Managing smart card authentication

Configuring and managing smart card authentication in RHEL
Configuring and managing smart card authentication in RHEL
Abstract

This documentation collection provides instructions on how to manage smart card authentication in RHEL.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>5.2. INSTALLING TOOLS FOR MANAGING AND USING SMART CARDS</td>
<td>41</td>
</tr>
<tr>
<td>5.3</td>
<td>5.3. PREPARING YOUR SMART CARD AND UPLOADING YOUR CERTIFICATES AND KEYS TO YOUR SMART CARD</td>
<td>42</td>
</tr>
<tr>
<td>5.4</td>
<td>5.4. ENABLING SMART CARD AUTHENTICATION FOR THE WEB CONSOLE</td>
<td>44</td>
</tr>
<tr>
<td>5.5</td>
<td>5.5. LOGGING IN TO THE WEB CONSOLE WITH SMART CARDS</td>
<td>44</td>
</tr>
<tr>
<td>5.6</td>
<td>5.6. ENABLING PASSWORD-LESS SUDO FOR SMART CARD USERS</td>
<td>45</td>
</tr>
<tr>
<td>5.7</td>
<td>5.7. LIMITING USER SESSIONS AND MEMORY TO PREVENT A DOS ATTACK</td>
<td>47</td>
</tr>
<tr>
<td>6.1</td>
<td>6.1. CREATING LOCAL CERTIFICATES</td>
<td>48</td>
</tr>
<tr>
<td>6.2</td>
<td>6.2. COPYING CERTIFICATES TO THE SSSD DIRECTORY</td>
<td>51</td>
</tr>
<tr>
<td>6.3</td>
<td>6.3. INSTALLING TOOLS FOR MANAGING AND USING SMART CARDS</td>
<td>52</td>
</tr>
<tr>
<td>6.4</td>
<td>6.4. PREPARING YOUR SMART CARD AND UPLOADING YOUR CERTIFICATES AND KEYS TO YOUR SMART CARD</td>
<td>52</td>
</tr>
<tr>
<td>6.5</td>
<td>6.5. CONFIGURING SSH ACCESS USING SMART CARD AUTHENTICATION</td>
<td>54</td>
</tr>
<tr>
<td>7.1</td>
<td>7.1. CERTIFICATES ELIGIBLE FOR SMART CARDS</td>
<td>57</td>
</tr>
<tr>
<td>7.2</td>
<td>7.2. CONFIGURE YOUR SYSTEM TO ENABLE BOTH SMART CARD AND PASSWORD AUTHENTICATION</td>
<td>57</td>
</tr>
<tr>
<td>7.3</td>
<td>7.3. CONFIGURING YOUR SYSTEM TO ENFORCE SMART CARD AUTHENTICATION</td>
<td>58</td>
</tr>
<tr>
<td>7.4</td>
<td>7.4. CONFIGURING SMART CARD AUTHENTICATION WITH LOCK ON REMOVAL</td>
<td>58</td>
</tr>
<tr>
<td>8.1</td>
<td>8.1. CREATING SUDO RULES IN IDM</td>
<td>60</td>
</tr>
<tr>
<td>8.2</td>
<td>8.2. SETTING UP THE PAM MODULE FOR SUDO</td>
<td>61</td>
</tr>
<tr>
<td>8.3</td>
<td>8.3. CONNECTING TO SUDO REMOTELY USING A SMART CARD</td>
<td>61</td>
</tr>
<tr>
<td>9.1</td>
<td>9.1. TESTING SMART CARD ACCESS ON THE SYSTEM</td>
<td>63</td>
</tr>
<tr>
<td>9.2</td>
<td>9.2. TROUBLESHOOTING SMART CARD AUTHENTICATION WITH SSSD</td>
<td>66</td>
</tr>
<tr>
<td>9.3</td>
<td>9.3. VERIFYING THAT IDM KERBEROS KDC CAN USE PKINIT AND THAT THE CA CERTIFICATES ARE CORRECTLY LOCATED</td>
<td>68</td>
</tr>
<tr>
<td>9.4</td>
<td>9.4. INCREASING SSSD TIMEOUTS</td>
<td>70</td>
</tr>
<tr>
<td>9.5</td>
<td>9.5. TROUBLESHOOTING CERTIFICATE MAPPING AND MATCHING RULES</td>
<td>71</td>
</tr>
<tr>
<td>9.5.1</td>
<td>9.5.1. Checking how the certificates are mapped to users</td>
<td>71</td>
</tr>
<tr>
<td>9.5.2</td>
<td>9.5.2. Checking the user associated with a smart card certificate</td>
<td>73</td>
</tr>
</tbody>
</table>
MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.

In Identity Management, planned terminology replacements include:

- **block list** replaces **blacklist**
- **allow list** replaces **whitelist**
- **secondary** replaces **slave**
- The word **master** is being replaced with more precise language, depending on the context:
  - **IdM server** replaces **IdM master**
  - **CA renewal server** replaces **CA renewal master**
  - **CRL publisher server** replaces **CRL master**
  - **multi-supplier** replaces **multi-master**
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting comments on specific passages

1. View the documentation in the Multi-page HTML format and ensure that you see the Feedback button in the upper right corner after the page fully loads.

2. Use your cursor to highlight the part of the text that you want to comment on.

3. Click the Add Feedback button that appears near the highlighted text.

4. Add your feedback and click Submit.

Submitting feedback through Bugzilla (account required)

1. Log in to the Bugzilla website.

2. Select the correct version from the Version menu.

3. Enter a descriptive title in the Summary field.

4. Enter your suggestion for improvement in the Description field. Include links to the relevant parts of the documentation.

5. Click Submit Bug.
CHAPTER 1. UNDERSTANDING SMART CARD AUTHENTICATION

Authentication based on smart cards is an alternative to passwords. You can store user credentials on a smart card in the form of a private key and a certificate, and special software and hardware is used to access them. Place the smart card into a reader or a USB port and supply the PIN code for the smart card instead of providing your password.

This section describes what a smart card is and how smart card authentication works. It describes the tools that you can use to read and manipulate smart card content. It also provides sample use cases and describes the setup of both the IdM server and IdM client for smart card authentication.

NOTE
If you want to start to use smart card authentication, see the hardware requirements: Smart Card support in RHEL9.

1.1. WHAT IS A SMART CARD

A smart card is a physical device, usually a plastic card with a microprocessor, that can provide personal authentication using certificates stored on the card. Personal authentication means that you can use smart cards in the same way as user passwords.

You can store user credentials on the smart card in the form of a private key and a certificate, and special software and hardware is used to access them. You place the smart card into a reader or a USB socket and supply the PIN code for the smart card instead of providing your password.

1.2. WHAT IS SMART CARD AUTHENTICATION

Public-key based authentication and certificate based authentication are two widely used alternatives to password based authentication. Your identity is confirmed by using public and private keys instead of your password. A certificate is an electronic document used to identify an individual, a server, a company, or other entity and to associate that identity with a public key. Like a driver’s license or passport, a certificate provides generally recognized proof of a person’s identity. Public-key cryptography uses certificates to address the problem of impersonation.

In the case of smart card authentication, your user credentials, that is your public and private keys and certificate, are stored on a smart card and can only be used after the smart card is inserted into the reader and a PIN is provided. As you need to possess a physical device, the smart card, and know its PIN, smart card authentication is considered as a type of two factor authentication.

1.2.1. Examples of smart card authentication in IdM

The following examples describe two simple scenarios on how you can use smart cards in IdM.

