Dartmouth College

Dartmouth Digital Commons

Computer Science Technical Reports

Computer Science

1-7-2004

Greenpass: Flexible and Scalable Authorization for Wireless Networks

Sean Smith
Dartmouth College

Nicholas C. Goffee
Dartmouth College

Sung Hoon Kim
Dartmouth College

Punch Taylor
Dartmouth College

Meiyuan Zhao
Dartmouth College

See next page for additional authors

Dartmouth Digital Commons Citation

Smith, Sean; Goffee, Nicholas C.; Kim, Sung Hoon; Taylor, Punch; Zhao, Meiyuan; and Marchesini, John, "Greenpass: Flexible and Scalable Authorization for Wireless Networks" (2004). Computer Science Technical Report TR2004-484. https://digitalcommons.dartmouth.edu/cs_tr/227

Authors
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

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

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

Version of January 7, 2004

Abstract

Wireless networks break the implicit assumptions that supported authorization in wired networks (that is: if one could connect, then one must be authorized). However, ensuring that only authorized users can access a campus-wide wireless network creates many challenges: we must permit authorized guests to access the same network resources that internal users do; we must accommodate the de-centralized way that authority flows in real universities; we also must work within standards, and accommodate the laptops and systems that users already have, without requiring additional software or plug-ins.

This paper describes our ongoing project to address this problem, using SPKI/SDSI delegation on top of X.509 keypair within EAP-TLS. Within the "living laboratory" of Dartmouth's wireless network, this project lets us solve 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 secure wireless networks in large institutions, while also permitting authorized guest access to the institution's network and selected internal resources (as well as to the guest's home systems).

This project—based on SPKI/SDSI and EAP-TLS—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 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`

This Paper. This paper is intended as a snapshot of the current state of our project. Section 2 reviews the problem, and Section 3 reviews the wireless background. Section 4 presents weaknesses in some current attempts to secure a wireless network. Section 5 presents our approach: Section 6 discusses the delegation piece; Section 7 discusses the access decision piece. Section 8 discusses the current status of our prototype, and Section 10 discusses future directions.

More lengthy discussions (e.g., [Gof04, Kim04]) of this work will be appearing this spring.

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 is as unthinkable as a lack of electricity. To achieve its potential in these environments, however, wireless needs to be secure against unauthorized access via moderately sophisticated adversaries, who may use drive-by networking, sniffing, black hat software, and other tools. (At a minimum, we should expect adversaries to do what we can do with only moderate effort—see Section 4.)

This future raises some challenges:

- We need to permit authorized users to access the network.
- We also need to permit "guests" to access the network.
- We must minimize the hassle needed to delegate authorization to guests, and we must accommodate the decentralized ways that authority really flows in large universities.
- 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 within each university.
- 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 keep encountering discussions of "guest access" that define the term substantially different from how we define it. We are not letting just anyone onto the net (one definition we have seen); we are also not restricting guests to special access points, such as in a conference room, that connect to outside the firewall; we are also not forcing all guest access to go outside the firewall.

We want guests to access the inside; that's the whole point.

We also want to permit authorization to flow the way it flows in the real world; we don't want central authority (or a central box and rights system purchased from a single vendor).

3 Background

Access Control. Institutions often wish to restrict who can use their networks, for several reasons. A basic one is economic. Other reasons include 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 lock out 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.

The Wired World. In the typical *tethered* or *wired* network, a machine is physically connected to an institution's physical network. Access control often implicitly depends on the physical limits of this arrangement. If a machine is connected to a subnet whose only jacks are inside building X, then that machine must be in building X; if the jacks are inside locked offices, then the machine must be in the putative control of a keyholder; if the IT policy encourages that (in this building) only the central sysadmins can have root privileges, then this machine is probably controlled by the sysadmins.

Here at Dartmouth, one example of this implicit assumption is that the Computer Science Department's NFS installation used to assume that any machine with a Sudikoff Laboratory IP address lived by the Computer Science configuration—so any such machine had full access to the departmental filesystem, because the machine itself would enforce the proper access control. Other examples abound—for example, the JSTOR online journal archive assumes that if a request comes from a machine with an IP address of a subscribing institution, then the requester is an authorized user from that institution.

Wireless. Wireless changes all of that. In wireless networking, a machine "connects" to the network via radio transmission. On a basic physical level, the only requirement is that the two stations be in radio range of each other—which is much looser than the previous constraint of a machine needing to be in the same room as the network jack.

Wireless networking comes in two basic flavors. In the *ad hoc* approach, the wireless stations talk to each other; in the *infrastructure* approach (more common in higher education institutions), wireless stations connect to special tethered stations, that in turn connect to a wired network. In this scenario, the wireless stations are called *supplicants* and the tethered bridge stations are called *access points.* Housely and Arbaugh [HA03] give a good, concise introduction to some of this terminology.

Navigating wireless also requires navigating through an alphabet soup of standards and drafts. *802.11* is the IEEE set of standards describing wireless. These standards have different letters after their names. Some of these refer to particular ways of stuffing the bits through at high speeds (e.g., "802.11a," "802.11b," "802.11g"). *802.11i* refers to security; it includes *802.1x*, a more general networking security standard. *WiFi* is as a specific term for a certification process established by the *WiFi Alliance*, a vendor consortium; it is often used less precisely for for 802.11 networks in general.

