OpenSSH Public Key Authentication on Linux

17 February 2016

Version 1.0

DOD PKE Team

Revision History

Issue Date	Revision	Change Description
2/17/2016	1.0	Initial release in reference guide format

Contents

Introduction

The Department of Defense (DoD) Public Key Enablement (PKE) reference guides are developed to help an organization augment their security posture through the use of the DoD Public Key Infrastructure (PKI). The PKE reference guides contain procedures for enabling products and associated technologies to leverage the security services offered by the DoD PKI.

Purpose

The procedures in this document guide the reader in configuring OpenSSH to use public key authentication. OpenSSH configured for public key authentication along with the use of smart cards, such as the DoD Common Access Card (CAC), Alternate Logon Token (ALT), and SIPRNet token, provides a two factor authentication method for Secure Shell (SSH) sessions.

Scope

This document is intended for system administrators who plan to implement two factor authentication for SSH sessions on their Linux systems. Experience installing and configuring software on Linux is helpful when reading this guide. Administrative privileges will be required. The system should have the latest UNIX Security Technical Implementation Guide (STIG) settings applied. In addition the National Institute of Standards and Technology (NIST) has a document NISTIR 7966 titled *Security of Interactive and Automated Access Management using Secure Shell,* http://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.7966.pdf, with information on securing SSH.

The process documented in this guide is based on OpenSSH 5.3.p1-112 on Red Hat Enterprise Linux (RHEL) 6. This guide has been tested with the following operating systems: RHEL 6, RHEL 7, and SUSE Linux 11. Other versions of OpenSSH and other operating systems could possibly require a different procedure. This document discusses Public Key Authentication with SSH protocol version 2.

NOTE: SSH public key authentication does not make use of PKI certificates therefore the typical certificate checks such as revocation checking and certificate expiration checking are not natively done. However it is possible to create a custom script to perform these checks as described in the section Revocation Checking with SSH Public Key Authentication.

Getting Started

Prerequisites

The steps documented in this guide will assume the Linux user accounts and groups have already been created. All commands should be run as the root user or using sudo.

SSH Clients

To implement two factor authentication, clients must be using RSA keys stored on a smart card such as the DoD CAC, ALT, or SIPRNet token.

In addition to completing the procedures mentioned in this guide, an SSH client that can leverage RSA keys stored on smart cards will be required to implement a complete two factor solution. In addition, it is a requirement that the client RSA keys used for authentication are stored on a smart card.

Client systems that will be making a SSH connection should have SSH client software installed that supports reading from smart cards. For client systems running Microsoft Windows, some examples of SSH clients that support reading from smart cards include PuttyCAC, Pragma Fortress SSH Client, and Tectia SSH client. This is not a complete list of SSH clients that support reading from smart cards.

The RSA public key provided to the Linux administrator for configuring access to the Linux server must be a RSA key from the DoD CAC, ALT, or SIPRNet token. If two factor authentication is being implemented, the key cannot be stored in software, such as keys generated using the ssh-keygen command and saved to a file. DoD PKE recommends using the public key that is associated with the certificate used for smart card logon. The certificate used for smart card logon asserts the smart card logon Extended Key Usage (EKU) and is typically the email signature certificate on CACs (or PIV authentication certificate) and the ID certificate on SIPRNet tokens.

Associating Public Keys with User Accounts

This section discusses storing the user's public key and associating the key with their Linux user account. There are several ways this can be done but this document will discuss two methods: storing the user public keys in local files and storing the user public keys in a Lightweight Directory Access Protocol (LDAP) directory.

Storing User's Public Key in Local Files for SSH Access

This section will go over storing the user's public key in a local file on the Linux system. By default the local file is named **authorized_keys** and is stored under each user's home directory. However best practices state users should not have the ability to modify their authorized_keys files, only super users should be able to make changes. DoD PKE recommends creating a central directory (e.g., /etc/ssh/authorized_keys) for storing authorized_keys files and preventing users from being able to write to that directory through file permissions. As an example, the user jdoe might have an authorized_keys file at /etc/ssh/jdoe-authorized_keys. Steps four through six will need to be repeated for each user for whom SSH public key authentication is being setup.