1.2.1.1. Logging in to your system with a smart card

You can use a smart card to authenticate to a RHEL system as a local user. If your system is configured to enforce smart card login, you are prompted to insert your smart card and enter its PIN and, if that fails, you cannot log in to your system. Alternatively, you can configure your system to authenticate using either smart card authentication or your user name and password. In this case, if you don’t have your smart card inserted, you are prompted for your user name and password.
1.2.1.2. Logging in to GDM with lock on removal

You can activate the lock on removal function if you have configured smart card authentication on your RHEL system. If you are logged in to the GNOME Display Manager (GDM) and you remove your smart card, screen lock is enabled and you must reinsert your smart card and authenticate with the PIN to unlock the screen. You cannot use your user name and password to authenticate.

**NOTE**

If you are logged in to GDM and you remove your smart card, screen lock is enabled and you must reinsert your smart card and authenticate with the PIN to unlock the screen.

1.3. SMART CARD AUTHENTICATION OPTIONS IN RHEL

You can configure how you want smart card authentication to work in a particular Identity Management (IdM) client by using the `authselect` command, `authselect enable-feature <smartcardoption>`. The following smart card options are available:

- **with-smartcard**: Users can authenticate with the user name and password or with their smart card.

- **with-smartcard-required**: Users can authenticate with their smart cards, and password authentication is disabled. You cannot access the system without your smart card. Once you have authenticated with your smart card, you can stay logged in even if your smart card is removed from its reader.

  **NOTE**

  The `with-smartcard-required` option only enforces exclusive smart card authentication for login services, such as `login`, `gdm`, `xdm`, `xscreensaver`, and `gnome-screensaver`. For other services, such as `su` or `sudo` for switching users, smart card authentication is not enforced and if your smart card is not inserted, you are prompted for a password.

- **with-smartcard-lock-on-removal**: Users can authenticate with their smart card. However, if you remove your smart card from its reader, you are automatically locked out of the system. You cannot use password authentication.

  **NOTE**

  The `with-smartcard-lock-on-removal` option only works on systems with the GNOME desktop environment. If you are using a system that is `tty` or console based and you remove your smart card from its reader, you are not automatically locked out of the system.

For more information, see Configuring smart cards using authselect.

1.4. TOOLS FOR MANAGING SMART CARDS AND THEIR CONTENTS

You can use many different tools to manage the keys and certificates stored on your smart cards. You can use these tools to do the following:

- List available smart card readers connected to a system.
List available smart cards and view their contents.

Manipulate the smart card content, that is the keys and certificates.

There are many tools that provide similar functionality but some work at different layers of your system. Smart cards are managed on multiple layers by multiple components. On the lower level, the operating system communicates with the smart card reader using the PC/SC protocol, and this communication is handled by the pcsclite daemon. The daemon forwards the commands received to the smart card reader typically over USB, which is handled by low-level CCID driver. The PC/SC low level communication is rarely seen on the application level. The main method in RHEL for applications to access smart cards is via a higher level application programming interface (API), the OASIS PKCS#11 API, which abstracts the card communication to specific commands that operate on cryptographic objects, for example, private keys. Smart card vendors provide a shared module, such as an .so file, which follows the PKCS#11 API and serves as a driver for the smart card.

You can use the following tools to manage your smart cards and their contents:

- OpenSC tools: work with the drivers implemented in opensc.
  - opensc-tool: perform smart card operations.
  - pkcs15-tool: manage the PKCS#15 data structures on smart cards, such as listing and reading PINs, keys, and certificates stored on the token.
  - pkcs11-tool: manage the PKCS#11 data objects on smart cards, such as listing and reading PINs, keys, and certificates stored on the token.

- GnuTLS utils: an API for applications to enable secure communication over the network transport layer, as well as interfaces to access X.509, PKCS#12, OpenPGP, and other structures.
  - p11tool: perform operations on PKCS#11 smart cards and security modules.
  - certtool: parse and generate X.509 certificates, requests, and private keys.

- Network Security Services (NSS) Tools: a set of libraries designed to support the cross-platform development of security-enabled client and server applications. Applications built with NSS can support SSL v3, TLS, PKCS #5, PKCS #7, PKCS #11, PKCS #12, S/MIME, X.509 v3 certificates, and other security standards.
  - modutil: manage PKCS#11 module information with the security module database.
  - certutil: manage keys and certificates in both NSS databases and other NSS tokens.

For more information on using these tools to troubleshoot issues with authenticating using a smart card, see Troubleshooting authentication with smart cards.

Additional resources

- opensc-tool man page
- pkcs15-tool man page
- pkcs11-tool man page
- p11tool man page
- certtool man page
1.5. CERTIFICATES AND SMART CARD AUTHENTICATION

If you use Identity Management (IdM) or Active Directory (AD) to manage identity stores, authentication, policies, and authorization policies in your domain, the certificates used for authentication are generated by IdM or AD, respectively. You can also use certificates provided by an external certificate authority and in this case you must configure Active Directory or IdM to accept certificates from the external provider. If the user is not part of a domain, you can use a certificate generated by a local certificate authority. For details, refer to the following sections:

- Configuring Identity Management for smart card authentication
- Configuring certificates issued by ADCS for smart card authentication in IdM
- Managing externally signed certificates for IdM users, hosts, and services
- Configuring and importing local certificates to a smart card

For a full list of certificates eligible for smart card authentication, see Certificates eligible for smart cards.

1.6. REQUIRED STEPS FOR SMART CARD AUTHENTICATION IN IDM

You must ensure the following steps have been followed before you can authenticate with a smart card in Identity Management (IdM):

- Configure your IdM server for smart card authentication. See Configuring the IdM server for smart card authentication
- Configure your IdM client for smart card authentication. See Configuring the IdM client for smart card authentication
- Add the certificate to the user entry in IdM. See Adding a certificate to a user entry in the IdM Web UI
- Store your keys and certificates on the smart card. See Storing a certificate on a smart card

1.7. REQUIRED STEPS FOR SMART CARD AUTHENTICATION WITH CERTIFICATES ISSUED BY ACTIVE DIRECTORY

You must ensure the following steps have been followed before you can authenticate with a smart card with certificates issued by Active Directory (AD):

- Copy the CA and user certificates from Active Directory to the IdM server and client.
- Configure the IdM server and clients for smart card authentication using ADCS certificates.
- Convert the PFX (PKCS#12) file to be able to store the certificate and private key on the smart card.
- Configure timeouts in the sssd.conf file.
- Create certificate mapping rules for smart card authentication.
CHAPTER 2. CONFIGURING IDENTITY MANAGEMENT FOR SMART CARD AUTHENTICATION

Identity Management (IdM) supports smart card authentication with:

- User certificates issued by the IdM certificate authority
- User certificates issued by an external certificate authority

This user story shows how to configure smart card authentication in IdM for both types of certificates. In the user story, the rootca.pem CA certificate is the file containing the certificate of a trusted external certificate authority.

For information on smart card authentication in IdM, see Understanding smart card authentication.

The user story contains the following modules:

- Configuring the IdM server for smart card authentication
- Configuring the IdM client for smart card authentication
- Adding a certificate to a user entry in the IdM Web UI
- Adding a certificate to a user entry in the IdM CLI
- Installing tools for managing and using smart cards
- Storing a certificate on a smart card
- Logging in to IdM with smart cards
- Configuring GDM access using smart card authentication
- Configuring su access using smart card authentication

2.1. CONFIGURING THE IDM SERVER FOR SMART CARD AUTHENTICATION

If you want to enable smart card authentication for users whose certificates have been issued by the certificate authority (CA) of the <EXAMPLE.ORG> domain that your Identity Management (IdM) CA trusts, you must obtain the following certificates so that you can add them when running the ipa-advise script that configures the IdM server:

- The certificate of the root CA that has either issued the certificate for the <EXAMPLE.ORG> CA directly, or through one or more of its sub-CAs. You can download the certificate chain from a web page whose certificate has been issued by the authority. For details, see Steps 1 - 4a in Configuring a browser to enable certificate authentication.

