not satisfying, from a usability perspective.

Instead, it would be nice to have our server (and backend system) cause this action automatically. One approach would be to use the administrative interface provided by the access point. For example, the Cisco 350 access point (that we're experimenting with) permits an administrator, by a password-authenticated Web connection, to dis-associate a specific supplicant (after which the supplicant re-initializes the network connection, and tries EAP-TLS again). We could write a daemon to perform this task, when it receives an authenticated request from our backend server. The server needs to know *which* access point the supplicant is associated with; however, in both scenarios, the RADIUS server has recently seen the supplicant MAC and access point IP address, since it told the access point to route this supplicant down the special VLAN. If nothing else, we can cache this information in a short-term store that the daemon can query.

We plan to explore other approaches here as well.

[4]As this paper goes to press, we have successfully implemented the cookie-based approach suggested here.

7.2 Executing the decision

On the server side, we are currently using FreeRadius version 0.9.2, running on a Dell P4 workstation running Red Hat 9, and an Apache Web server running on another Dell P4 workstation running Red Hat 9. We're testing with a Cisco 350 access point, with a Cisco Catalyst 2900 series XL switch and a hub to connect the two machines running the RADIUS server and Web server.

Setup. In our prototype, we have the access point configured to provide two different SSIDs. The broadcast SSID is called "Guest user" and authentication is not needed. It associates all users onto VLAN 2, the guest VLAN. The SSID "Dartmouth user" is not broadcast, and requires EAP authentication. Supplicants who pass EAP authentication are associated to this SSID on VLAN 1, the native VLAN that has access to the whole network. (We will abbreviate these designations as V_1 and V_2 in the following discussion.)

Our VLAN configuration is illustrated in Figure 3. The RADIUS server is connected to V_1 on the switch and the Web server is connected to V_2. The access point, connected to V_1, is configured to query the RADIUS server for user authentication. The hub connects the two machines and allows them to communicate to one another through the resulting private connection. In the future, we will obtain a router capable of VLAN trunking, which will allow the Web server to exist on both VLANs; this will eliminate the need for the private connection through a hub.[5]

Configuration. The EAP-TLS module of FreeRADIUS uses OpenSSL to execute the SSL/TLS handshake between the supplicant and the RADIUS server. After we changed the appropriate FreeRADIUS configuration files to enable EAP-TLS authentication and linked it with 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 (called a *network authentication server*, *NAS*, in the RADIUS protocol) with a Dartmouth IP address and the user file was set to only allow EAP (in our case EAP-TLS) authentication for all users, placing the user on V_1 if successfully authenticated. A shared secret between the RADIUS server and the NAS secures communication between these two components.

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. 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]. Once the RADIUS server has a trusted root CA to refer to, it can handle authentication requests from the access point.

We modified the RADIUS server code to link with XML-RPC libraries we installed on the same machine. These libraries allow the RADIUS server to commmunicate with the cache, mentioned above, that stores SPKI/SDSI certificates and searches for chains authorizing a given principal to connect.

The decision process. The RADIUS server idles and waits for packets. When it receives an EAP Access-Request packet, it checks to see if the NAS that sent the packet is recognized and the shared secret is correct. If so, then it looks at the packet and sees what type of authentication is used. Since the SSID is configured to require EAP authentication, the RADIUS server should only receive EAP authentication requests from the NAS.

Once the EAP-TLS module is done executing, the decision to accept or reject the supplicant has already been made and is packed into the response packet. Thus it is necessary to intercept the EAP-TLS module before a reject decision is made to accomodate any modifications to the decision process.