On a technical level, wireless raises several security questions. How do we protect the session between the supplicant and the access point? How does the access point ensure that the supplicant is authorized for network access? How does the supplicant determine that it's talking to the intended access point? To protect the session, the field initially offered *wired equivalent privacy (WEP)*, a fairly complex cryptographic protocol that had the drawbacks of being flawed, and being seldom used. (Cam-Winget et al [CWHWW03] give a good review of the flaws.) To authorize supplicants, standard approaches include allowing only those supplicants that know the correct *SSID* for this access point, or perhaps only allowing supplicants whose *MAC* addresses appear on an access control list.

To strengthen the situation, the WiFi alliance recently changed their certification rules to require *WiFi protected access (WPA)*. WPA provides a stronger encryption scheme, and requires support for a richer set of authentication techniques (by including a draft of 802.11i, and hence 802.1x). Edney and Arbaugh's recent book [EA04] provides a thorough overview of this space.

EAP-TLS. Authenticating a large user space suggests the use of *public-key cryptography*, since that can avoid the security problems of shared secrets and the scalability problems of ACLs. One public-key authentication techniques permitted by WPA is *EAP-TLS* [AS99, BV98]. TLS (*transaction layer security*) is the standardized version of *SSL (Secure Sockets Layer)*, the primary means for authentication and session security in the Web.

In the Web setting, the client and server want to protect their session and possibly authenticate each other. Although SSL/TLS permits the two parties to use Diffie-Hellman to establish shared secrets to cryptographically protect the session, the more common approach is for the server to present a public key certificate and prove knowledge of the corresponding private key. A growing number of institutions (including Dartmouth) also exploit the ability of SSL/TLS to also require the client to present a certificate and prove knowledge of its private key to the server; the server can use this certificate to decide whether to grant access, and what Web content to offer. This initial handshake also permits the client and server to negotiate what cryptographic suite they wish to use to protect the rest of the session, and to establish shared secrets (via the public keys) to use for this session cryptography.

The EAP-TLS variant 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 an *authentication server*; typically, this is a *RADIUS* server. [Ros03, Sul02].

The term RADIUS stands for *Remote Authentication Dial In User Service (RADIUS)* [Rig00a, Rig00b, Rig00c], and is often used both for the specific protocol to talk to an authentication server, as well as for the server itself (e.g., “a RADIUS server”). In a network, the RADIUS server would be responsible for authenticating users and returning the necessary configuration information to the Network Access Server (such as an access point) to provide service to the user . Access points use the *Extensible Authentication Protocol (EAP)* to shuttle more advanced authentication handshakes between the supplicant and the RADIUS server.

In EAP-TLS, the access point and authentication server initially act as a unit, and (like TLS) establish a session key with the supplicant (for WPA). When the RADIUS server acts as the authentication server, the access point acts just as the middleman in the process. The RADIUS server and the access point have a shared secret to provide secure communication. Once the RADIUS server authenticates the supplicant and establishes a session key to be used, it is done with the exchange.

Similar to browsers, supplicant software for EAP-TLS needs a way to “trust” the certificate that the authentication server presents. In our initial experiments on Windows, standard supplicant tools borrow the certificate store and trust process of IE.

Our Approach. WPA with EAP-TLS permits us to work within the existing WiFi standards, but enable the supplicant and access point to 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” SSID for local networks.
- We can read the MAC addresses of supplicants permitted to access the net.
- 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:

- We can easily demonstrate that security solutions which depend on secret SSIDs or authenticated MACs do not work.
- The current Dartmouth wireless net 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 Process

We want to use the decentralized, scalable, and flexible nature of PKI to follow the decentralized way that authorization flows in institutions, but glue this into EAP-TLS.

In a wireless network in a large institution, the decision to grant authorization won't 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. Consider the flow of authorization for access in the non-digital world. In a simple scenario, Alice is allowed to go in the 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.

Certificate Building Blocks. In the real world, standards and tools for PKI—including SSL/TLS—generally focus on *X.509 identity certificates* and (for end users) browser/OS keystores. Typical enterprise PKIs start with an enterprise-specific CA issuing X.509 identity certificates for its users. If one wants to deploy a PKI without reinventing the wheel, this is where one starts.

However, X.509 has some substantial disadvantages for this application, where we need to delegate to keyholders from another enterprise.

- In theory, numerous schemes permit easy use of one enterprise's certificates within an another (e.g., so the Dartmouth system can automatically obtain, validate, and understand the credential chain presented by a visitor from Princeton); in practice, painless interoperability between arbitrary deployed enterprise PKIs will not happen soon (although current efforts to build the *higher-ed bridge CA* may change that).
- More significantly, X.509 does not easily extend to express assertions such as "Alice said it was OK." (The closest match might be X.509 proxy certificates, but these are limited and inflexible.)