- The IdM CA certificate. You can obtain the CA certificate from the /etc/ipa/ca.crt file on the IdM server on which an IdM CA instance is running.

- The certificates of all of the intermediate CAs; that is, intermediate between the <EXAMPLE.ORG> CA and the IdM CA.

To configure an IdM server for smart card authentication:
1. Obtain files with the CA certificates in the PEM format.

2. Run the built-in `ipa-advice` script.

3. Reload the system configuration.

**Prerequisites**

- You have root access to the IdM server.
- You have the root CA certificate and all the intermediate CA certificates.

**Procedure**

1. Create a directory in which you will do the configuration:

   ```
   [root@server#] mkdir ~/SmartCard/
   ```

2. Navigate to the directory:

   ```
   [root@server#] cd ~/SmartCard/
   ```

3. Obtain the relevant CA certificates stored in files in PEM format. If your CA certificate is stored in a file of a different format, such as DER, convert it to PEM format. The IdM Certificate Authority certificate is in PEM format and is located in the `/etc/ipa/ca.crt` file. Convert a DER file to a PEM file:

   ```
   # openssl x509 -in <filename>.der -inform DER -out <filename>.pem -outform PEM
   ```

4. For convenience, copy the certificates to the directory in which you want to do the configuration:

   ```
   [root@server SmartCard#] cp /tmp/rootca.pem ~/SmartCard/
   [root@server SmartCard#] cp /tmp/subca.pem ~/SmartCard/
   [root@server SmartCard#] cp /tmp/issuingca.pem ~/SmartCard/
   ```

5. Optionally, if you use certificates of external certificate authorities, use the `openssl x509` utility to view the contents of the files in the PEM format to check that the **Issuer** and **Subject** values are correct:

   ```
   [root@server SmartCard#] openssl x509 -noout -text -in rootca.pem | more
   ```

6. Generate a configuration script with the in-built `ipa-advice` utility, using the administrator’s privileges:

   ```
   [root@server SmartCard#] kinit admin
   [root@server SmartCard#] ipa-advice config-server-for-smart-card-auth > config-server-for-smart-card-auth.sh
   ```

   The *config-server-for-smart-card-auth.sh* script performs the following actions:

   - It configures the IdM Apache HTTP Server.
- It enables Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) on the Key Distribution Center (KDC).

- It configures the IdM Web UI to accept smart card authorization requests.

7. Execute the script, adding the PEM files containing the root CA and sub CA certificates as arguments:

```
[root@server SmartCard]# chmod +x config-server-for-smart-card-auth.sh
[root@server SmartCard]# ./config-server-for-smart-card-auth.sh rootca.pem subca.pem issuingca.pem
Ticket cache:KEYRING:persistent:0:0
Default principal: admin@IDM.EXAMPLE.COM
[...]
Systemwide CA database updated.
The ipa-certupdate command was successful
```

**NOTE**

Ensure that you add the root CA's certificate as an argument before any sub CA certificates and that the CA or sub CA certificates have not expired.

8. Optionally, if the certificate authority that issued the user certificate does not provide any Online Certificate Status Protocol (OCSP) responder, you may need to disable OCSP check for authentication to the IdM Web UI:

a. Set the `SSLOCSPEnable` parameter to `off` in the `/etc/httpd/conf.d/ssl.conf` file:

```
SSLOCSPEnable off
```

b. Restart the Apache daemon (httpd) for the changes to take effect immediately:

```
[root@server SmartCard]# systemctl restart httpd
```

**WARNING**

Do not disable the OCSP check if you only use user certificates issued by the IdM CA. OCSP responders are part of IdM.

For instructions on how to keep the OCSP check enabled, and yet prevent a user certificate from being rejected by the IdM server if it does not contain the information about the location at which the CA that issued the user certificate listens for OCSP service requests, see the `SSLOCSPDefaultResponder` directive in Apache mod_ssl configuration options.

The server is now configured for smart card authentication.
2.2. USING ANSIBLE TO CONFIGURE THE IDM SERVER FOR SMART CARD AUTHENTICATION

You can use Ansible to enable smart card authentication for users whose certificates have been issued by the certificate authority (CA) of the <EXAMPLE.ORG> domain that your Identity Management (IdM) CA trusts. To do that, you must obtain the following certificates so that you can use them when running an Ansible playbook with the `ipasmartcard_server ansible-freeipa` role script:

- The certificate of the root CA that has either issued the certificate for the <EXAMPLE.ORG> CA directly, or through one or more of its sub-CAs. You can download the certificate chain from a web page whose certificate has been issued by the authority. For details, see Step 4 in Configuring a browser to enable certificate authentication.

- The IdM CA certificate. You can obtain the CA certificate from the `/etc/ipa/ca.crt` file on any IdM CA server.

- The certificates of all of the CAs that are intermediate between the <EXAMPLE.ORG> CA and the IdM CA.

Prerequisites

- You have root access to the IdM server.
- You know the IdM admin password.
- You have the root CA certificate, the IdM CA certificate, and all the intermediate CA certificates.
- You have configured an Ansible control node that meets the following requirements:
  - You are using Ansible version 2.8 or later.
  - You have installed the `ansible-freeipa` package.
  - In the `~/MyPlaybooks/` directory, you have created an Ansible inventory file with the fully-qualified domain name (FQDN) of the IdM server where you are configuring the smart card access.

Procedure

1. If your CA certificates are stored in files of a different format, such as DER, convert them to PEM format:

   ```bash
   # openssl x509 -in <filename>.der -inform DER -out <filename>.pem -outform PEM
   ```

   The IdM Certificate Authority certificate is in PEM format and is located in the `/etc/ipa/ca.crt` file.

2. Optionally, use the `openssl x509` utility to view the contents of the files in the PEM format to check that the Issuer and Subject values are correct:
3. Navigate to your `~/MyPlaybooks` directory:

   ```shell
   $ cd ~/MyPlaybooks/
   ```

4. Create a subdirectory dedicated to the CA certificates:

   ```shell
   $ mkdir SmartCard/
   ```

5. For convenience, copy all the required certificates to the `~/MyPlaybooks/SmartCard/` directory:

   ```shell
   # cp /tmp/root-ca.pem ~/MyPlaybooks/SmartCard/
   # cp /tmp/intermediate-ca.pem ~/MyPlaybooks/SmartCard/
   # cp /etc/ipa/ca.crt ~/MyPlaybooks/SmartCard/ipa-ca.crt
   ```

6. In your Ansible inventory file, specify the following:
   - The IdM servers that you want to configure for smart card authentication.
   - The IdM administrator password.
   - The paths to the certificates of the CAs in the following order:
     - The root CA certificate file
     - The intermediate CA certificates files
     - The IdM CA certificate file

   The file can look as follows:

   ```yaml
   [ipaserver]
   ipaserver.idm.example.com

   [ipareplicas]
   ipareplica1.idm.example.com
   ipareplica2.idm.example.com

   [ipacluster:children]
   ipaserver
   ipareplicas

   [ipacluster:vars]
   ipadmin_password=SomeADMINpassword
   ipasmartcard_server_ca_certs=/home/<user_name>/MyPlaybooks/SmartCard/root-ca.pem,/home/<user_name>/MyPlaybooks/SmartCard/intermediate-ca.pem,/home/<user_name>/MyPlaybooks/SmartCard/ipa-ca.crt
   ```

7. Create an `install-smartcard-server.yml` playbook with the following content:

   ```yaml
   ---
   - name: Playbook to set up smart card authentication for an IdM server
     hosts: ipaserver
   ```
become: true

roles:
- role: ipasmartcard_server
  state: present

8. Save the file.

9. Run the Ansible playbook specifying the playbook file and the inventory file:

```
$ ansible-playbook -v -i inventory install-smartcard-server.yml
```

The `ipasmartcard_server` Ansible role performs the following actions:

- It configures the IdM Apache HTTP Server.
- It enables Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) on the Key Distribution Center (KDC).
- It configures the IdM Web UI to accept smart card authorization requests.

