

Public Key Distribution and Authentication Using LDAP

Prentice Bisbal

Department of Electrical and Computer Engineering
Rutgers University

Introduction

Public-key cryptography provides the solution to two important computer security issues: secure communication and identification. It provides two entities the ability to communicate securely using encryption without first having to establish a secure channel to exchange an encryption key. It also provides the power to positively identify and authenticate the person or computer you are communicating with.

Public key cryptography was introduced by Diffie and Hellman in [1]. It was independently discovered by Merkle at approximately the same time [2]. Rivest, Shamir, and Adleman provided the first secure algorithm for public key cryptography and digital signatures in [3].

In order for public-key cryptography to work, the two parties involved in a communication need to know each other's public keys prior to communicating. Although these keys can be distributed publicly without weakening the security of the keys, the parties involved in a communication still need to know how and where to retrieve the keys, and be able to verify the public key really belongs to the person or computer they wish to communicate with.

Although public keys can be used to authenticate users, this is not done as much as it can or should be. Some utilities, such as ssh, allow a user to authenticate as he moves from machine to machine, but it is highly likely that he authenticated with the computer he is sitting in front of using nothing more than a simple 6 to 8 digit password.

In this paper, it is argued that public keys should be distributed through a distributed, hierarchical directory system similar to DNS, and that this can be done using the LDAP protocol and existing standards. Finally, a method for authenticating users at initial login using public keys, using LDAP is demonstrated.

Public Key Infrastructure – A Brief History

In [1], [2], and [3], it is suggested that public keys could be stored in a central repository where a user's or server's public key could be retrieved from. Diffie and Hellman [1] called this central database the Public File, and suggested that the public keys be associated with a keyholder's name and address. Although a great idea, there are several problems with this approach.

1. Single point of failure - This central directory would present a single point of failure. If there is a blackout, a network outage, or even a war, the whole directory could be taken offline, preventing users around the world from retrieving public keys
2. Administration and maintenance - Who would be responsible for maintaining this directory service? Who would pay for it?
3. Server and network load – It would take an enormous computer system to be able to respond to all the queries that could come in from all over the globe. Not only would a very fast server be needed, but the network link would need to be large enough to prevent saturation.
4. Name space collisions – How would public keys be looked up? Diffie and Hellman [1] suggested names and addresses. Many people are uncomfortable publishing their

name and address online. This would require knowing someone's full name and address in advance. Knowing someone's name is more likely than knowing their full address. With a (possibly) unique name like Prentice Bisbal, this might work, but with a common name like Robert Smith, this would not work. As the number of key-pair holders grows, the probability of name collisions increases.

Loren Kohnfelder identified these problems with a public-key directory service as an undergraduate student at MIT [4]. In his bachelor's thesis, he introduced digital certificates. A digital certificate is an electronic document containing a public key. This document is signed by a trusted party known as a *certificate authority*, or CA. By signing this certificate, the CA is signifying that it has thoroughly checked the identity of the person or organization possessing the private key corresponding to the public key presented in the certificate.

Since the identity of the key owner has been verified by a CA, and cryptographically signed with the CA's private key, the key owner can provide his public key and certificate to anyone who he wished to communicate with. Any recipient of the certificate can verify the signature of the CA if they know the CA's public key. This requires that the CA's public keys will be well-known and readily available. This eliminates the need for a public directory, since the burden of providing the public key has been transferred to the keyholder.

Public Key Infrastructure – The Present

The current public key infrastructure is based on the certificate/certificate authority model introduced by Kohnfelder. Companies have formed whose entire business is based on verifying identities and providing public key certificates. These large companies have established business relationships with software publishers to have their public certificates included with the software. This is most common with web browsers, since public key encryption is often used to encrypt communications for online shopping through SSL.

This arrangement isn't perfect, and it has many critics. Ellison and Schneier [5] provide ten criticisms of this system. I won't re-list all ten of their complaints, but I will mention two that I think are the most serious flaws with this system: Who do we trust, and why? How did the CA identify the certificate holder?

Trust is an interesting concept. Some people can go through their whole life and never trust anyone, while others are willing to trust just about everyone one they meet. In their world of cryptography, as Ellison and Schneier [5] explain, trust has a more specific definition. In cryptography terminology, trust means that a CA handles it's own private keys well. This means that we can trust that a document bearing a CA's digital signature was signed by that CA.