However, alternative standards and tools exist that can fill this gap Of particular interest here is *SPKI/SDSI* [EFL$^+$98, EFL$^+$99a, EFL$^+$99b]: a format and semantics for certificates that is intentionally streamlined, and focused on exactly the issue at hand: authorization. An additional advantage of SPKI/SDSI is that (unlike X.509) coding for it is fairly lightweight.

Ordinary Users. As noted, we want to build on EAP-TLS. As part of our campus PKI, we are already setting up each Dartmouth user with a keypair and X.509 identity certificate. We'll also need to train them on how to arm their supplicant software with these items. If an ordinary user wants to access the network, the access point/authentication server requires that they use EAP-TLS, and lets them in if they prove knowledge of a private key matching a Dartmouth-issued X.509 identity certificate.

We'll also need to make a policy decision on how many RADIUS servers should be involved. If more than one, how many server keypairs should be involved? If only one, doesn't that create a scalability problem? We also need to examine supplicant tools to determine how to set them up to recognize a trust root, and then configure them to be happy with our RADIUS certificates (perhaps by setting these up as trust roots; perhaps by issuing them from our Dartmouth CA, and setting that up as a trust root).

Guests. In the physical world, a guest gets access to a physical campus resource because, according to the local policy governing that resource, someone who has the power to do so said it was OK. To reproduce this in our wireless setting, we'll start by assuming that guests have keypairs and X.509 identity certificates—although we have no idea who these are issued by, and thus our access points and authentication servers can make no reasonable trust conclusions about them (without further steps). If a guest doesn't have a keypair, we'll have to see that he or she gets one—self-signed will do. To handle the authorization, we need:

1. a way to indicate what local people have the power to "say it's OK";
2. a way for them to say that about guests;
3. a way for the access point and authentication servers to figure that out, within the confines of EAP-TLS;
4. and a way for us to code this up and make it real, without too much work.

As noted above, our initial solution here is to use SPKI/SDSI certificates: a lightweight approach (point 4) for the access point's source of authority to sign a certificate for the local people authorized to delegate (point 1), who in turn can sign authorization certificates for guests (point 2). All we need to do is modify the way that the RADIUS server evaluates guest certificates to accept such a SPKI/SDSI chain (point 3).

The Rest of the Paper. In what follows, Section 6 discusses our proposed delegation process, and Section 7 discusses our proposed changes to how the RADIUS server decides to accept a supplicant certificate.

Section 8 then discusses what happened when we actually tried all of this. Section 10 discusses further extensions.

6 Delegation

When trying to design the delegation process, we run into some design questions:

- How do we express the fact that the delegator is allowed to delegate?
- How does the delegator learn what the guest's public key is?
- What do we about guests who don't have a keypair?

The Ability to Delegate. To express the that the delegator can delegate, we initially thought we would add a flag as an extension in the delegator's X.509 identity certificate. However, this approach would impact the process of issuing the identity certificates and potentially impact their use in other applications. This would also require changing the identity certificate each time the delegator's status changes, and require that validation of guest access require detailed X.509 parsing.

Instead, we realized that a much simpler approach would be (as noted above) to move the source of authority for delegation from the identity CA to a separate entity—and then have this entity express that ability as a SPKI/SDSI certificate about the delegator's public key. This approach avoids the above problem, and gives us much additional flexibility.

Learning the Guest's Public Key. The next challenge is how the delegator learns what the guest's public key is. This challenge has (at least) two parts.

First is the obvious: the bits need to get from the guest's machine to the entity that composes the SPKI/SDSI certificate. (This entity might be the delegator's machine or a backend server.) This task requires an information path from the guest's machine to the composing entity; we considered many possible approaches, depending on this path. Can we assume that the guest gives the delegator a fbppy or memory stick? (Not reasonable.) Can the guest provide their certificate via a Web action before they come to Dartmouth? (Doesn't cover enough scenarios.) Can the guest try to authenticate to the access point/authentication server and fail—but our modified RADIUS server will retain this rejected certificate? (Awkward!) Can the guest's machine connect directly to the delegator's machine via ad hoc wireless networking? (Intriguing, but not for version 0.)

We finally decided to borrow a page from the standard vendor literature. Our networking staff assure us that's it's not unreasonable for us to assume the access point/authentication server can have two logical wires coming out of it: traffic from authorized supplicants goes to the internal network, and other traffic goes (via a VLAN) to a special Web page. (For example, the RADIUS server might accept the supplicant, but with the proviso that supplicant be routed only to a specific VLAN; the only thing on that VLAN is our Web server.) If the guest fires up a Web browser, the only thing they can reach is our server. A special page there will then have links for a "get delegated" service, which permits the guest to upload their certificate (with minimal or no user work, hopefully).

The second part of the challenge is more subtle. In the delegation scenarios we envision, the delegator and guest might be sitting across from each other at a table, and decide to start this process. The guest uploads their certificate to a backend server, which stores in some temporary cache. (The certificate is deleted when delegation happens, or when some timeout occurs.) The delegator connects to the server and starts a delegation process. The backend server may have many certificates in its cache. How does the delegator confirm that they have the right one? (One colleague imagines a scenario where the adversary overhears that guest Gary requests authorization, and quickly generates a self-signed certificate with Gary's identity information and uploads it to the cache.)