10. Optionally, if the certificate authority that issued the user certificate does not provide any Online Certificate Status Protocol (OCSP) responder, you may need to disable OCSP check for authentication to the IdM Web UI:

   a. Connect to the IdM server as `root`:

      ```
      ssh root@ipaserver.idm.example.com
      ```

   b. Set the `SSLOCSPEnable` parameter to `off` in the `/etc/httpd/conf.d/ssl.conf` file:

      ```
      SSLOCSPEnable off
      ```

   c. Restart the Apache daemon (httpd) for the changes to take effect immediately:

      ```
      # systemctl restart httpd
      ```

**WARNING**

Do not disable the OCSP check if you only use user certificates issued by the IdM CA. OCSP responders are part of IdM.

For instructions on how to keep the OCSP check enabled, and yet prevent a user certificate from being rejected by the IdM server if it does not contain the information about the location at which the CA that issued the user certificate listens for OCSP service requests, see the `SSLOCSPDefaultResponder` directive in Apache `mod_ssl` configuration options.

The server listed in the inventory file is now configured for smart card authentication.
NOTE

To enable smart card authentication in the whole topology, set the `hosts` variable in the Ansible playbook to `ipacluster`:

```yaml
- name: Playbook to setup smartcard for IPA server and replicas
  hosts: ipacluster
  [...]
```

Additional resources

- Sample playbooks using the `ipasmartcard_server` role in the `/usr/share/doc/ansible-freeipa/playbooks/` directory

### 2.3. CONFIGURING THE IDM CLIENT FOR SMART CARD AUTHENTICATION

This section describes how to configure IdM clients for smart card authentication. The procedure needs to be run on each IdM system, a client or a server, to which you want to connect while using a smart card for authentication. For example, to enable an `ssh` connection from host A to host B, the script needs to be run on host B.

As an administrator, run this procedure to enable smart card authentication using

- The `ssh` protocol
  For details see Configuring SSH access using smart card authentication.
- The console login
- The Gnome Display Manager (GDM)
- The `su` command

This procedure is not required for authenticating to the IdM Web UI. Authenticating to the IdM Web UI involves two hosts, neither of which needs to be an IdM client:

- The machine on which the browser is running. The machine can be outside of the IdM domain.
- The IdM server on which `httpd` is running.

The following procedure assumes that you are configuring smart card authentication on an IdM client, not an IdM server. For this reason you need two computers: an IdM server to generate the configuration script, and the IdM client on which to run the script.

**Prerequisites**

- Your IdM server has been configured for smart card authentication, as described in Configuring the IdM server for smart card authentication.
- You have root access to the IdM server and the IdM client.
- You have the root CA certificate and all the intermediate CA certificates.
You installed the IdM client with the `--mkhomedir` option to ensure remote users can log in successfully. If you do not create a home directory, the default login location is the root of the directory structure, `/.

**Procedure**

1. On an IdM server, generate a configuration script with `ipa-advise` using the administrator’s privileges:

```
[root@server SmartCard]# kinit admin
[root@server SmartCard]# ipa-advise config-client-for-smart-card-auth > config-client-for-smart-card-auth.sh
```

The `config-client-for-smart-card-auth.sh` script performs the following actions:

- It configures the smart card daemon.
- It sets the system-wide trust store.
- It configures the System Security Services Daemon (SSSD) to allow users to authenticate with either their user name and password or with their smart card. For more details on SSSD profile options for smart card authentication, see [Smart card authentication options in RHEL](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/managing_smart_card_authentication/

2. From the IdM server, copy the script to a directory of your choice on the IdM client machine:

```
[root@server SmartCard]# scp config-client-for-smart-card-auth.sh root@client.idm.example.com:/root/SmartCard/
Password: 
```

3. From the IdM server, copy the CA certificate files in PEM format for convenience to the same directory on the IdM client machine as used in the previous step:

```
[root@server SmartCard]# scp {rootca.pem,subca.pem,issuingca.pem} root@client.idm.example.com:/root/SmartCard/
Password: 
```

4. On the client machine, execute the script, adding the PEM files containing the CA certificates as arguments:

```
[root@client SmartCard]# kinit admin
[root@client SmartCard]# chmod +x config-client-for-smart-card-auth.sh
[root@client SmartCard]#/config-client-for-smart-card-auth.sh rootca.pem subca.pem issuingca.pem
Ticket cache:KEYRING:persistent:0:0
Default principal: admin@IDM.EXAMPLE.COM
[...] 
Systemwide CA database updated.
The ipa-certupdate command was successful
```
NOTE

Ensure that you add the root CA’s certificate as an argument before any sub CA certificates and that the CA or sub CA certificates have not expired.

The client is now configured for smart card authentication.

2.4. USING ANSIBLE TO CONFIGURE IDM CLIENTS FOR SMART CARD AUTHENTICATION

This section describes how to use the `ansible-freeipa ipasmarcard_client` module to configure specific Identity Management (IdM) clients to permit IdM users to authenticate with a smart card. Run this procedure to enable smart card authentication for IdM users that use any of the following to access IdM:

- The `ssh` protocol
  For details see Configuring SSH access using smart card authentication.
- The console login
- The Gnome Display Manager (GDM)
- The `su` command

NOTE

This procedure is not required for authenticating to the IdM Web UI. Authenticating to the IdM Web UI involves two hosts, neither of which needs to be an IdM client:

- The machine on which the browser is running. The machine can be outside of the IdM domain.
- The IdM server on which `httpd` is running.

Prerequisites

- Your IdM server has been configured for smart card authentication, as described in Using Ansible to configure the IdM server for smart card authentication.
- You have root access to the IdM server and the IdM client.
- You have the root CA certificate, the IdM CA certificate, and all the intermediate CA certificates.
- You have configured an Ansible control node that meets the following requirements:
  - You are using Ansible version 2.8 or later.
  - You have installed the `ansible-freeipa` package.
  - In the `~/MyPlaybooks/` directory, you have created an Ansible inventory file with the fully-qualified domain name (FQDN) of the IdM server where you are configuring the smart card access.

Procedure
1. If your CA certificates are stored in files of a different format, such as DER, convert them to PEM format:

   ```
   # openssl x509 -in <filename>.der -inform DER -out <filename>.pem -outform PEM
   
   The IdM CA certificate is in PEM format and is located in the /etc/ipa/ca.crt file.
   ```

2. Optionally, use the `openssl x509` utility to view the contents of the files in the PEM format to check that the Issuer and Subject values are correct:

   ```
   # openssl x509 -noout -text -in root-ca.pem | more
   
   # openssl x509 -noout -text -in intermediate-ca.pem | more
   
   # openssl x509 -noout -text -in ipa-ca.crt | more
   ```

3. On your Ansible control node, navigate to your `~/MyPlaybooks/` directory:

   ```
   $ cd ~/MyPlaybooks/
   ```

4. Create a subdirectory dedicated to the CA certificates:

   ```
   $ mkdir SmartCard/
   ```

5. For convenience, copy all the required certificates to the `~/MyPlaybooks/SmartCard/` directory, for example:

   ```
   # cp /tmp/root-ca.pem ~/MyPlaybooks/SmartCard/
   # cp /tmp/intermediate-ca.pem ~/MyPlaybooks/SmartCard/
   # cp /etc/ipa/ca.crt ~/MyPlaybooks/SmartCard/ipa-ca.crt
   ```