Our modification determines if there is an error code returned by reading the supplicant's certificate. For example, the most common case would be the certificate is issued by an unrecognized CA. Once the validity checks are finished, we read the resulting error code to see if the validation passed or failed. If it passed, then the certificate presumably was issued by the known CA and the supplicant has provided knowledge of the corresponding private key. If the handshake failed due to an unrecognized CA, however, we use XML-RPC to query the Java SDSI library code about the public key of the X.509 certificate provided. The library uses the SPKI/SDSI certificate chain discovery algorithm proposed by Clark et al. [CEE+01]. If the Java code finds a valid SPKI certificate chain vouching for the supplicant, then we accept the supplicant and the EAP-TLS module returns an accept code. If such a SPKI/SDSI certificate chain cannot be found, then the user is rejected. Once graceful VLAN switching is implemented, the unauthorized guest will be placed on V_2 and see a Web browser window with instructions for obtaining guest access.

[5]We recently revised our setup to include a Cisco 2600 series router to handle VLAN trunking. The revised setup also uses newer models of the switch (Cisco 3550 series) and access point (Cisco 1100 series).

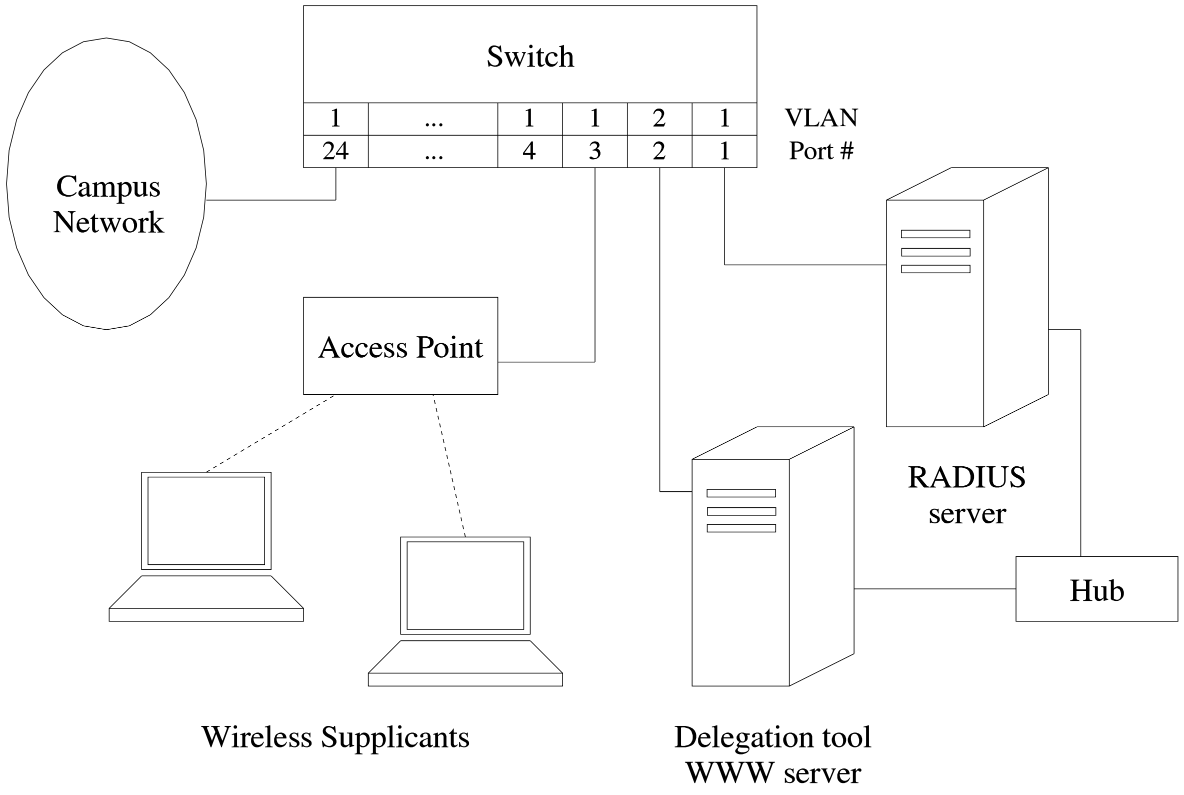


Figure 3: The setup of the Greenpass prototype. The switch is configured to associate VLANs with physical port numbers. The Web server is the only element currently housed on VLAN 2. Eventually, VLAN trunking will be used to communicate between the RADIUS server and the web server, eliminating the need for the private connection that exists between the two.