Just because we can trust a CA to manage it's private keys securely, that doesn't mean we can trust what they say. This leads to the second question. How did the CA identify the certificate holder? Can we trust that the CA was thorough in identifying the keyholder?

If it's private individual representing only themselves online, providing a driver's license, birth certificate, or passport might be enough. These documents used to identify an individual are relatively trustworthy when presented in person. State governments in the United States have spent a considerable amount of money to make it hard to falsify driver's licenses, the de facto standard for proving identity in the US. The federal government of the US has done the same to prevent false passports. Other governments around the world have also addresses the same challenges. Any doorman at a nightclub, or bartender has learned the how to spot a false driver's license fo the state they live in, and possibly any neighboring states.

Unfortunately, it's impractical to have everyone requesting a certficate to show up in person at a CA's office with the original documents themselves, photocopies or facsimiles are provided to the CA, making forgeries trivial.

What if that certificate is for business use, and the certificate holder will be using it to represent a company? Examples of this would be a spokesperson who would like to digitally sign press releases, a purchasing officer who will need to digitally sign electronic payments, or a system administrator needing to provide a certificate for the web server of the online shopping system.

In addition to the documentation listed earlier for a private individual, how can a CA be sure that

1. The keyholder really is employed by the company
2. The keyholder is employed in the role they present to the CA

The situation with corporate ID cards is much worse. They aren't nearly as sophisticated as government ID cards. Also, few people outside a company even know what they should look like. While the security guard at the main entrance might be able to spot a fake, the same doorman or bartender wouldn't even know what a correct one should look like.

Often, a letter on official company letterhead is used as proof of employment, and the contents of a letter on company letterhead is taken as official and authoritative. The problems with this should be obvious. In most companies, anyone can get access to company letterhead. It's also trivial to save an image of the logo from a company website and, using retail desktop publishing software, create your own stationery with the company letterhead using a laser or inkjet printer.

Ellison and Schneier [5] argue that CAs need PKI in order to make money, not the other way around. Why should someone have to pay an outside company to prove their own identity? If I want to start a business selling items through a web page, I am forced to pay a CA in order to provide customers a secure web page. When an encrypted connection starts, the server provides its certificate to the client browser. That browser then checks the signature against the CA public keys built into it. If the certificate is not signed by a CA the web browser knows about, it will display a warning that the identity of the web server can't be verified, and shouldn't be trusted. This will turn away some potential customers. Not providing encrypted communication for ordering will turn away even more.

The name public key infrastructure is itself misleading. The current CA model doesn't really provide an infrastructure for distributing public keys. If a CA's private key gets compromised, there is no way for them to quickly distribute a new key or distribute a certificate revocation list (CRL) including the revoked key.

A CRL is a list of certificates issued by a CA that should no longer be accepted as valid for one reason or another. There are arguments against CRLs. Rivest [6] shows that CRLs can be eliminated completely. One problem with CRLs is that they are not issued frequently enough. They are often released periodically. It is then up to a client application to download a CRL from a CA, which it does periodically. Having any one of these actions occur periodically introduces a significant delay between when a certificate is revoked and a client is made aware. Having both actions occur periodically drastically increases the latency. During that period, a hacker could impersonate a legitimate online store, collecting credit card numbers and charging accounts.

Based on the current model, if a CA key is compromised, they would have to contact all the providers of PKI-aware software and have them issue updates to their software. Most computing professionals may learn of the problem and update their software, but what percentage of the general public will learn of this and update their software? How many will even care?

Clark [7] states that one of the design goals of the Internet was to distribute management of its resources. This was to ensure that the network could survive a war or other disaster. Although there are multiple CAs, there are several that are much more well known than others. As a result, these popular CAs own the lion's share of the market. Correspondingly, the majority of

the certificates in circulation are signed by only a handful of CAs. I would not consider distributed management of certificates. If just one or two major certificate authorities was rendered inoperable (during a war, for example), this model would collapse.

One of the most powerful features of public-key cryptography is that it provides for very strong identification. If you encrypt a message with someone's public key and they can unencrypt it, or they send you an encrypted message and you can decrypt it with their public key, you can be sure that person has the private key matching the public key. This can be used to provide an authentication mechanism much stronger than passwords. But current public key infrastructures doesn't provide a method to identify and authenticate a user using public key cryptography.