To solve this problem, the guest and the delegator both need to verify that they are looking at the same public key: the same pair of long numbers. Having humans read digits to each other would be neither usable nor reliable.

Giving the Guest a Keypair. A key component of our strawman solution is that the guest already possess a keypair and an X.509 identity certificate for the public key. (But again, note that we do not particularly care about who signs this certificate or the exact contents of the various fields.) In the long run, we expect that this will be the normal case—particularly as universities and enterprises migrate to local solutions like EAP-TLS. Because all we care about is the public key and the ability of the client to use it, we overcome most of the problems of interoperability. (If the guest already has a keypair, we want to use that one; having too many personal keypairs leads to poor usability.)

In the short term, however, we need to see that guests who don't already have a keypair and certificate get them. Since we don't care about the issuer, even a self-signed certificate will do.

Here again, we sketched out many possible solutions, depending on the information path we have from the Dartmouth system to the guest's machine. The solution we currently settled on uses the same path we assume for the public key transmission: the unauthorized guest can connect to a special Web page. We can then use standard Web techniques to have the guest's machine generate a keypair and receive a certificate.

Generating Certificates. The above processes require that we able to compose SPKI/SDSI certificates that talk about the public key in an X.509 identity certificate, and then sign them. (Section 7 will then require that we verify them, as well.) Signing the guest's certificate creates an additional challenge: we need to do this with the delegator's private key, which probably lives in their browser-based keystore.

For the prototype, we can set up a special case of this tool for the "source of authority": a designated individual will play this role, and sign delegator certificates with a keypair dedicated this purpose.

Storing Certificates. The delegation backend needs to store uploaded guest certificates in a temporary cache, as noted above. The signed SPKI/SDSI certificates (guest and delegator) then need to be put somewhere so that the modified RADIUS server can fetch them when necessary. We'll consider the structure of that latter store in Section 7.

Putting the Pieces Together. As one way of putting all these pieces together, we could build a SPKI/SDSI signing tool that then gets used in two settings.

In the basic tool:

- Alice connects over the Web (and server-side SSL), uploads her X.509 certificate, and requests signing.
- The server caches this certificate, and displays a visual hash.
- Bob connects, and asks to sign a certificate for Alice. (As part of this connection, Bob uploads his X.509 identity certificate as well.)
- The server lets Bob select one (via one of the options discussed above) and displays a visual hash for confirmation.
- If Bob confirms, the server composes the SPKI/SDSI certificate for Alice's public key, and sends this back to Bob's browser to be signed with Bob's private key.

In the first setting, we use this tool to generate the delegator certificates. We designate an individual as the wireless source of authority, and set them up with a special keypair for this purpose. Per policy, Dartmouth users who wish to act as delegators then contact this person to get the a SPKI/SDSI delegator certificate.

In the second setting, we use this tool to generate delegation certificates. Unauthorized guests try to connect to the wireless network, but get shunted to the special page. This page helps them create a keypair and X.509 certificate, if they don't already have one. This page then leads them to the above tool; they request authorization, and a Dartmouth delegator connects and signs their certificate.

In the long term, we plan to explore many extensions and variations, such as local sources of authority controlling local access points, less trivial delegation policies, and other ways of authorizing delegators than the above face-to-face scheme.

7 Making the Decision

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

Dartmouth Users. In the initial case, ordinary Dartmouth show authorization by (via EAP-TLS) proving knowledge of a private key matching an X.509 identity certificate issued by the Dartmouth CA. Figure 1 illustrates this case.

Guests. Authorized guests also show authorization by (via EAP-TLS) proving knowledge of a private key matching an X.509 identity certificate. However, we use a different process to decide whether the certificate is acceptable; we must find:

- a valid SPKI/SDSI delegator certificate, whose signature can be verified against the public key of the Dartmouth wireless source-of-authority;

- a valid SPKI/SDSI guest certificate speaking about the public key in the guest's X.509 certificate, and whose signature can be verified against the public key in the above delegator certificate.

Figure 2 illustrates this case.

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 10.

The Algorithm. Putting it all together, the modified RADIUS server follows a procedure such as this:

- We start the supplicant with EAP-TLS.
- If the supplicant cannot present an identity certificate, we shunt them to the special VLAN; if the guest has a Web browser running, they'll see the "we don't know you" page.
- If the supplicant can present an identity certificate, we then evaluate it:
  - If the certificate is valid and issued by the Dartmouth 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 the reject page.
- If we accept the certificate, and the supplicant proceeds to prove knowledge of the private key, then we let them in.
- Otherwise, we shunt the supplicant to the reject page.

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

Getting the Certificates (Centralized Approach). To carry out the Figure 2 case, the RADIUS server needs to know the X.509 identity certificate, the source-of-authority public key, and the SPKI/SDSI certificates. EAP-TLS gives us the first, and we can build in the second. But how do we get the 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. 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.

Getting the Certificates (Decentralized 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.

We note that Web cookies will provide most of the functionality we need.

- 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 as a cookie, and the store that and the new authorization certificate as cookies as the guest's machine.