8 Related Work

Balfanz et al. [BDS$^+$03] propose using secret keys to let wireless parties authenticate. We've already noted related work [DE02] in the "introduction" problem between two devices.

In the SPKI/SDSI space, the Geronimo project at MIT [Cla01, May00] uses SPKI/SDSI to control objects on an Apache Web server. The project uses a custom authorization protocol, with an Apache module handling the server side of the protocol and a Netscape plug-in handling the client side. The protocol can be tunneled inside an SSL channel for further protection; the authors also considered replacing X.509 with SPKI/SDSI within SSL. Howell and Kotz [HK00] describe a similar SPKI/SDSI-based access-control system for Web content as part of their *Snowflake* authorization architecture. Koponen et al. [KNRP00] propose having an Internet cafe operator interact with a customer via infrared, and then having that customer authenticate to the local access point via a SPKI/SDSI certificate; however, this work does not use standard tools and institution-scale authentication servers. Eronen and Nikander [EN01] describe several SPKI/SDSI-based enhancements to both authorization and authentication in Jini, a Java-based distributed computing environment.

Canovas and Gomez [CG02] describe a distributed management system for SPKI/SDSI name certificates and authorization certificates. The system contains name authorities (NAs) and authorization authorities (AAs) from which entities can request name and authorization certificates, including certificates which permit the entity to make requests of further NAs and RAs. The system takes advantage of both name certificates that define groups (i.e., roles) and authorization certificates that grant permissions to either groups or individual entities.

9 Future Directions

Initially, we plan to "take the duct tape" off of our current prototype, and try it in a more extensive pilot. Beyond this initial work, we also hope to expand in several directions.

No PKI. We note that our approach could also accommodate the scenario where *all* users are "guests" with no keypairs—in theory, obviating the need for an X.509 identity PKI for the local population. For example, if an institution already has a way of authenticating users, then they could use a modified delegator tool that:

- authenticates the delegator (via the legacy method)
- sees that the delegator has a self-signed certificate (like our guest tool does)
- then signs a SPKI/SDSI delegator certificate for this public key (like our delegator tool does).

In some sense, the division between the X.509 PKI and the delegated users is arbitrary. It would be interesting to explore the implications of dividing the population in other ways than users versus guests (perhaps "permanent Dartmouth staff" versus "students likely to lose their expensive smart-card dongles while skiing").

Not just the network. Many types of digital services use X.509 identity certificates as the basis for authentication and authorization. For example, at Dartmouth, we're migrating many current Web-based information services to use X.509 and client-side SSL/TLS. In the Greenpass pilot, we're adding flexibility to wireless access by extending X.509/TLS with SPKI/SDSI. This same PKI approach can work for networked applications that expect X.509, such as our Web-based services.

In the second phase, we will extend the Greenpass infrastructure to construct a single tool that allows delegation of authorization to networked applications as well as to the network itself.

Not just EAP-TLS. Some colleagues insist that *virtual private networks* (*VPNs*) with client-side keypairs are the proper way to secure wireless networks. In theory, our scheme should work just as well there. In the second phase, we plan to try this.[6]

Alternative approaches to hash verification. An attacker could potentially abuse our delegator applet if the delegator chooses to skip the fingerprint-verification step. Visual fingerprints are designed to discourage users from skipping crucial verification steps: it is faster and less painful to compare two visual fingerprints than to compare hashes represented as hexadecimal strings. We must devise either a method that ensures the delegator cannot skip this step,[7] or a method that takes humans out of the loop entirely. Balfanz et al. [BSSW02] suggest an introduction phase based on a *location-limited channel*; this approach might allow us to eliminate human interaction from the introduction phase in the future. We are also considering alternative models of fingerprint verification: for example, using PGPfone's [PGP] mapping of hash values to word lists would allow introduction to take place over the phone as well as in person.