6. In your Ansible inventory file, specify the following:

   - The IdM clients that you want to configure for smart card authentication.
   - The IdM administrator password.
   - The paths to the certificates of the CAs in the following order:
     - The root CA certificate file
     - The intermediate CA certificates files
     - The IdM CA certificate file

   The file can look as follows:

   ```
   [ipaclients]
   ipaclient1.example.com
   ipaclient2.example.com
   
   [ipaclients:vars]
   ipadmin_password=SomeADMINpassword
   ipasmartcard_client_ca_certs=/home/<user_name>/MyPlaybooks/SmartCard/root-ca.pem,/home/<user_name>/MyPlaybooks/SmartCard/intermediate-ca.pem,/home/<user_name>/MyPlaybooks/SmartCard/ipa-ca.crt
   ```

7. Create an `install-smartcard-clients.yml` playbook with the following content:
---
- name: Playbook to set up smart card authentication for an IdM client
  hosts: ipaclients
  become: true
  roles:
  - role: ipasmartcard_client
    state: present

8. Save the file.

9. Run the Ansible playbook. Specify the playbook and inventory files:

```
$ ansible-playbook -v -i inventory install-smartcard-clients.yml
```

The `ipasmartcard_client` Ansible role performs the following actions:

- It configures the smart card daemon.
- It sets the system-wide trust store.
- It configures the System Security Services Daemon (SSSD) to allow users to authenticate with either their user name and password or their smart card. For more details on SSSD profile options for smart card authentication, see [Smart card authentication options in RHEL](#).

The clients listed in the `ipaclients` section of the inventory file are now configured for smart card authentication.

**NOTE**

If you have installed the IdM clients with the `--mkhomedir` option, remote users will be able to log in to their home directories. Otherwise, the default login location is the root of the directory structure, `/`.

Additional resources

- Sample playbooks using the `ipasmartcard_server` role in the `/usr/share/doc/ansible-freeipa/playbooks/` directory

**2.5. ADDING A CERTIFICATE TO A USER ENTRY IN THE IDM WEB UI**

This procedure describes how to add an external certificate to a user entry in IdM Web UI.

**NOTE**

Instead of uploading the whole certificate, it is also possible to upload certificate mapping data to a user entry in IdM. User entries containing either full certificates or certificate mapping data can be used in conjunction with corresponding certificate mapping rules to facilitate the configuration of smart card authentication for system administrators. For details, see [Certificate mapping rules for configuring authentication on smart cards](#).
NOTE

If the user’s certificate has been issued by the IdM Certificate Authority, the certificate is already stored in the user entry, and you can skip this section.

Prerequisites

- You have the certificate that you want to add to the user entry at your disposal.

Procedure

1. Log into the IdM Web UI as an administrator if you want to add a certificate to another user. For adding a certificate to your own profile, you do not need the administrator’s credentials.

2. Navigate to Users → Active users → sc_user.

3. Find the Certificate option and click Add.

4. In the Command-Line Interface, display the certificate in the PEM format using the cat utility or a text editor:

   [user@client SmartCard]$ cat testuser.crt

5. Copy and paste the certificate from the CLI into the window that has opened in the Web UI.

6. Click Add.

Figure 2.1. Adding a new certificate in the IdM Web UI

Red Hat Enterprise Linux 9 Managing smart card authentication
The sc_user entry now contains an external certificate.

### 2.6. ADDING A CERTIFICATE TO A USER ENTRY IN THE IDM CLI

This procedure describes how to add an external certificate to a user entry in IdM CLI.

**NOTE**

Instead of uploading the whole certificate, it is also possible to upload certificate mapping data to a user entry in IdM. User entries containing either full certificates or certificate mapping data can be used in conjunction with corresponding certificate mapping rules to facilitate the configuration of smart card authentication for system administrators. For details, see Certificate mapping rules for configuring authentication on smart cards.

**NOTE**

If the user’s certificate has been issued by the IdM Certificate Authority, the certificate is already stored in the user entry, and you can skip this section.

**Prerequisites**

- You have the certificate that you want to add to the user entry at your disposal.

**Procedure**

1. Log into the IdM CLI as an administrator if you want to add a certificate to another user:

   [user@client SmartCard]$ kinit admin

   For adding a certificate to your own profile, you do not need the administrator’s credentials:

   [user@client SmartCard]$ kinit sc_user

2. Create an environment variable containing the certificate with the header and footer removed and concatenated into a single line, which is the format expected by the `ipa user-add-cert` command:

   [user@client SmartCard]$ export CERT=`openssl x509 -outform der -in testuser.crt | base64 -w0`

   Note that certificate in the `testuser.crt` file must be in the PEM format.

3. Add the certificate to the profile of sc_user using the `ipa user-add-cert` command:

   [user@client SmartCard]$ ipa user-add-cert sc_user --certificate=$CERT

The sc_user entry now contains an external certificate.

### 2.7. INSTALLING TOOLS FOR MANAGING AND USING SMART CARDS

To configure your smart card, you need tools which can generate certificates and store them on a smart card.
You must:

- Install the `gnutls-utils` package, which helps you to manage certificates.
- Install the `opensc` package, which provides a set of libraries and utilities to work with smart cards.
- Start the `pcscd` service, which communicates with the smart card reader.

### Procedure

1. Install the `opensc` and `gnutls-utils` packages:
   ```bash
   # dnf -y install opensc gnutls-utils
   ```
2. Start the `pcscd` service.
   ```bash
   # systemctl start pcscd
   ```

Verify that the `pcscd` service is up and running.

### 2.8. PREPARING YOUR SMART CARD AND UPLOADING YOUR CERTIFICATES AND KEYS TO YOUR SMART CARD

This section describes smart card configuration with the `pkcs15-init` tool, which helps you to configure:

- Erasing your smart card
- Setting new PINs and optional PIN Unblocking Keys (PUKs)
- Creating a new slot on the smart card
- Storing the certificate, private key, and public key in the slot
- If required, locking the smart card settings as certain smart cards require this type of finalization

**NOTE**

The `pkcs15-init` tool may not work with all smart cards. You must use the tools that work with the smart card you are using.

### Prerequisites

- The `opensc` package, which includes the `pkcs15-init` tool, is installed. For details, see [Installing tools for managing and using smart cards](#).
- The card is inserted in the reader and connected to the computer.
- You have the private key, public key, and certificate to store on the smart card. In this procedure, `testuser.key`, `testuserpublic.key`, and `testuser.crt` are the names used for the private key, public key, and the certificate.
- You have your current smart card user PIN and Security Officer PIN (SO-PIN).
1. Erase your smart card and authenticate yourself with your PIN:

   $ pkcs15-init --erase-card --use-default-transport-keys

   Using reader with a card: Reader name

   PIN [Security Officer PIN] required.

   Please enter PIN [Security Officer PIN]:

   The card has been erased.

2. Initialize your smart card, set your user PIN and PUK, and your Security Officer PIN and PUK:

   $ pkcs15-init --create-pkcs15 --use-default-transport-keys

   --pin 963214 --puk 321478

   --so-pin 65498714 --so-puk 784123

   Using reader with a card: Reader name

   The pkcs15-init tool creates a new slot on the smart card.

3. Set the label and the authentication ID for the slot:

   $ pkcs15-init --store-pin --label testuser

   --auth-id 01 --so-pin 65498714 --pin 963214 --puk 321478

   Using reader with a card: Reader name

   The label is set to a human-readable value, in this case, testuser. The auth-id must be two hexadecimal values, in this case it is set to 01.