The only remaining question would be how to get these two cookies 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 this X.509 cert. If none can be found, the supplicant is routed to a special Web page, that will pick up their two certificate cookies (this requires the guest must have a browser running) and save them in the store.

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 after being delegated.

(We plan to explore other decentralized approaches as well.)

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.

8 Project Status

Currently, we have a working prototype of Greenpass, for delegators and guests with standard Windows platforms.

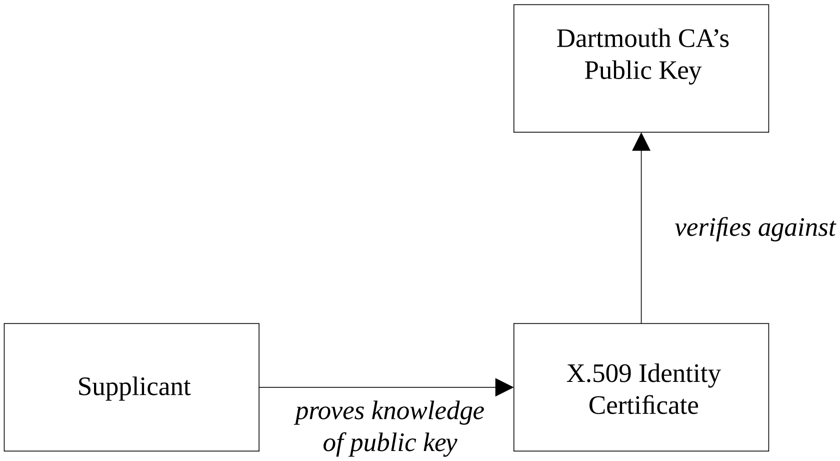
Dartmouth CA's Public Key
verifies against
Supplicant
proves knowledge of public key
X.509 Identity Certificate

Figure 1: Ordinary Dartmouth users prove authorization via an X.509 identity certificate issued by the Dartmouth CA. Users do this with TLS, with standard OS keystores.

8.1 The Certificate Center

If a user possesses a SPKI certificate that grants him WLAN access *and* allows him to delegate this privilege, then he may allow a guest onto the WLAN by using our *certificate center*. Delegation requires the following three steps:

- The guest must *introduce* his public key (i.e., his SPKI identity) to the delegator.
- The delegator must construct and sign a SPKI certificate that authorizes the guest to access the WLAN.
- The delegator or guest must make the RADIUS server aware of this new authorization certificate (this populates the RADIUS server's certificate cache).

This section describes how the certificate center allows the delegator and guest to complete these steps. The guest interface to the certificate center is currently implemented as a group of CGI scripts written in Python and accessed through an Apache web server; delegators construct and sign SPKI certificates using a trusted Java applet.

Key introduction

Before the delegator can authorize the guest to do anything, he must obtain the guest's public key (recall that SPKI uses public keys as unique identifiers). We identified three cases that might apply here:

- The guest may have a Web browser which supports SSL or TLS client authentication, and his home organization may have already issued him a certificate to use for this purpose.
- The guest may not have a certificate. We then need to generate a temporary one for him to use for EAP-TLS authentication.
- The guest may simply have a PEM-formatted certificate stored on his local filesystem; we allow him to upload it to our Web server.

Web browsers which support SSL or TLS client authentication include Microsoft Internet Explorer, Mozilla, and recent versions of Netscape Communicator. These browsers typically maintain a built-in keystore from which they retrieve certificates to present to websites which request client authentication. If the guest's home organization issued