1) On the Linux system, create a central directory for holding the authorized_keys files.

```
$ mkdir /etc/ssh/authorized_keys
```

2) Set the ownership on the new directory.

```
$ chown root:root /etc/ssh/authorized_keys
```

3) Set the permissions on the new directory.

```
$ chmod 755 /etc/ssh/authorized_keys
```

4) Obtain the public key for each user that will be configured for SSH public key authentication. If setting up two factor authentication, ensure the public key for the user is the public key from their smart card that is associated with the correct certificate as discussed in the **SSH Clients** section. The SSH clients mentioned in the **SSH Clients** section can export the public key to a file in the proper SSH format.

   NOTE: A public key is not the same as a certificate. A .cer file on a Windows system is a certificate file and not a public key. The user should export their public key to a file using their SSH client. If exporting the public key is not possible, the user's Base-64 encoded certificate can also be used to extract the public key as described in Appendix A: Creating SSH Formatted Public Key from User Certificate.

5) Create the authorized_keys file for the user using the SSH public key file the user supplied or manually using a text editor such as vi.

```
$ vi /etc/ssh/authorized_keys/<user>-authorized_keys
```

The authorized_keys file should contain the key type, the key itself, and an optional comment. These items should all appear on a single line in the file separated by a space. DoD/NSS PKI uses RSA keys so the key type in the authorized_keys file will be ssh-rsa.

Here is an example authorized_keys file (the key itself has been truncated in this example so it all appears on a single line in the document).

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEArNbrpbene4tLtjtmKvg comment
```

Save the file and exit vi.

6) Check the ownership and permissions on the authorized_keys file created in the previous step. Then change ownership and permissions as needed.

The authorized_keys file should normally be owned by the user and only that user should have read (400) permissions on the file. For example, the user jdoe should own the authorized_keys file at /etc/ssh/authorized_keys/jdoe-authorized_keys and only jdoe should have permissions to read (400) on the file.

```
$ chown <user>:<user> /etc/ssh/authorized_keys/<user>-
authorized_keys
$ chmod 400 /etc/ssh/authorized_keys/<user>-authorized_keys
```

Storing User's Public Key in LDAP Directory for SSH Access

This section will go over storing the user's public key in a LDAP directory. OpenSSH supports having the user's public key stored in a LDAP directory instead of in a local authorized_keys file as discussed in the previous section. This section assumes the LDAP directory is already setup and contains entries for the users.

1) Obtain the public key for each user that will be configured for SSH public key authentication. If setting up two factor authentication, ensure the public key for the user is the public key from their smart card that is associated with the correct certificate as discussed in the **SSH Clients** section. The SSH clients mentioned in the **SSH Clients** section can export the public key to a file in the proper SSH format.

NOTE: A public key is not the same as a certificate. A .cer file on a Windows system is a certificate file and not a public key. The user should export their public key to a file using their SSH client. If exporting the public key is not possible, the user's Base-64 encoded certificate can also be used to extract the

public key as described in Appendix A: Creating SSH Formatted Public Key from User Certificate.

2) The LDAP directory will need to be able to support storing the user's public key for SSH access. The public key is normally stored in an LDAP user attribute titled **sshPublicKey** which is part of the ldapPublicKey object class. Adding the **sshPublicKey** attribute may require extending the LDAP schema which is out of the scope of this reference guide.

   Add the user's public key obtained in step one to their LDAP user entry. A sample user's LDAP entry may look something like this:

```
dn: cn=user1,ou=users,dc=pkecloud,dc=local
uid: user1
gecos: user1
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
objectClass: ldapPublicKey
userPassword: {SSHA}XXXXXX
shadowLastChange: 15140
shadowMin: 0
shadowMax: 99999
shadowWarning: 7
loginShell: /bin/bash
uidNumber: 2000
gidNumber: 2000
homeDirectory: /home/user1
cn: user1
sshPublicKey: ssh-rsa AAAAB3NzaC1yc2EAAAAB comment
```

   In this example the key has been truncated for it to fit on one line in this document. The format of the value that goes in the sshPublicKey attribute is the same as the format for the authorized_keys file when local files are used to store the public keys. The format is key type, the key itself, and an optional comment.

These items should all appear on a single line in the sshPublicKey attribute separated by a space.

This step will need to be repeated for each user that is being setup for SSH public key authentication.

3) The Linux operating system, where OpenSSH is running, may require installing additional packages in order to support keys being stored in a LDAP directory. In the case of Red Hat Enterprise Linux the openssh-ldap package must be installed for this to work. If additional packages are required, install them before proceeding.

   The openssh-ldap package on RHEL provides a utility ssh-ldap-helper along with a wrapper script ssh-ldap-wrapper. This utility handles looking up the user in the LDAP directory and returning the sshPublicKey attribute with the user's public key. Some operating systems may include these required utilities in the OpenSSH package, so no additional packages may be required. If the Linux operating system being used does not supply such a utility, a custom script may need to be developed to handle this. The script would need to accept the username as input, query the LDAP directory, and output the user's SSH public key information to standard output.

4) When using LDAP to store user's public keys, it is important to use secure LDAP or LDAPS when communicating with the LDAP server. In order to set up a secure configuration with LDAPS, the DoD/NSS PKI CA certificates need to be trusted on the OpenSSH server.

   a. Create the folder where the CA certificates will be stored with the command.

      ```
      $ mkdir –p /etc/ssh/cacerts/
      ```

   b. Navigate to the CA certificates folder that was created.

      ```
      $ cd /etc/ssh/cacerts
      ```

   c. Obtain the PKI CA certificates for installation.

      The DoD/NSS PKI CA certificates can be obtained as PKCS#7 files from the DoD PKE Engineering web site at http://iase.disa.mil/pki-pke or http://iase.disa.smil.mil/pki-pke on SIPRNet under **Tools > Trust Store.** Select the appropriate certificate bundle under PKI CA Certificate Bundles: PKCS#7. Save the package locally. Extract the zip file and

navigate into the extracted directory structure. Follow the instructions in the README.txt to validate the signature of the PKCS#7 package.

d. Convert the pem-signed .p7b file(s) to a single PEM file of concatenated certificates using 'openssl' by entering the following command(s) from the directory containing the p7b file(s).

```
$ openssl pkcs7 -inform PEM -outform PEM -in <p7b-file> -out
/etc/ssh/cacerts/<output-file> -print_certs
```

Where <*p7b-file*> is the pem-signed .p7b file (e.g, Certificates_PKCS7_v4.0.1_DoD.pem.p7b) from the PKI CA certificate bundle downloaded in step 4c and <*output-file*> is the name of the file where the output should be saved (e.g., dodcerts.pem). The example openssl command assumes the directory created in step 4a is /etc/ssh/cacerts. If a different directory was used, please adjust the openssl command to account for that.

5) Create a SSH LDAP configuration file using a text editor such as vi. This file will normally be created at /etc/ssh/ldap.conf.

```
$ vi /etc/ssh/ldap.conf
```

Here is sample ldap.conf file, the actual file may be different depending on the Linux distribution and the environment:

```
Base      ou=users,dc=pkecloud,dc=local
URI       ldaps://ldap.pkecloud.local:636
BindDN    cn=admin,dc=pkecloud,dc=local
BindPW    password
TimeLimit 15
TLS_CACertFile /etc/ssh/cacerts/dodcerts.pem
TLS_CheckPeer  demand
#EOF
```

When using the openssh-ldap package on RHEL, the ldap.conf file may contain an optional parameter, **search_format**, that can be used to specify how the LDAP search will be done when finding the user's SSH public key. The default LDAP search is for an entry with objectClass equal to posixAccount, objectClass equal to ldapPublicKey, and uid equal to the user name. The **search_format** parameter can be used in the ldap.conf to specify a different LDAP search.

It is important that LDAPS be used for the URI so the bind password is not passed in the clear over the network. The Linux operating system may supply more information on this ldap.conf file setup by running **man ssh-ldap.conf**.

Configure OpenSSH for Public Key Authentication

This section discusses setting up the OpenSSH server to perform public key authentication and securing the server from a PKE perspective.

OpenSSH Server Key

Many OpenSSH installations will have multiple SSH server keys supporting different algorithms (e.g., DSA, RSA, ECDSA). DoD PKE does not recommend using DSA keys with SSH implementations due to security concerns with DSA. Newer versions of OpenSSH may have DSA keys disabled by default. We recommend using a RSA 2048 bit or higher key for the OpenSSH server. DoD PKE also recommends generating a new SSH server key at least every three years.

1) Find the OpenSSH server key files by checking the /etc/ssh/sshd_config file's HostKey directives. There will often be multiple HostKey directives, which point to multiple key files, in the sshd_config file.