4. Store and label the private key in the new slot on the smart card:

   $ pkcs15-init --store-private-key testuser.key --label testuser_key

   --auth-id 01 --id 01 --pin 963214

   Using reader with a card: Reader name

   NOTE

   The value you specify for --id must be the same when storing your private key and storing your certificate in the next step. Specifying your own value for --id is recommended as otherwise a more complicated value is calculated by the tool.

5. Store and label the certificate in the new slot on the smart card:

   $ pkcs15-init --store-certificate testuser.crt --label testuser_crt

   --auth-id 01 --id 01 --format pem --pin 963214

   Using reader with a card: Reader name

6. (Optional) Store and label the public key in the new slot on the smart card:

   $ pkcs15-init --store-public-key testuserpublic.key

   --label testuserpublic_key --auth-id 01 --id 01 --pin 963214

   Using reader with a card: Reader name
NOTE

If the public key corresponds to a private key or certificate, specify the same ID as the ID of the private key or certificate.

7. (Optional) Certain smart cards require you to finalize the card by locking the settings:

```
$ pkcs15-init -F
```

At this stage, your smart card includes the certificate, private key, and public key in the newly created slot. You have also created your user PIN and PUK and the Security Officer PIN and PUK.

2.9. LOGGING IN TO IDM WITH SMART CARDS

This section describes using smart cards for logging in to the IdM Web UI.

Prerequisites

- The web browser is configured for using smart card authentication.
- The IdM server is configured for smart card authentication.
- The certificate installed on your smart card is either issued by the IdM server or has been added to the user entry in IdM.
- You know the PIN required to unlock the smart card.
- The smart card has been inserted into the reader.

Procedure

1. Open the IdM Web UI in the browser.

2. Click Log In Using Certificate

![RED HAT® IDENTITY MANAGEMENT](image)

3. If the Password Required dialog box opens, add the PIN to unlock the smart card and click the OK button.

   The User Identification Request dialog box opens.

   If the smart card contains more than one certificate, select the certificate you want to use for authentication in the drop down list below **Choose a certificate to present as identification**
4. Click the **OK** button.

Now you are successfully logged in to the IdM Web UI.

2.10. LOGGING IN TO GDM USING SMART CARD AUTHENTICATION ON AN IDM CLIENT

The Gnome Desktop Manager (GDM) requires authentication. You can use your password; however, you can also use a smart card for authentication.

This section describes smart card authentication to access GDM.

**Prerequisites**

- The system has been configured for smart card authentication. For details, see [Configuring the IdM client for smart card authentication](#).
- The smart card contains your certificate and private key.
- The user account is a member of the IdM domain.
- The certificate on the smart card maps to the user entry through:
  - Assigning the certificate to a particular user entry. For details, see [Adding a certificate to a user entry in the IdM Web UI](#) or [Adding a certificate to a user entry in the IdM CLI](#).
  - The certificate mapping data being applied to the account. For details, see [Certificate mapping rules for configuring authentication on smart cards](#).

**Procedure**

1. Insert the smart card in the reader.
2. Enter the smart card PIN.
3. Click **Sign In**.

You are successfully logged in to the RHEL system and you have a TGT provided by the IdM server.

**Verification steps**
In the Terminal window, enter `klist` and check the result:

```
$ klist
Ticket cache: KEYRING:persistent:1358900015:krb_cache_TObtNMd
Default principal: example.user@REDHAT.COM

Valid starting   Expires       Service principal
04/20/2020 13:58:24  04/20/2020 23:58:24  krbtgt/EXAMPLE.COM@EXAMPLE.COM
renew until 04/27/2020 08:58:15
```

### 2.11. USING SMART CARD AUTHENTICATION WITH THE SU COMMAND

Changing to a different user requires authentication. You can use a password or a certificate. This section describes using your smart card with the `su` command. It means that after entering the `su` command, you are prompted for the smart card PIN.

**Prerequisites**

- Your IdM server and client have been configured for smart card authentication.
  - See [Configuring the IdM server for smart card authentication](#)
  - See [Configuring the IdM client for smart card authentication](#)
- The smart card contains your certificate and private key. See [Storing a certificate on a smart card](#)
- The card is inserted in the reader and connected to the computer.

**Procedure**

- In a terminal window, change to a different user with the `su` command:

```
$ su - example.user
PIN for smart_card
```

If the configuration is correct, you are prompted to enter the smart card PIN.
CHAPTER 3. CONFIGURING CERTIFICATES ISSUED BY ADCS FOR SMART CARD AUTHENTICATION IN IDM

This scenario describes the following situation:

- Your deployment is based on cross-forest trust between Identity Management (IdM) and Active Directory (AD).
- You want to allow smart card authentication for users whose accounts are stored in AD.
- Certificates are created and stored in Active Directory Certificate Services (ADCS).

For an overview of smart card authentication, see Understanding smart card authentication.

Configuration will be accomplished in the following steps:

- Copying CA and user certificates from Active Directory to the IdM server and client
- Configuring the IdM server and clients for smart card authentication using ADCS certificates
- Converting a PFX (PKCS#12) file to be able to store the certificate and private key into the smart card
- Configuring timeouts in the sssd.conf file
- Creating certificate mapping rules for smart card authentication

Prerequisites

- Identity Management (IdM) and Active Directory (AD) trust is installed
  For details, see Installing trust between IdM and AD.
- Active Directory Certificate Services (ADCS) is installed and certificates for users are generated

3.1. WINDOWS SERVER SETTINGS REQUIRED FOR TRUST CONFIGURATION AND CERTIFICATE USAGE

This section summarizes what must be configured on Windows Server:

- Active Directory Certificate Services (ADCS) is installed
- Certificate Authority is created
- [Optional] If you are using Certificate Authority Web Enrollment, the Internet Information Services (IIS) must be configured

Export the certificate:

- Key must have 2048 bits or more
- Include a private key
- You will need a certificate in the following format: Personal Information Exchange – PKCS #12(.PFX)
Enable certificate privacy

3.2. COPYING CERTIFICATES FROM ACTIVE DIRECTORY USING SFTP

To be able to use smart card authentication, you need to copy the following certificate files:

- A root CA certificate in the CER format: adcs-winserver-ca.cer on your IdM server.
- A user certificate with a private key in the PFX format: aduser1.pfx on an IdM client.

**NOTE**

This procedure expects SSH access is allowed. If SSH is unavailable, the user must copy the file from the AD Server to the IdM server and client.

**Procedure**

1. Connect from the IdM server and copy the adcs-winserver-ca.cer root certificate to the IdM server:

   ```
   root@idmserver ~]# sftp Administrator@winserver.ad.example.com
   Administrator@winserver.ad.example.com's password: 
   Connected to Administrator@winserver.ad.example.com.
   sftp> cd <Path to certificates>
   sftp> ls
   adcs-winserver-ca.cer  aduser1.pfx
   sftp>
   sftp> get adcs-winserver-ca.cer
   Fetching <Path to certificates>/adcs-winserver-ca.cer to adcs-winserver-ca.cer
   <Path to certificates>/adcs-winserver-ca.cer                 100%  1254    15KB/s 00:00
   sftp quit
   ```

2. Connect from the IdM client and copy the aduser1.pfx user certificate to the client:

   ```
   [root@client1 ~]# sftp Administrator@winserver.ad.example.com
   Administrator@winserver.ad.example.com's password: 
   Connected to Administrator@winserver.ad.example.com.
   sftp> cd /<Path to certificates>
   sftp> get aduser1.pfx
   Fetching <Path to certificates>/aduser1.pfx to aduser1.pfx
   <Path to certificates>/aduser1.pfx                 100%  1254    15KB/s 00:00
   sftp quit
   ```

Now the CA certificate is stored in the IdM server and the user certificates is stored on the client machine.