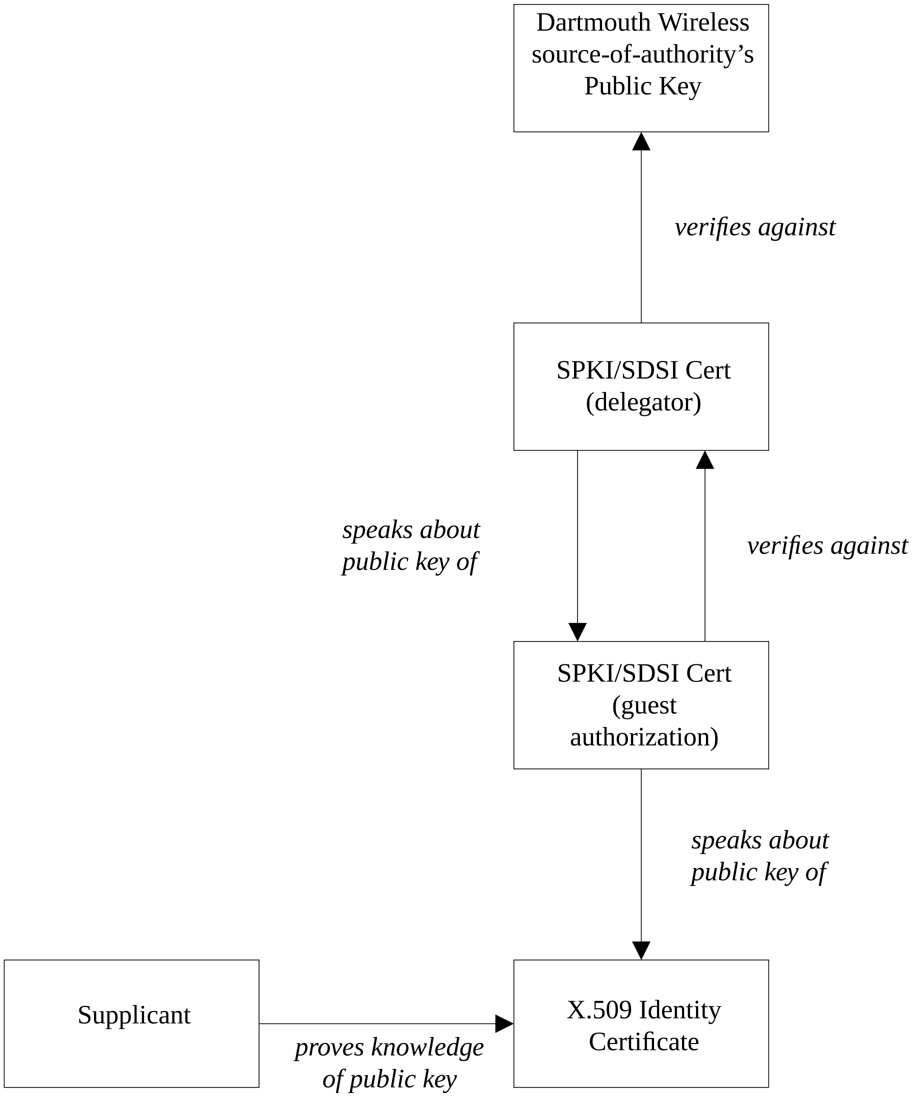
Dartmouth Wireless source-of-authority's Public Key
verifies against
SPKI/SDSI Cert (delegator)
speaks about public key of
verifies against
SPKI/SDSI Cert (guest authorization)
speaks about public key of
Supplicant
proves knowledge of public key
X.509 Identity Certificate

Figure 2: Guest users prove authorization via an X.509 identity certificate supported by a SPKI/SDSI chain. Again, users do this with TLS, with standard OS keystores—we do not need to change how the user platform or the network protocols work to accommodate SPKI/SDSI.

him an X.509 certificate for some other purpose, it is still likely that the web browser will have access to it: Mozilla and Netscape share browser keystores with their S/MIME email components, and IE retrieves certificates from a centralized keystore provided by Windows.

Our certificate center obtains the client's certificate from his Web browser as follows. The entire introduction session takes place over an SSL connection, and we configure Apache to request SSL client authentication. (We configure Apache to *request*—not require—client authentication; otherwise it might prevent a guest from connecting if he did not yet have a certificate to present.) If the client presents a certificate, Apache places it in the `SSL_CLIENT_CERT` environment variable, where a CGI script can read it.

Normally, when Apache is configured to request a client certificate, it will automatically validate it; i.e., a client will be served a Web page only if his certificate was signed by a CA which Apache has been configured to trust. This is a problem for us, because guests may arrive from a number of different organizations whose CAs we may not trust or even have knowledge of. Fortuitously, Apache provides a rarely-used option which tells it to collect a client's certificate, challenge him to prove knowledge of the corresponding private key, and pass this certificate on to a CGI script successfully *even* if the certificate's issuer is unknown. This works perfectly for our purposes, because we only need to extract the client's public key for use in a SPKI certificate; access control does not depend on our trusting any CAs.

Once Apache passes the client certificate to a CGI script, the script must process it. We used SWIG [Swi] to access the OpenSSL library from within our Python scripts. Using OpenSSL, we extract the distinguished name information from the client certificate, then place this information and the original encoded certificate in a temporary database. A delegator can later search this database to find the certificate of the guest he wishes to allow onto the network.

Guests who do not already have an X.509 certificate require slightly more effort on our part; we need to generate keypairs for them and issue corresponding certificates. Netscape and IE both support keypair generation. Netscape and Mozilla support a `<KEYGEN>` HTML tag. When the user submits a form containing this tag, Netscape generates an RSA keypair; it stores the private key in the user's keystore and submits the public key value to the Web server along with the rest of the form data. (In our case, the guest uses the form to input his common name and organization so we can attach some identifying information to the new certificate.) A CGI script that is part of the certificate center processes this public key value and places it into a short-term X.509 certificate; a dummy CA (trusted only for this purpose) signs it. The CGI script then sends a confirmation Web page back to the user along with the new certificate. The user's Web browser automatically recognizes the fresh certificate and matches it with the private key it recently generated. IE provides an ActiveX object that supports the same functionality, but needs to be called by a small segment of JavaScript code.

If the guest's Web browser does not support SSL client authentication, or if he uses his X.509 certificate for some other purpose and simply hasn't installed it in his Web browser, then he can simply upload the certificate from a file on disk. Our CGI scripts process the uploaded file exactly as they would an SSL client certificate passed in an Apache environment variable.

Delegation

The delegator must perform the following steps in order for his guest to be allowed onto the WLAN:

- He must visit our certificate center and search for the guest's X.509 certificate so he can extract the guest's public key from it.
- He must verify that the certificate he finds is really the guest's certificate and not one that was injected by an impostor.
- He must build and sign a SPKI certificate for the guest.
- He must upload this fresh authorization certificate to the RADIUS server's database.