Public Key Infrastructure – The Future?

When the hosts file that contained the mapping of hostnames to IP addresses became too big to manage, the Domain Name Service (DNS) was created. DNS is a hierarchical, distributed directory service that provides hostname to IP address mappings. Organizations with their own domain on the Internet are responsible for maintaining their own hostname and IP address information. The scalability and flexibility of DNS easily accommodated the explosive growth of the Internet in the 1990s.

For a host on the Internet to use the DNS system, it only has to know the IP addresses for its local nameservers. When it needs to get the IP address for a hostname, it queries the local nameserver. If the hostname is not local, the nameserver is able to find what nameserver has that information, query that nameserver, and return the IP address to the host that initiated the query.

The distributed management nature of DNS can, and should, be applied to public key distribution. Each organization would be responsible for maintaining its own key information. Just like with DNS, the key serving can be split into subdomains with key serving delegated to additional servers accordingly. Keys can then easily be added or removed from a keyserver, just like adding or removing a DNS entry for a host from a nameserver.

The key information would be online, and could be instantly retrieved by any computer configured to use this key distribution protocol. This would ensure that the key information retrieved is the latest and most accurate. As Rivest points out in [6], the existence of a key should be all that's needed to prove it's valid.

Since the keys will be supplied directly by the source, in an ideal world certificates would no longer be needed. Unfortunately, poorly managed systems exist and system compromised systems do occur. To prevent an intruder from spoofing public keys, the public keys should be delivered in certificates signed by the organization's own key. In effect, each organization would become its own certificate authority. It would be up to each organization to maintain its own certificate-signing keys.

This raises the question of how to distribute an organization's CA certificate in a secure manner. The keyserver immediately above and below a keyserver could maintain a copy of the certificate for that key server. As a client traverses the hierarchy searching for a key, the keyserver that directs the client to the next server would also provide the key for that keyserver.

For example, a client in the domain caip.rutgers.edu is requesting the key for nb.bisbal.us. caip.rutgers.edu refers the client to the rutgers.edu keyserver, and provides the certificate for rutgers.edu. Since rutgers.edu doesn't have the key, either, it refers the client to the keyserver for the .edu top-level domain, and provides the certificate for the .edu keyserver. This server refers to the root keyserver and provides the certificate for the root keyserver. This same process continues down the hierarchy until the keyserver for nb.bisbal.us is reached.

This may seem complicated, but in reality, it isn't. A keyserver's certificate only needs to be distributed to two other keyservers. With DNS, entries must be made to specify other nameservers as well as the addresses of the root servers. In such an arrangement, an intruder

would have to compromise at least two servers to spoof a keyserver. Most likely those keyservers would be maintained by completely different organizations and on different networks, making the task that much more difficult.

Allowing organizations to manage their own keys has several benefits. When an organization needs to put a new key into circulation, it can do so immediately. No longer does a CA middleman need to be involved, along with its associated costs and delays. The reverse is also true. As soon as an organization realizes a key has been compromised, it can immediately remove it from its key servers.

Another benefit of this arrangement is that the individual organization is the best authority of who or what should be representing that organization. For example, if a company hires a new employee, it can positively identify that person as an employee in person, issue the new employee a private key, and publish the corresponding public key certificate on their key servers. By meeting the keyholder face-to-face, and inspecting actual identification documents presented, a much more thorough identification can be done than a CA can do.

Two of the main problems cited with the Public File key distribution envisioned by the pioneers of public key cryptography were being able to uniquely identify the keyholder whose public key you were trying to retrieve, and the load placed on the Public File server. This arrangement also solves both of these problems.

E-mail addresses and hostnames on the Internet are guaranteed to be unique. If they weren't, mail wouldn't reach the intended destination, and packets would never reach the intended hosts. These work because an organization's domain name, as assigned by a registration authority which guarantees that no two organizations can have the same domain name. It is then up to the organization to make sure that hostnames and e-mail addresses within the organization are unique. By combining the unique internal component of an ID with the unique, external domain name, a unique hostname or e-mail address is guaranteed.