Other threats. We designed our prototype around the concept that a user's public key is his or her online identity; as a result, delegators authenticate to Web servers and the RADIUS server using the same keypair (identity) as they use to sign SPKI/SDSDI certificates for guests. A weakly designed authentication protocol—one that requires a user to sign an unstructured random challenge using his private key—could be exploited by a malicious server. Specifically, the server might present a "random" challenge that actually contains a SPKI/SDSI certificate or its hash: a delegator could then be tricked into signing that certificate by authenticating to the malicious server, whose owner might use the resulting signature to obtain unauthorized access to the wireless LAN or some other resource. We therefore need to consider whether the TLS and SSL client authentication handshakes are vulnerable to such an attack.

The TLS 1.0 [AD99] and SSL version 3.0 [FKK96][8] handshakes appear to be immune to this attack due to the format of the MAC that the client signs in order to authenticate (at least when using an RSA keypair: see below). In both protocols, the MAC that is signed is a *concatenation* of both the MD5 and SHA-1 hashes of the values in question. The SPKI/SDSI certificate format [EFL+99a] defines signatures using *either* an MD5 or a SHA-1 hash.

When authentication using a DSA keypair or using SSL version 2.0 [Hic95], on the other hand, the client signs *only* a SHA-1 or an MD5 hash, respectively. In both these latter cases, however, the signed MAC is function of previous handshake values, including random values generated by the client and structured values such as complete handshake messages (in TLS or SSLv3) or the server's certificate (in SSLv2; note that the client will already have verified this certificate before sending its own authentication materials). As a result, it is not possible for the server to get the client to sign a value that is a valid SPKI/SDSI certificate structure. A malicious server would need to engineer its own handshake values in such a way that the entire sequence signed by the client has the same MAC value as the server's desired

[6]As this paper goes to press, we have successfully completed an initial test of VPN guest access using our existing client tools and RADIUS server; see Goffee [Gof04] for further details.

[7]An in-progress revision of our delegator tool requires the user to select the correct visual hash from among several choices.

[8]Also see Rescorla [Res01] for further discussion of both these protocols.

SPKI/SDSI certificate. This would require finding a collision in the hash function used; if a particular hash function is proven to be vulnerable to such attacks, we could begin accepting only those authorization certificates signed using stronger hash function.

Location-aware authorization and services. By definition, the RADIUS server making the access-control decision knows the supplicant's current access point. In some scenarios, we may want users to access the network only from certain access points; in some scenarios, users should be able to access some applications only from certain access points; potentially, the nature of the application content itself may change depending on access location.

In the second phase, we plan to extend the Greenpass infrastructure to enable authorization certs to specify the set of allowable access points. We will also enable the RADIUS back-end to sign short-term certificates testifying to the location of the supplicant (which requires an authorization cert for the server public key), and to enable applications to use these certificates for their own location-aware behavior. For example, we might put different classes of users (professors, students, guests, etc.) on different VLANs according to the resources we would like them to access. It might also be interesting to allow certain users to access the WLAN only from certain locations—e.g., conference rooms and lecture halls.

Who is being authorized? Campus environments are not monolithic. At Dartmouth, we already have multiple schools, departments, and categories of users within departments. Managing authorization of such internal users is a vexing problem. Centralized approaches are awkward and inflexible: a colleague at one university ended up developing over 100 different user profiles; a colleague at another noted she has to share her password to team-teach a security course, because the IT department has no other way to let her share access to the course materials.

In the second phase, we plan to extend the Greenpass infrastructure to support authorization delegation for "local users" as well as guests, and to permit local users to easily manage authorization for information resources they own or create.