The delegator searches for the guest's certificate using a simple web interface; he enters information such as the guest's common name or organization. The certificate center matches this identifying data against the subject names on certificates in its database, and presents the delegator with a list of certificates that match his criteria. The delegator chooses the certificate that appears to belong to his guest.

Before proceeding, the delegator must verify that the certificate he has chosen really *does* belong to the guest. Dohrmann and Ellison faced a similar *introduction* problem when investigating the formation of collaborative groups in ad-hoc networks [DE02]. We have adopted their basic solution, *visual hashing* (also discussed by Perrig and Song [PS99]). After a guest uploads his certificate, the certificate center computes its MD5 fingerprint and transforms this hash value into a visual representation, which it then displays to the guest. The delegator is shown this same image before he delegates any privilege to the guest, and he must verify that the image on his screen matches that on the guest's screen (it is faster and easier to compare two visual hashes than two compare two hexadecimal hash values). This step allows the delegator to bind a public key to the actual person he intends to delegate to. The visual hash we use is generated by the Visprint program [Gol, Joh]; this is a C program which we are currently porting to Java for use in our delegation tool (see below).

The third step, signing a new SPKI certificate, is particularly problematic. Building the certificate is fairly straightforward; we use the Java SDSI/SPKI library built at MIT [SDS]. (Parsing the guest's X.509 certificate is handled by cryptography classes that are part of the standard Java 2 platform.) For our delegation tool to *sign* a SPKI certificate, however, it must have access to the delegator's private key. Web browsers do not allow Web servers to access a user's private key: this would allow malicious websites to sign arbitrary statements which seem to come from the user!

We considered a number of possibilities for our delegation tool. We initially considered `capicom.dll`, an optional ActiveX component that Microsoft provides for Windows developers. This component provides a JavaScript function within IE that, after displaying several warning messages, signs a string provided as an argument. We decided that installing this component on an organization's machines is far too risky: a malicious Web site could trick the user into signing arbitrary strings.

We considered writing a standalone application to support delegation. Since only users *within* an organization will normally delegate, it is reasonable to install a simple delegation utility on each user's computer. This approach, too, has its problems: it requires rewriting the tool for each platform that delegators might use, and it does not integrate particularly well with the Web-based portion of our certificate center. A slight variation on this approach would have been to implement the application as an XPCOM component that would run inside Mozilla (or Netscape); we would only need to recompile, not rewrite, this component for new platforms. We dismissed this idea also: it provides good integration with our web-based interface, but forces all delegators to install Mozilla. We did not wish to force users to install a particular browser just to delegate.

Currently, we are testing a delegation tool implemented as a Java application; it is acceptable in this form, and we hope to transform it into a *trusted applet* in the future. A trusted applet is signed by some entity; a Web browser will run it if the user has marked that entity's certificate as trusted. It is entirely reasonable to expect that an organization can install this certificate on all its home users' devices and mark it as trusted (alternatively, users can download the certificate while connected via a private LAN that is known to be secure). Trusted applets have greater access privileges than untrusted applets; in particular, they may access the user's local filesystem. This means that our applet will be able to load a user's keystore and, after requesting a password, unlock his private key.

Once the delegator finds the guest's certificate using our Web-based interface, a CGI script will generate a web page that runs our delegation applet. The HTML `<APPLET>` tag can provide an applet with an argument; our Web page will provide the guest's encoded certificate as an argument to the applet. The applet will display the identity information in the guest's certificate so the delegator can verify it and, as a final check on the authenticity of the certificate, will display its Visprint-generated visual fingerprint. After verifying this fingerprint, the tool will use the delegator's private key to sign a freshly-generated SPKI certificate.

One remaining problem is to allow the applet to access the native keystores on a delegator's machine. Currently, the delegator must export his certificate and private key to a PKCS12 bundle, which we import using the standard Java security extensions. Our planned approach is to provide a small glue library for each common platform that delegators might use; this library would use JNI (the Java Native Interface) to access the Mozilla/Netscape keystore, the Windows keystore, or Apple's Keychain. This modification would add a degree of convenience and ease for the delegator, and would not require him to store his private key in multiple locations.

SPKI certificate upload

At present, we transfer the newly created certificate to the RADIUS server manually. Once we have finalized the implementation of the modified RADIUS server by integrating SPKI verification code into the server itself (right now, it uses XML-RPC to query a Java-based verification server), we plan to specify an XML-RPC interface over which the RADIUS server will listen for new certificates to add to its cache. The delegation applet will then upload new certificates directly via this link.

Ultimately, we would like to decentralize our design further by allowing the guest to present his certificate chain directly to the RADIUS server using HTTP cookies, as discussed earlier.

8.2 The Server Side

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 switch and a hub to connect the two machines running the RADIUS server and web server.