The current CA-based PKI uses e-mail addresses or fully-qualified hostnames to match a certificate to a user. This is fine as long as an individual only uses one e-mail address, or a host only uses one hostname. This breaks down immediately when an individual sends e-mail from multiple addresses, or a computer can be identified by multiple hostnames (this is common for web servers, particularly when a web server is maintained by a web hosting service). Since a certificate has the corresponding hostname or e-mail address embedded in it, the client application will issue a warning if the e-mail address or hostname the certificate came from doesn't match what is embedded in the certificate. The solution is to create a new certificate for each e-mail address or hostname that will be using that key. This is cumbersome, and a better solution is possible.

Thus far, I haven't mentioned user authentication in this hypothetical system. When an individual joins an organization, his private key is given to him on a hardware token that can be read by any computer. There are no complicated security features in the hardware design of this token. The private key is protected by a strong passphrase, and has no identifying information stored in it, or along side it on the hardware token. Even if the passphrase is guessed by brute force, the cracker will still not know who the key belongs to, and therefore cannot impersonate the key owner. To do so would require possession of the corresponding public certificate or the identification information necessary to retrieve the public key from the keyserver.

In this system the both the public and private key of a keypair would exist in only one place (ignoring securely stored offline backups). The public key would only exist on the key server, and the private key would be in the possession of the keyholder. For a user, the key would exist on a small hardware token that would be treated exactly as a physical key. For a server or host computer, the private key would have to be stored in a file that only the server application or superuser could read. Since the private key for a server or host exists on a computer that is always on line, there is much greater risk of this key being compromised than the hardware key a

user possesses which only needs to be online during the authentication process.

When a user needs to authenticate, the user plugs the hardware token into the computer, and then provides the identification information necessary to retrieve the public key from the keyserver. The authentication program then retrieves the key and verifies that the public and private keys match. This can be done by simply encrypting a random string with one key and decrypting it with the other. If the encrypted/decrypted string matches the original, the keys match, and the user is authenticated. The user then removes the key and re-inserts it if necessary to sign documents or authenticate with another system. If the user wishes to authenticate with a remote system (through ssh, for example) the process becomes slightly more complicated, since the private key should never be transmitted over the network, whether it's encrypted or not.

PAL – Public-key Authentication with LDAP

Up to this point I haven't discussed any specific protocols or implementations of this system. To accomplish this vision, no new protocols need to be developed. The lightweight directory access protocol (LDAP), already provides the necessary functionality for distributing and retrieving information in a distributed, hierarchical manner as described above. LDAP is a well-defined, open standard which eliminates compatibility issues across platforms and organizations and allows for software products from various vendors. Version 3 of the LDAP protocol is defined by RFC 2251 [8]. Carter [9] provides an overview of LDAP, as well as instructions on setting up and maintaining LDAP servers.

LDAP is extensible, meaning the format of the data stored in it can be defined to meet the need of the application. These data structure definitions are called schemas, using terminology from the database world. A data structure stored in LDAP is called an object, and the members of this object are referred to as attributes. And just like a programming object, you can build larger objects out of smaller objects. This extensibility means that as key and certificates requirements change, new schemas can be defined or extended to meet the needs of the key infrastructure. Most attributes of objects can be multivalued. For example, an object representing a user can have 3 e-mail addresses stored in it instead of just one. The only attribute that must have only one value is the distinguished name (dn) which uniquely identifies that object.

Earlier, it was stated that one of the problems with the Public File concept was being able to identify a keyholder uniquely on the internet. LDAP uses a method similar to e-mail or DNS where a locally unique identifier (a hostname or username) is combined with a domain name to provide a globally unique name for an LDAP distinguished name. Describing the format of an LDAP distinguished name is beyond the scope of this paper. For an object representing the user Prentice Bisbal in the bisbal.us domain, the distinguished name might look like

cn=Prentice Bisbal,dc=bisbal,dc=us

where cn stands for *common name*, and dc is short for *domain component*. IETF RFC 2247 defines how to incorporate domain names into LDAP distinguished names [9]. With a standardized name system that guarantees uniqueness, it is possible to uniquely identify and retrieve a key anywhere on the Internet.

It should be pointed out that an LDAP object's dn doesn't necessarily need to be known to retrieve a key. LDAP searches can be done using any attribute, but the LDAP server must be configured to create indexes based on the various attributes. It's even possible to use the key as the index. For example, you could search based on e-mail address, or even a URL, or a phone number. This would still require knowing the domain of the LDAP server, that could easily be derived from the e-mail address or URL. Certificates would no longer need to say what e-mail address or hostname is associated with it. Retrieving the key using one attribute could return all

the other attributes for the keyholder, such as alternate hostnames or e-mail addresses.