Devices. Currently, laptops are probably the most common platform for access to wireless networks. Other platforms are emerging, however. At Dartmouth, students and staff already carry around an RFID tag embedded in their ID cards, a research team is developing experimental wireless PDAs for student use, and we are beta-testing Cisco's new VoIP handset device; we're also testing Vocera's device for Wi-Fi voice communication.

In the second phase, we plan to explore using these alternate devices in conjunction with Greenpass. For example, a department's administrative assistant might be able to create a SPKI/SDSI cert and enter it in a directory simply by pointing a "delegation stick" (RFID tag reader) at the student (detecting the student's ID card). In another example, when a physician at the Dartmouth-Hitchcock Medical Center collars a passing colleague for advice on a difficult case, he might be able to delegate permission to read that file simply by pointing his PDA at the colleague's PDA.

Distributed authorization. The PKI community has long debated the reason for PKI's failure to achieve its full potential. The technology exists and has clear benefits; adoption and deployment has been a challenge.

One compelling hypothesis is that the centralized, organizational-specific hierarchy inherent in traditional approaches to PKI, compounded by a dependence on usable, globally unique names and awkward certificate structure, does not match the way that authorization really flows in human activities. By permitting authorization to start at the end-users (rather than requiring that it start at centralized places), and by using a system (SPKI/SDSI) designed to address the namespace and structure issues, Greenpass may overcome these obstacles.

In the second phase, we plan to extend Greenpass to reproduce real-world policies more complex than just "Prof. Kotz said it was OK," to examine (with our colleagues in the Dept of Sociology) how readily this authorization system matches the norms of human activity, and to examine whether humans are able to manage the user interfaces our Greenpass tools provide.

We also plan to take a closer look at how other authorization schemes might fit in this setting, in comparison to SPKI/SDSI. Some candidates that might work in this setting include the X.509-based PERMIS attribute certificate system [COB03, PER] and X.509 proxy certificates [TWE$^+$03, WFK$^+$04], as well as KeyNote [BFIK99, Key]. Theses by Nazareth [Naz03] and Goffee [Gof04] give overviews of many such systems.

10 Conclusion

In this paper we described a method of securing a wireless network while providing meaningful guest access. We added a step to EAP-TLS authentication that performs an additional authorization check based on SPKI/SDSI certificates. By using SPKI/SDSI, we eliminate the need for a cumbersome central authority; by grafting it on top of the existing X.509-based PKI, we do not require our users to install any additional client software.

The two major components of the Greenpass project are the delegation tools and the modified RADIUS server. The delegation tools automate the process of creating temporary SPKI/SDSI certificates for a guest, allowing an authorized (but not necessarily computer-savvy) delegator to grant an invited guest permission to use the network. The modified RADIUS server takes into account that guests will want to access the network and checks for guest credentials before making a decision to accept or reject a supplicant's request for network access.

The goal of our project is to create a solution that implements delegation in a way that reflects real-world authorization flow that does not rely too heavily on a centralized authority; SPKI/SDSI allows us to accomplish this goal. Our future work will allow us to investigate how our solution fits with other existing ideas, hopefully resulting in a solution that is secure, completely decentralized, and capable of adapting to new technology and delegation policies.

Acknowledgments

We gratefully acknowledge Kwang-Hyun Baek, Bob Brentrup, Bill Cote, Peter Glenshaw, Dave Kotz, Brad Noblet, and our colleagues at Cisco for the advice, and Eileen Ye for her initial SPKI/SDSI explorations. We plan to make code for this project available under an open-source license.

References

[AD99] Christopher Allen and Tim Dierks. The TLS Protocol Version 1.0. IETF RFC 2246, January 1999.

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

[BDS+03] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana Smetters, Jessica Staddon, and Hao-Chi Wong. Secret Handshakes from Pairing-Based Key Agreements. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 180–196, May 2003.

[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.