Setup

Currently we have the access point configured to have two 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 configured the access point to query the RADIUS server for user authentication. The shared secret between the access point and the RADIUS server was provided in the configuration so they can communicate. The RADIUS server is connected to VLAN 1, the native VLAN. The Web server is connected to VLAN 2, the limited VLAN. The hub connects the two machines and allows them to communicate to one another through their private connection. This was necessary because we did not have a router capable of VLAN trunking, which would essentially allow the machine to communicate through either VLAN. Without such a router, each machine could only be connected to a single VLAN, and the private connection was needed to work around this.

Configuration

We downloaded OpenSSL and installed it, and linked the libraries to be used by FreeRADIUS as directed by the HOW-TO references [Ros03, Sul02]. (There was a problem with the Kerberos libraries because under Red Hat 9, the include files for Kerberos 5 required by OpenSSL were placed under `/usr/kerberos/include`, rather than `/usr/include/kerberos`, which OpenSSL was expecting. Creating a symlink between the two directories solved the problem.)

We then used OpenSSL to generate a trusted root certificate, a server certificate, and client certificates with the scripts provided by the HOW-TO references.

We changed the configuration file (`radiusd.conf.in`) for FreeRADIUS to disable other EAP protocols and set TLS as the default EAP type to use. The location of the CA certificate chain, RADIUS server certificate, private key, and private key password, and a random number generator were provided for use with EAP-TLS. The clients file was changed to include APs with a Dartmouth IP address, providing the shared secret shared with the RADIUS server, so our access point can act as an NAS for the RADIUS protocol. The users file was changed to include a profile for DEFAULT users (i.e., any user) using EAP authentication, providing attributes needed to assign a VLAN to the user.

We then downloaded and installed the XML-RPC libraries. We modified the Makefile for the RADIUS server to link the XML-RPC libraries. This is used in the SDSI/SPKI chain verification process.

The Process

The RADIUS server idles and waits for packets. When it receives a 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. All the different authentication types are organized into modules in the source code, and the whole process is separated out very cleanly.

Once the EAP-TLS module is done, the decision to accept or reject the supplicant has already been made and is packed into the response packet. So the modifications needed to allow for a SDSI-SPKI chain are made in the EAP-TLS module while the authentication decision hasn't been made yet. Under EAP-TLS, the supplicant presents a X.509 certificate to the RADIUS server. The RADIUS server checks the certificate to see if it's valid and uses OpenSSL to establish an SSL session. If this fails, the user is rejected.

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 is good and the supplicant is presumably a valid user with a X.509 certificate issued by the known CA. However, if it failed, we use XML-RPC to query the Java SDSI library (currently running on the certificate center machine, discussed above) about the public key of the X.509 certificate provided. The library uses the proposed SPKI/SDSI certificate chain discovery algorithm [CEE$^+$01]. If the Java code finds a valid SPKI certificate chain vouching for the supplicant, then we accept the supplicant and change the error code to report that the certificate is valid. Currently, we have a cache of SPKI certificates, and this decision is based on whether there are SPKI certificates in the cache to form the necessary chain. If the Java code cannot find such a SPKI cert chain, then the user is rejected.

9 Related Work

Balfanz et al [BDS$^+$03] propose using secret keys to let wireless parties authenticate. We've already noted related work [BSSW02, 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. Koponen et al [KNRP00] proposes 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.

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.

10 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 vs guests. (perhaps "permanent Dartmouth staff" vs "students likely to lose their expensive dongles while skiing").

Not just the network. Many types of digital services use X509 identity certificates as the basis for authentication and authorization. For example, at Dartmouth, we're migrating many current Web-based information services to use X509 and client-side SSL/TLS. In the Greenpass pilot, we're adding flexibility to wireless access by extending X509/TLS with SPKI/SDSI. This same PKI approach can work for networked applications that expect X509, 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* 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.

Attacks. 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 which ensures that the delegator cannot skip this step, 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 us allow us to eliminate human interaction from the introduction phase in the future. We also might consider alternative methods of hash comparison, so that, for example, the delegator could verify the guest's fingerprint during a phone conversation rather than visually.

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 WiFi 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 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, did 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 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 include the X.509-based PERMIS attribute certificate system might work in this setting [COB03, Per], as well as KeyNote [BFIK99, Key]. Nazareth [Naz03] gives an overview of many such systems.

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.

References

[AS99] B. Aboba and D. 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.

[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*, 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.

[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.

[EA04] Jon Edney and William A. Arbaugh. *Real 802.11 Security*. Addison-Wesley, 2004.

[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, draft-ietf-spki-cert-structure-06.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.

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

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

[HA03] Russ Housley and Bill Arbaugh. Security Problems in 802.11-based Networks. *Communications of the ACM*, 46(5):31–34, May 2003.

[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, May 2004. (expected).

[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.

[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.

[PS99] Adrian Perrig and Dawn Song. Hash Visualization: a New Technique to improve Real-World Security. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC) 1999*, 1999.

[Rig00a] C. Rigney. RADIUS Accounting. RFC 2866, June 2000.

[Rig00b] C. Rigney et al. RADIUS Extensions. RFC 2869, June 2000.

[Rig00c] C. Rigney et al. Remote Authentication Dial In User Service (RADIUS). RFC 2865, June 2000.

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

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

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

[Swi] The SWIG Project. `http://www.swig.org`.