The IETF also has RFCs that define several schemas that are of use to the Internet community. RFC 2798 defines the inetOrgPerson schema [10]. As the name implies, this schema contains common attributes for a person that is part of an organization connected to the Internet. In addition to basic information like name and phone number and office, this schema also has attributes for an e-mail address, a web page URL, and a user certificate. The user certificate attribute means that a new schema doesn't need to be defined to store a user's X.509 certificate in LDAP.

The final feature needed from LDAP in order to distribute keys across the Internet is the ability to setup a distributed, hierarchical directory structure similar to DNS. LDAP provides for superior and subordinate referrals, which provide this functionality. As the name implies a superior referral provide a URL to an LDAP server higher up in the hierarchy. This could be the next server up in the hierarchy, the server for a top-level domain, or a root server at the top of the hierarchy. Subordinate referrals provide URLs to LDAP servers further down in the hierarchy. They can also be used for distributing the load across multiple servers. RFC 2255 [11] defines the format of an LDAP URL.

RFC 3088 [12] defines an experimental LDAP root server run by the OpenLDAP project [13] that returns LDAP server referrals based on DNS server resource records. This mechanism allows an organization to publish information identifying its LDAP servers globally through its DNS records. Although it is possible to create an LDAP hierarchy independent of DNS, this experimental service allows early adopters begin creating a hierarchy. Anyone with a registered domain can participate, since this mechanism doesn't require the parent domain to participate.

LDAP can be used to create a global framework for distributing public key certificates, but there is still one piece of the puzzle missing. How can a user store his private key in a convenient, secure manner? An excellent solution are compact USB drives that have recently appeared on the market. These small, solid-state storage devices are small enough that they can fit on a keyring, similar to physical key, and are relatively cheap. A 16 MB capacity drive can be purchased for as cheap as \$12.

Most computers today have an abundance of USB ports in easily accessible locations. Most have them on directly on the keyboard, or on the monitor. This means no additional hardware needs to be installed on a machine to use these keys, unlike smartcards and biometrics, which require additional, more expensive equipment to be installed.

For a user to use this system, they would install the USB drive into any USB port they can access on the computer. Most operating systems automatically mount these devices (a standardized mount point and/or filename would be required). The user then provides their dn so the computer can retrieve the public key certificate and then verify the public and private keys match this can easily be done by encrypting a random string with one key, and decrypting with another. If the encrypted/decrypted string matches the original, the keys match, and the user can proceed.

To encrypt or decrypt with the private key, a passphrase is required, which adds another layer of security. Even if the key is lost, the passphrase would have to be determined before the private key could be used. These passphrases can be much longer than a normal password, making cracking a passphrase much more difficult. And unless the dn associated with the key is stored on the key, any intruder wouldn't be able to masquerade as the legitimate keyholder, anyway.

To prevent someone from eavesdropping on a communication channel and learning a dn, encrypted communication methods should be used, such as SSL. The private key should never be transmitted either, so to authenticate a user on a remote machine, the authentication becomes slightly more complicated. The remote machine would have to encrypt a string with the public key and then send it to the computer where the user is located to have it decrypted. After

successfully decrypting the message, it would be sent back to the remote system. Of course the communications between these two machines also needs to be encrypted, using the public keys of each host to establish an encrypted channel. Abadi and Needham [14] provide guidelines for developing such a protocol.

Since this system can be used for authentication and not just key distribution, I call this system PAL, which stands for Public-key Authentication using LDAP.

Applications for PAL

PAL has applications anywhere authentication stronger than a normal password is required. This system could compete with smartcards and biometric devices. Unlike smartcards, where the security is in not knowing the algorithm stored in the smartcard [15], the security in this system lies entirely in the key. This system could also compete with electronic one-time password systems.