3.3. CONFIGURING THE IDM SERVER AND CLIENTS FOR SMART CARD AUTHENTICATION USING ADCS CERTIFICATES

You must configure the IdM (Identity Management) server and clients to be able to use smart card authentication in the IdM environment. IdM includes the `ipa-advice` scripts which makes all necessary changes:
- install necessary packages
- it configures IdM server and clients
- copy the CA certificates into expected locations

You can run `ipa-advise` on your IdM server.

This procedure describes:

- On an IdM server: Preparing the `ipa-advise` script to configure your IdM server for smart card authentication.
- On an IdM server: Preparing the `ipa-advise` script to configure your IdM client for smart card authentication.
- On an IdM server: Applying the the `ipa-advise` server script on the IdM server using the AD certificate.
- Moving the client script to the IdM client machine.
- On an IdM client: Applying the the `ipa-advise` client script on the IdM client using the AD certificate.

Prerequisites

- The certificate has been copied to the IdM server.
- Obtain the Kerberos ticket.
- Log in as a user with administration rights.

Procedure

1. On the IdM server, use the `ipa-advise` script for configuring a client:

   ```bash
   [root@idmserver ~]# ipa-advise config-client-for-smart-card-auth > sc_client.sh
   ```

2. On the IdM server, use the `ipa-advise` script for configuring a server:

   ```bash
   [root@idmserver ~]# ipa-advise config-server-for-smart-card-auth > sc_server.sh
   ```

3. On the IdM server, execute the script:

   ```bash
   [root@idmserver ~]# sh -x sc_server.sh adcs-winserver-ca.cer
   ```
   - It configures the IdM Apache HTTP Server.
   - It enables Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) on the Key Distribution Center (KDC).
   - It configures the IdM Web UI to accept smart card authorization requests.

4. Copy the `sc_client.sh` script to the client system:
5. Copy the Windows certificate to the client system:

```
[root@idmserver ~]# scp adcs-winserver-ca.cer root@client1.idm.example.com:/root
Password:
adcs-winserver-ca.cer                 100%  1254   952.0KB/s   00:00
```

6. On the client system, run the client script:

```
[root@idmclient1 ~]# sh -x sc_client.sh adcs-winserver-ca.cer
```

The CA certificate is installed in the correct format on the IdM server and client systems and next step is to copy the user certificates onto the smart card itself.

### 3.4. CONVERTING THE PFX FILE

Before you store the PFX (PKCS#12) file into the smart card, you must:

- convert the file to the PEM format
- extract the private key and the certificate to two different files

**Prerequisites**

- The PFX file is copied into the IdM client machine.

**Procedure**

1. On the IdM client, into the PEM format:

```
[root@idmclient1 ~]# openssl pkcs12 -in aduser1.pfx -out aduser1_cert_only.pem -clcerts -nodes
Enter Import Password:
```

2. Extract the key into the separate file:

```
[root@idmclient1 ~]# openssl pkcs12 -in aduser1.pfx -nocerts -out aduser1.pem > aduser1.key
```

3. Extract the public certificate into the separate file:

```
[root@idmclient1 ~]# openssl pkcs12 -in aduser1.pfx -clcerts -nokeys -out aduser1_cert_only.pem > aduser1.crt
```

At this point, you can store the `aduser1.key` and `aduser1.crt` into the smart card.

### 3.5. INSTALLING TOOLS FOR MANAGING AND USING SMART CARDS
To configure your smart card, you need tools which can generate certificates and store them on a smart card.

You must:

- Install the `gnutls-utils` package, which helps you to manage certificates.
- Install the `opensc` package, which provides a set of libraries and utilities to work with smart cards.
- Start the `pcscd` service, which communicates with the smart card reader.

**Procedure**

1. Install the `opensc` and `gnutls-utils` packages:
   
   ```bash
   # dnf -y install opensc gnutls-utils
   ```

2. Start the `pcscd` service.

   ```bash
   # systemctl start pcscd
   ```

   Verify that the `pcscd` service is up and running.

3.6. PREPARING YOUR SMART CARD AND UPLOADING YOUR CERTIFICATES AND KEYS TO YOUR SMART CARD

This section describes smart card configuration with the `pkcs15-init` tool, which helps you to configure:

- Erasing your smart card
- Setting new PINs and optional PIN Unblocking Keys (PUKs)
- Creating a new slot on the smart card
- Storing the certificate, private key, and public key in the slot
- If required, locking the smart card settings as certain smart cards require this type of finalization

**NOTE**

The `pkcs15-init` tool may not work with all smart cards. You must use the tools that work with the smart card you are using.

**Prerequisites**

- The `opensc` package, which includes the `pkcs15-init` tool, is installed. For details, see [Installing tools for managing and using smart cards](#).
- The card is inserted in the reader and connected to the computer.
- You have the private key, public key, and certificate to store on the smart card. In this procedure, `testuser.key`, `testuserpublic.key`, and `testuser.crt` are the names used for the private key, public key, and the certificate.
.. You have your current smart card user PIN and Security Officer PIN (SO-PIN).

**Procedure**

1. Erase your smart card and authenticate yourself with your PIN:

   ```
   $ pkcs15-init --erase-card --use-default-transport-keys
   Using reader with a card: Reader name
   PIN [Security Officer PIN] required.
   Please enter PIN [Security Officer PIN]:
   
   The card has been erased.
   ```

2. Initialize your smart card, set your user PIN and PUK, and your Security Officer PIN and PUK:

   ```
   $ pkcs15-init --create-pkcs15 --use-default-transport-keys \
   --pin 963214 --puk 321478 --so-pin 65498714 --so-puk 784123
   Using reader with a card: Reader name
   
   The `pkcs15-init` tool creates a new slot on the smart card.
   ```

3. Set the label and the authentication ID for the slot:

   ```
   $ pkcs15-init --store-pin --label testuser \
   --auth-id 01 --so-pin 65498714 --pin 963214 --puk 321478
   Using reader with a card: Reader name
   
   The label is set to a human-readable value, in this case, testuser. The **auth-id** must be two hexadecimal values, in this case it is set to **01**.
   ```

4. Store and label the private key in the new slot on the smart card:

   ```
   $ pkcs15-init --store-private-key testuser.key --label testuser_key \
   --auth-id 01 --id 01 --pin 963214
   Using reader with a card: Reader name
   
   **NOTE**
   The value you specify for **--id** must be the same when storing your private key and storing your certificate in the next step. Specifying your own value for **--id** is recommended as otherwise a more complicated value is calculated by the tool.
   ```

5. Store and label the certificate in the new slot on the smart card:

   ```
   $ pkcs15-init --store-certificate testuser.crt --label testuser_crt \
   --auth-id 01 --id 01 --format pem --pin 963214
   Using reader with a card: Reader name
   ```

6. (Optional) Store and label the public key in the new slot on the smart card:

   ```
   $ pkcs15-init --store-public-key testuserpublic.key \
   --label testuserpublic_key --auth-id 01 --id 01 --pin 963214
   Using reader with a card: Reader name
   ```
NOTE

If the public key corresponds to a private key or certificate, specify the same ID as the ID of the private key or certificate.

7. (Optional) Certain smart cards require you to finalize the card by locking the settings:

```
$ pkcs15-init -F
```

At this stage, your smart card includes the certificate, private key, and public key in the newly created slot. You have also created your user PIN and PUK and the Security Officer PIN and PUK.