```
$ more /etc/ssh/sshd_config
```

Find the HostKey lines, which will point to the OpenSSH server's key files. They may appear like this:

```
#HostKey for protocol version 1
#HostKey /etc/ssh/ssh_host_key
#HostKeys for protocol version 2
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_dsa_key
```

If the lines are commented out as in the example, then OpenSSH uses the default values.

2) Check each key file found in the previous step to determine the key type (e.g., DSA, RSA, ECDSA) and key size (e.g., 1024, 2048).

```
$ ssh-keygen -lf <key-file>
```

Example Output:

```
2048 a2:73:98:d3:43:21:b7:a5:09 ssh_host_rsa_key.pub (RSA)
```

3) If any DSA keys were found and the system supports it, configure OpenSSH to no longer use the DSA keys.

```
$ rm /etc/ssh/ssh_host_dsa_key /etc/ssh/ssh_host_dsa_key.pub
$ cp -p /etc/ssh/sshd_config /etc/ssh/sshd_config.bak
$ vi /etc/ssh/sshd_config
```

Make sure the only HostKey value specified in the file points to RSA 2048 bit or higher keys. The new line should look similar to this **HostKey /etc/ssh/ssh_host_rsa_key**. Save the file and exit vi.

NOTE: Some newer OpenSSH installations, including those that come with RHEL 7, will have DSA keys disabled by default so no actions will be required.

4) In RHEL 6, OpenSSH can be configured to not generate a new DSA key on restart of sshd. This is done by editing the /etc/sysconfig/sshd file setting **AUTOCREATE_SERVER_KEYS=RSAONLY**, removing the DSA keys from the system, and editing /etc/ssh/sshd_config to specify the RSA HostKey file.

```
$ cp -p /etc/sysconfig/sshd /etc/sysconfig/sshd.bak
$ vi /etc/sysconfig/sshd
```

Change **AUTOCREATE_SERVER_KEYS=YES** to **AUTOCREATE_SERVER_KEYS=RSAONLY**. Save the file and exit vi.

NOTE: This step should be skipped on systems that are not running RHEL 6.

5) If needed, generate a new OpenSSH server key. This step should only be completed when a new server key for OpenSSH is needed.

   a. Rename the two existing key files associated with the server key being replaced.

```
$ mv /etc/ssh/ssh_host_rsa_key /etc/ssh/ssh_host_rsa_key.bak
$ mv /etc/ssh/ssh_host_rsa_key.pub
/etc/ssh/ssh_host_rsa_key.pub.bak
```

   b. Generate the new server key.

```
$ ssh-keygen -f /etc/ssh/ssh_host_rsa_key  -t rsa -b 2048 -N ''
```

   c. Notify all users of new OpenSSH server key so they may update their known_hosts (or similar) files. The command **ssh-keygen –lf <*key-file*>** may be used to find the fingerprint of the new key to supply to users.

6) Restart sshd for the changes to take effect.

```
$ service sshd restart
```

Setting OpenSSH Configuration

These steps document the process of configuring OpenSSH to perform public key authentication.

1) Create a backup of the file /etc/ssh/ssd_config file.

```
$ cp -p /etc/ssh/sshd_config /etc/ssh/sshd_config.bak2
```

2) Open the sshd_config file for editing using a text editor such as vi.

```
$ vi /etc/ssh/sshd_config
```

In the sshd_config file, verify **PubkeyAuthentication** is set to yes. If PubkeyAuthentication is commented out in the file, then it is enabled as by default this setting is enabled. If SSH password authentication is being disabled and only public key authentication will be allowed, the line **PasswordAuthentication** should be set to no.

```
PubkeyAuthentication yes
PasswordAuthentication no     #if disabling password authentication
```

Find the **Ciphers** directive in the sshd_config file and set the value to **aes128-cbc,aes192-cbc,aes256-cbc** if the OpenSSH server and SSH clients in use support these ciphers.

```
Ciphers aes128-cbc,aes192-cbc,aes256-cbc
```

Find the **MACs** directive in the sshd_config file and set the value to **hmac-sha1,hmac-sha2-256,hmac-sha2-512** if the OpenSSH server and SSH clients in use support these MACs.