Applications could be modified so that anywhere you would normally place a password (either in plaintext or encrypted form), you would instead put the dn for the users LDAP entry in the password field:

```
prentice:cn=Prentice Bisbal,dc=bisbal,dc=us:501:501:Prentice:/home/prentice:/bin/bash
```

Since LDAP can store the username, uid, displayname and other user information normally stored in a passwd file entry, we can streamline this passwd file entry even more:

```
cn=Prentice Bisbal, dc=bisbal,dc=us
```

Ssh is one of the few applications I know of that actually uses public key cryptography to authenticate users that is in widespread use. In order to use this feature, a key pair is created, and the public key is put in a file named `authorized_keys` on the remote system. The private key stays on the local machine. Using PAL, the `authorized_keys` file could be replaced with a file named `authorized_dn`, which would contain the dn of the authorized user. The advantage of this is that the private keys do not stay on the system where they can be accessed by an intruder, and it encourages users to have only one key pair instead of a key pair for every every system they connect to the remote system from. If a user leaves the company, only one key has to be removed. Likewise, if a new key pair is needed for some reason, the public key needs to be replaced in only one location.

This system could have applications in grid computing. In grid computing several different organizations share computing resources. This requires sharing or organizing authentication information between multiple organizations. This eliminates that. The participating companies can maintain their own authentication information, and make it available to the shared resources using LDAP.

Another benefit of this system is that it makes it easier for system administrators to remove access for a user when he leaves the company. Normally, passwords would have to be removed from all the systems as well as ssh `authorized_keys` files. All that is required in this configuration is removing the user certificate from LDAP.

Demonstration – pathauth.sh

To demonstrate the viability of the PAL system, the appendix contains a shell script, `palauth.sh` that performs a mock authentication. The USB drive must be inserted and mounted first. Then executing `palauth.sh` does the following:

1. Make sure the file `$HOME/.pal/authorized_dn` exists. This file should contain a dn for an `inetOrgPerson` object containing a `userCertificate` attribute.

2. Checks for the existence of the file .pal/private.pem exists in /mnt/usb (the mountpoint of the USB drive).
3. Searches for the LDAP entry specified in HOME/.pal/authorized_dn using the ldapsearch command.
4. Checks to see if (3) returned the object desired, or a referral to another LDAP server. If a referral was received, it does another search on the referenced server. It loops through this process until it no longer receives a referral to another server.
5. Extracts the user certificate in binary DER form using ldapsearch
6. Converts the certificate from DER to the usable PEM format using openssl.
7. Create a test string using the output of the date command.
8. Encrypts the test string with the public key and decrypts it with the private key and compares the result to the original string
9. Prints a message indicating whether or not the keys match.

This shell script was tested on a small network with two LDAP servers, one superior to the other. When executed on the subordinate LDAP server, the local LDAP server returns a URL to the superior server. It then successfully follows the URL and retrieves the user certificate. A better test would be to have retrieve the certificate from another server on a different network in a different domain, but do to the limited resources of graduate students home network, this was the best that could be done. Regardless, it demonstrates that using properly configured referrals, it is easy to find the proper certificate and use it to authenticate a user possessing the corresponding private key.

Conclusion

What we refer to as public key infrastructure is not a true infrastructure for key distribution. It doesn't allow for keys to be checked for validity or revoked in anything approaching real-time. It is also relatively centralized system, which goes against one of the fundamental design considerations of the Internet.

By learning from the design and success of DNS, it is possible to distribute keys in similar manner using standardized protocols that already exist for use on the Internet. This system would allow for quicker key distribution and revocation. It would also put the authority and responsibility of identifying an individual or server where it belongs.

All the pieces for building such a system exist and are readily available. Which lead to one question: why isn't anyone doing this already?