[BGW01] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In *Proceedings of the International Conference on Mobile Computing and Networking (MOBICOM)*, 2001.

[BSSW02] Dirk Balfanz, D. K. Smetters, Paul Stewart, and H. Chi Wong. Talking to Strangers: Authentication in Ad-Hoc Wireless Networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, February 2002.

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

[CEE+01] D. Clark, J. Elien, C. Ellison, M. Fredette, A. Morcos, and R. 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.

[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. *IEEE Internet Computing*, March-April 2003.

[CS01] IEEE Computer Society. IEEE Standard for Local and metropolitan area networks: Port-Based Network access control. IEEE Standard 802.1X-2001, October 2001.

[CWHWW03] Nancy Cam-Winget, Russ Housley, David Wagner, and Jesse Walker. Security Flaws in 802.11 Data Link Protocols. *Communications of the ACM*, 46(5):35–39, May 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$^+$99a] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. Simple public key certificate. IETF Internet-Draft, `http://theworld.com/~cme/examples.txt`, 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.

[EN01] Pasi Eronen and Pekka Nikander. Decentralized Jini Security. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, pages 161–172, February 2001.

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

[FKK96] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL Protocol Version 3.0. IETF Internet-Draft, `http://wp.netscape.com/eng/ssl3/draft302.txt`, November 1996.

[Gof04] Nicholas C. Goffee. Greenpass Client Tools for Delegated Authorization in Wireless Networks. Master's thesis, Dartmouth College, June 2004. Technical Report TR2004-509.

[Gol] Ian Goldberg. Visual Fingerprints (Visprint software homepage). `http://www.cs.berkeley.edu/~iang/visprint.html`.

[Hic95] Kipp E. B. Hickman. SSL 2.0 Protocol Specification. Netscape draft specification, `http://wp.netscape.com/eng/security/SSL_2.html`, February 1995.

[HK00] Jon Howell and David Kotz. End-to-end authorization. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI)*, pages 151–164, 2000.

[Joh] David Johnston. Visprint, the Visual Fingerprint Generator. `http://www.pinkandaint.com/oldhome/comp/visprint/`.

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

[Kim04] Sung Hoon Kim. Greenpass RADIUS Tools for Delegated Authorization in Wireless Networks. Master's thesis, Dartmouth College, June 2004. Technical Report TR2004-510.

[KNRP00] Juha Koponen, Pekka Nikander, Juhana Rasanen, and Juha Paajarvi. Internet Access through WLAN with XML-encoded SPKI Certificates. In *Proceedings of NORDSEC 2000*, 2000.

[May00] Andrew J. Maywah. An Iplementation of a Secure Web Client Using SPKI/SDSI Certificates. Master's thesis, Massachusetts Institute of Technology, May 2000.

[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, Department of Computer Science, Dartmouth College, May 2003. `http://www.cs.dartmouth.edu/~pkilab/theses/sidharth.pdf`.

[PER] PERMIS home page. `http://www.permis.org/`.

[PGP] PGPfone: Pretty Good Privacy Phone Owner's Manual. `http://web.mit.edu/network/pgpfone/manual/`.

[Res01] Eric Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, 2001.

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

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

[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.

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

[TER] Trans-European Research and Education Networking Association (TERENA). `http://www.terena.nl/tech/task-forces/tf-mobility/`.

[TWE+03] Steven Tuecke, Von Welch, Doug Engert, Laura Pearlman, and Mary Thompson. Internet X.509 Public Key Infrastructure Proxy Certificate Profile. IETF Internet-Draft, `http://www.ietf.org/internet-drafts/draft-ietf-pkix-proxy-10.txt`, December 2003.

[WFK+04] Von Welch, Ian Foster, Carl Kesselman, Olle Mulmo, Laura Pearlman, Steven Tuecke, Jarek Gawor, Sam Meder, and Frank Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. In *3rd Annual PKI Research and Development Workshop*, 2004.