```
MACs hmac-sha1,hmac-sha2-256,hmac-sha2-512
```

If using local authorized_keys files to store the user's public key, as described in the section **Storing User's Public Key in Local Files for SSH Access**, set **AuthorizedKeysFile** to **/etc/ssh/authorized_keys/%u-authorized_keys**.

```
# If using local authorized_keys files
AuthorizedKeysFile /etc/ssh/authorized_keys/%u-authorized_keys
```

If using LDAP to store the user's public key, as described in the section **Storing User's Public Key in LDAP Directory for SSH Access**, find the line **AuthorizedKeysCommand** and verify the value is set to either /usr/libexec/openssh/ssh-ldap-wrapper, a custom script developed for this, or some other script/utility provided by the Linux operating system being used. Also verify the line **AuthorizedKeysCommandRunAs** is set to user that the script/utility referenced in AuthorizedKeysCommand will run as. If this value is left blank, the script/utility will run as the user attempting to perform the SSH login. In some newer versions of OpenSSH, the AuthorizedKeysCommandRunAs directive may be named AuthorizedKeysCommandUser.

```
# If using LDAP to store user keys
AuthorizedKeysCommand /usr/libexec/openssh/ssh-ldap-wrapper
AuthorizedKeysCommandRunAs nobody
```

NOTE: When using the AuthorizedKeysCommand in sshd_config, if the command fails to return the user's public key it will fail back to checking the local authorized_keys file for that user. This is important to keep in mind when using LDAP to store the user's public key.

Save the changes and exit vi.

NOTE: If setting any of the directives mentioned in this step and a line does not already exist with that directive, create a new line for it. If setting any of the directives and a line already exists but is commented out, remove the '#' to uncomment the line when applicable.

3) Restart sshd for the changes to take effect.

```
$ service sshd restart
```

The OpenSSH server is now configured to support public key authentication.

Revocation Checking with SSH Public Key Authentication

SSH public key authentication uses only key pairs and not PKI certificates, therefore the normal PKI certificate checks, such as revocation checking and expiration checking, are not natively possible. However, it may be possible to create a custom setup to check for revocation and expiration. This would involve saving the user's PKI certificate along with their public key and creating a custom script to perform the PKI checks.

In the case of storing the user's public key in the local authorized_keys files, the user's PKI certificate could possibly also be saved to this file on a commented out line. The authorized_keys file might look something like this:

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEArNbrpbene4tLtjtmKvg comment
#userCertificate MIIFTDCCBDSgAwIBAgICf5YwDQYJKoZIhvcNAQEFBQAwXDELMAkGA
```

The key and certificate blobs have been truncated in this example so they appear as a single line in this document. In an actual authorized_keys file, the entire user PKI certificate would need to be on a single line and that line must be commented out (start with a #).

In the case of storing the user's public key in a LDAP directory, the user's PKI certificate could possibly be stored in a userCertificate attribute in the user's LDAP entry. The LDAP entry might look something like this:

```
dn: cn=user1,ou=users,dc=pkecloud,dc=local
uid: user1
gecos: user1
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
objectClass: ldapPublicKey
userPassword: {SSHA}XXXXXX
shadowLastChange: 15140
shadowMin: 0
shadowMax: 99999
shadowWarning: 7
loginShell: /bin/bash
```

```
uidNumber: 2000
gidNumber: 2000
homeDirectory: /home/user1
cn: user1
sshPublicKey: ssh-rsa AAAAB3NzaC1yc2EAAAAB comment
userCertificate: MIIFTDCCBDSgAwIBAgICf5YwDQYJKoZIhvcNAQEF
```

Once the user's PKI certificates are stored as described, a custom script could be developed to check the PKI certificate for revocation and expiration. This script could be implemented in two ways. First would be as a cron job that runs periodically (once a day) and checks every user's PKI certificate for revocation and expiration. If the user's certificate is expired or revoked, the script could remove the associated public key for that user which would prevent them from being able to use SSH public key authentication. Alternatively, the script could be called when a user attempts SSH public key authentication by calling the script on the AuthorizeKeysCommand line in the sshd_config file. The script would then check the user's certificate. If it is expired or revoked, the script would not return the public key and remove the public key for that user which would deny them access.

It is important to note when using the AuthorizeKeysCommand line in the sshd_config file, if the command fails to return the public key information SSH will fail back to check the local authorized_keys file for the user's public key. Please keep this in mind if developing a custom script.