References

- [1] W. Diffie and M. Hellman “New Directions in Cryptography,” *IEEE Transactions on Information Processing*, 1976
- [2] R. Merkle, “Secure Communications Over Insecure Channels”, *Communications of the ACM* vol. 21, no. 4 (April 1978), pp. 294 – 299
- [3] R. L. Rivest, A. Shamir, L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” *Communications of the ACM* vol. 21, no. 2 (Feb. 1978), pp. 120-126
- [4] L. M. Kohnfelder, “Towards a Practical Public-Key Cryptosystem,” B.S. Thesis, supervised by L. Adleman (May 1978) [Note: This document is not published or available online, so it was not actually read in preparation for this paper. It is referenced in many other papers on public key infrastructure, including [6]. It is referenced here to the original document here to give credit where it's due.]
- [5] C. Ellison, B. Schneier, “Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure,” *Computer Security Journal*, vol. 16, no.1 (2000), pp 1-7
- [6] R. L. Rivest, “Can We Eliminate Certificate Revocation Lists?,” *Financial Cryptography*, 1445 (1998), pp. 178-183
- [7] D. Clark, “The Design Philosophy of the DARPA Internet Protocol,” In Proc. of ACM SIGCOMM '88
- [8] M. Wahl, T. Howes, and S. Kille, “ RFC 2251: Lightweight Directory Access Protocol (v3),” IETF Network Working Group, (December 1997)
- [9] S. Kille, M. Wahl, A. Grimsted, R. Huber, and S. Sataluri, “RFC 2247: Using Domains in LDAP/X.500 Distinguished Names,” IETF Network Working Group, (January 1998)
- [10] M. Smith, “RFC 2798: Definition of the inetOrgPerson LDAP Object Class,” IETF Network Working Group, (April 2000)
- [11] T. Howes and M. Smith, “ RFC 2255: The LDAP URL Format,” IETF Networking Group, (December 1997)
- [12] K. Zeilenga, “RFC 3088: OpenLDAP Root Service: An Experimental LDAP Referral Service,” IETF Networking Group, (April 2001)
- [13] <http://www.openldap.org>
- [14] M. Abadi and R Needham, “Prudent Engineering Practice for Cryptographic Protocols,” *Software Engineering* vol. 22, no. 1, 1996, pp 6-15
- [15] B .Schneier, *Secrets and Lies*, Wiley Publishing, Inc., Indianapolis, Indiana, 2004

Appendix 1- palauth.sh shell script

```
#!/bin/bash

# ldapauth.sh
# Shell script to demonstrate how authentication can be done by storing a
# public key in LDAP and the private key on a USB "thumb drive".

tmpfile=/tmp/.pal.$$

# Make sure key file exists
if [ ! -r $HOME/.pal/authorized_dn ]; then
    echo "ERROR: Cannot access $HOME/.pal/authorized_dn. You cannot be"
    echo "    be authorized this way"
    exit
fi

# Make sure key file exists
if [ ! -r /mnt/usb/.pal/private.pem ]; then
    echo "ERROR: Key file /mnt/usb/.pal/private.pem not"
    echo "    not found, or not readable."
    exit
fi

# Get my dn
mydn=$(cat $HOME/.pal/authorized_dn)

# See if this entry is in LDAP, check if it has a userCertificate, or
# we get a referral.
ldapsearch -x -b "$mydn" > $tmpfile 2>&1

# If exit status is 10, we've been referred to another LDAP server
result=$?
if [ $result -eq 10 ]; then
    referred=1
    while [ $result -eq 10 ]
    do
        echo "Referred to "
        # get referral URL and change commas to slashes
        ldapurl=$(grep ^ref: $tmpfile | awk '{ print $2 }' | sed -e s/,/\//g )
        echo $ldapurl
        ldapsearch -x -H $ldapurl > $tmpfile 2>&1
        result=$?
    done
else
    referred=0
fi

# get key
# ldapsearch outputs the key to a random filename in /tmp the grep and awk
if [ $referred -eq 1 ]; then
    certfile=$(ldapsearch -x -H $ldapurl -t userCertificate | grep
^userCertificate | awk -F: '{print $3}')
elif [ $referred -eq 0 ]; then
    certfile=$(ldapsearch -x -b "$mydn" -t userCertificate | grep
^userCertificate | awk -F: '{print $3}')
fi

# convert from DER to PEM
openssl x509 -inform DER -outform PEM -in $certfile -out pubcert.pem
# create challenge output of date, is good enough for now
challenge=$(date)
echo $challenge > challenge
```

```
#encrypt with public key
openssl rsautl -certin -encrypt -inkey pubcert.pem -in challenge -out
challenge.encrypted

# decrypt encrypted challenge with private key
challenge_decrypted=$(openssl rsautl -decrypt -inkey /mnt/usb/.pal/private.pem
-in challenge.encrypted)

if [ "$challenge" = "$challenge_decrypted" ]; then
    echo "Congratulations. You have been successfully authenticated. You may
    proceed"
elif [ "$challenge" != "$challenge_decrypted" ]; then
    echo "Sorry, the keys do not match. I'm calling security."
fi

# remove temp files
rm pubcert.pem
rm $tmpfile
rm challenge.encrypted
rm challenge
rm $certfile
```