Appendix A: Creating SSH Formatted Public Key from User Certificate

This appendix will go over how to take a Base-64 encoded user certificate file and create a properly formatted SSH authorized_keys file from it. These steps require using an OpenSSL version of 1.0.1 or higher and an OpenSSH version of 5.6 or higher. The steps assume the Base-64 encoded user certificate file is at /root/user.cer.

1) Use OpenSSL to extract the public key from the user certificate file. Save the public key to /root/user.pub.

```
$ openssl x509 -inform PEM -in /root/user.cer -noout -pubkey >
/root/user.pub
```

2) Convert the user.pub file to the SSH format for the authorized_keys file. Save the output to /root/user.authorized_keys.

```
$ ssh-keygen -i -f /root/user.pub -m pkcs8 >
/root/user.authorized_keys
```

3) The file /root/user.authorized_keys can now be copied into place as a user's authorized_keys file. This example is for user jdoe with the authorized_keys files being stored in /etc/ssh/authorized_keys. The file ownership and permissions should be checked on the new authorized_keys file.

```
$ cp /root/user.authorized_keys /etc/ssh/authorized_keys/jdoe-
authorized_keys
```

Appendix B: Common Issues

This section discusses common problems that can occur with SSH public key authentication.

When SSH public key authentication is not working, check the Linux system logs for possible messages related to the problem. On RHEL systems, SSH issues often get logged to /var/log/messages and /var/log/secure.

Cannot Connect to LDAP Server

A common message that may appear on RHEL systems in /var/log/messages is "Cannot connect to LDAP server" when SSH is configured to lookup user keys in a LDAP directory. There are a number of different issues that can cause this message to appear.

- **Network Issue:** Check to make sure the network is up and allowing access to the LDAP server on the port specified in /etc/ssh/ldap.conf.
- **CA Trust Issue:** If the LDAP server's SSL/TLS certificate is issued by a CA that the OpenSSH server does not trust the "Cannot connect to LDAP server" error may appear. Verify the correct CA certificates are in the certificate file specified in the /etc/ssh/ldap.conf file's TLS_CACertFile directive (e.g., /etc/ssh/cacerts/dodcerts.pem).
- **SELinux Issue:** It is possible SELinux may prevent the ssh-ldap-helper utility from accessing port 636 to make the LDAPS connection. This may result in "Cannot connect to LDAP server" being recorded in the logs. Check the logs for SELinux related messages. Modify SELinux as required.

Utility to Support LDAP SSH Public Key Lookup Does Not Exist

As discussed in the section **Storing User's Public Key in LDAP Directory for SSH Access**, some operating systems will supply a utility to look up the user's SSH public keys in a LDAP directory. On RHEL 6 and 7, this utility is named ssh-ldap-helper and also comes with a wrapper script ssh-ldap-wrapper. RHEL provides these items in a separate package named openssh-ldap. Some operating systems may supply these or similar utilities in their OpenSSH package instead of in a separate package, and some operating systems may not supply these utilities at all. In the case of them not being supplied at all, a custom script or utility would need to be developed to handle this or an existing utility from a different operating system would need to be ported over.

In testing SUSE Linux 11, it was found that SUSE 11.3 contains ssh-ldap-helper and ssh-ldap-wrapper as part of the OpenSSH package but in SUSE 11.4 these files do not appear to be supplied. On SUSE 11.4, the files would need to be obtained elsewhere. It

may be possible to port the files over from SUSE 11.3 or create a custom script to handle this.

If creating a custom script, the script would need to accept the username as input, query the LDAP directory, and output the user's SSH public key information to standard output.

Appendix C: Supplemental Information

Website

Please visit the URL below for additional information
http://iase.disa.mil/pki-pke

Technical Support

Contact technical support
dodpke@mail.mil

Acronyms

ALT	Alternate Logon Token
CA	Certification Authority
CAC	Common Access Card
EKU	Extended Key Usage
LDAP	Lightweight Directory Access Protocol
NIST	National Institute of Standards and Technology
NSS	National Security System
PKE	Public Key Enablement
PKI	Public Key Infrastructure
RHEL	Red Hat Enterprise Linux
SCL	Smart Card Logon
SSH	Secure Shell
STIG	Security Technical Implementation Guide
URI	Uniform Resource Identifier