### 3.7. CONFIGURING TIMEOUTS IN SSSD.CONF

Authentication with a smart card certificate might take longer than the default timeouts used by SSSD. Time out expiration can be caused by:

- slow reader
- a forwarding form a physical device into a virtual environment
- too many certificates stored on the smart card
- slow response from the OCSP (Online Certificate Status Protocol) responder if OCSP is used to verify the certificates

In this case you can prolong the following timeouts in the `sssd.conf` file, for example, to 60 seconds:

- `p11_child_timeout`
- `krb5_auth_timeout`

#### Prerequisites

- You must be logged in as root.

#### Procedure

1. Open the `sssd.conf` file:

   ```
   [root@idmclient1 ~]# vim /etc/sssd/sssd.conf
   ```

2. Change the value of `p11_child_timeout`:

   ```
   [pam]
   p11_child_timeout = 60
   ```

3. Change the value of `krb5_auth_timeout`:

   ```
   [domain/IDM_EXAMPLE.COM]
   krb5_auth_timeout = 60
   ```

4. Save the settings.
Now, the interaction with the smart card is allowed to run for 1 minute (60 seconds) before authentication will fail with a timeout.

3.8. CREATING CERTIFICATE MAPPING RULES FOR SMART CARD AUTHENTICATION

If you want to use one certificate for a user who has accounts in AD (Active Directory) and in IdM (Identity Management), you can create a certificate mapping rule on the IdM server.

After creating such a rule, the user is able to authenticate with their smart card in both domains.

For details about certificate mapping rules, see Certificate mapping rules for configuring authentication on smart cards.
CHAPTER 4. CERTIFICATE MAPPING RULES FOR CONFIGURING AUTHENTICATION ON SMART CARDS

Certificate mapping rules are a convenient way of allowing users to authenticate using certificates in scenarios when the Identity Management (IdM) administrator does not have access to certain users’ certificates. This lack of access is typically caused by the fact that the certificates have been issued by an external certificate authority. A special use case is represented by certificates issued by the Certificate System of an Active Directory (AD) with which the IdM domain is in a trust relationship.

Certificate mapping rules are also convenient if the IdM environment is large with a lot of users using smart cards. In this situation, adding full certificates can be complicated. The subject and issuer are predictable in most scenarios and thus easier to add ahead of time than the full certificate. As a system administrator, you can create a certificate mapping rule and add certificate mapping data to a user entry even before a certificate is issued to a particular user. Once the certificate is issued, the user can log in using the certificate even though the full certificate has not yet been uploaded to the user entry.

In addition, as certificates have to be renewed at regular intervals, certificate mapping rules reduce administrative overhead. When a user’s certificate gets renewed, the administrator does not have to update the user entry. For example, if the mapping is based on the Subject and Issuer values, and if the new certificate has the same subject and issuer as the old one, the mapping still applies. If, in contrast, the full certificate was used, then the administrator would have to upload the new certificate to the user entry to replace the old one.

To set up certificate mapping:

1. An administrator has to load the certificate mapping data (typically the issuer and subject) or the full certificate into a user account.

2. An administrator has to create a certificate mapping rule to allow successful logging into IdM for a user
   a. whose account contains a certificate mapping data entry
   b. whose certificate mapping data entry matches the information on the certificate

For details on the individual components that make up a mapping rule and how to obtain and use them, see Components of an identity mapping rule in IdM and Obtaining the issuer from a certificate for use in a matching rule.

Afterwards, when the end-user presents the certificate, stored either in the filesystem or on a smart card, authentication is successful.

4.1. CERTIFICATE MAPPING RULES FOR TRUSTS WITH ACTIVE DIRECTORY DOMAINS

This section outlines the different certificate mapping use cases that are possible if an IdM deployment is in a trust relationship with an Active Directory (AD) domain.

Certificate mapping rules are a convenient way to enable access to IdM resources for users who have smart card certificates that were issued by the trusted AD Certificate System. Depending on the AD configuration, the following scenarios are possible:

- If the certificate is issued by AD but the user and the certificate are stored in IdM, the mapping and the whole processing of the authentication request takes place on the IdM side. For details of configuring this scenario, see Configuring certificate mapping for users stored in IdM.
If the user is stored in AD, the processing of the authentication request takes place in AD. There are three different subcases:

- The AD user entry contains the whole certificate. For details how to configure IdM in this scenario, see Configuring certificate mapping for users whose AD user entry contains the whole certificate.

- AD is configured to map user certificates to user accounts. In this case, the AD user entry does not contain the whole certificate but instead contains an attribute called altSecurityIdentities. For details how to configure IdM in this scenario, see Configuring certificate mapping if AD is configured to map user certificates to user accounts.

- The AD user entry contains neither the whole certificate nor the mapping data. In this case, the only solution is to use the `ipa idoverrideuser-add` command to add the whole certificate to the AD user’s ID override in IdM. For details, see Configuring certificate mapping if AD user entry contains no certificate or mapping data.

### 4.2. COMPONENTS OF AN IDENTITY MAPPING RULE IN IDM

This section describes the components of an identity mapping rule in IdM and how to configure them. Each component has a default value that you can override. You can define the components in either the web UI or the CLI. In the CLI, the identity mapping rule is created using the `ipa certmaprule-add` command.

**Mapping rule**

The mapping rule component associates (or maps) a certificate with one or more user accounts. The rule defines an LDAP search filter that associates a certificate with the intended user account. Certificates issued by different certificate authorities (CAs) might have different properties and might be used in different domains. Therefore, IdM does not apply mapping rules unconditionally, but only to the appropriate certificates. The appropriate certificates are defined using matching rules.

Note that if you leave the mapping rule option empty, the certificates are searched in the `userCertificate` attribute as a DER encoded binary file.

Define the mapping rule in the CLI using the `--maprule` option.

**Matching rule**

The matching rule component selects a certificate to which you want to apply the mapping rule. The default matching rule matches certificates with the `digitalSignature key` usage and `clientAuth extended key` usage.

Define the matching rule in the CLI using the `--matchrule` option.

**Domain list**

The domain list specifies the identity domains in which you want IdM to search the users when processing identity mapping rules. If you leave the option unspecified, IdM searches the users only in the local domain to which the IdM client belongs.

Define the domain in the CLI using the `--domain` option.

**Priority**

When multiple rules are applicable to a certificate, the rule with the highest priority takes precedence. All other rules are ignored.

- The lower the numerical value, the higher the priority of the identity mapping rule. For example, a rule with a priority 1 has higher priority than a rule with a priority 2.
If a rule has no priority value defined, it has the lowest priority.

Define the mapping rule priority in the CLI using the \texttt{--priority} option.

Certificate mapping rule example

To define, using the CLI, a certificate mapping rule called \texttt{simple\_rule} that allows authentication for a certificate issued by the \texttt{Smart Card CA} of the \texttt{EXAMPLE.ORG} organisation as long as the \texttt{Subject} on that certificate matches a \texttt{certmapdata} entry in a user account in IdM:

\begin{verbatim}
# ipa certmaprule-add simple\_rule --matchrule \texttt{\'<ISSUER>CN=Smart Card CA,O=EXAMPLE.ORG' --maprule \texttt{'(ipacertmapdata=X509:<I>{issuer_dn!nss\_x500}<S>{subject_dn!nss\_x500})'\n
4.3. OBTAINING THE ISSUER FROM A CERTIFICATE FOR USE IN A MATCHING RULE

This procedure describes how to obtain the issuer information from a certificate so that you can copy and paste it into the matching rule of a certificate mapping rule. To get the issuer format required by a matching rule, use the \texttt{openssl x509} utility.

Prerequisites

- You have the user certificate in a \texttt{.pem} or \texttt{.crt} format

Procedure

1. Obtain the user information from the certificate. Use the \texttt{openssl x509} certificate display and signing utility with:

   - the \texttt{-noout} option to prevent the output of an encoded version of the request
   - the \texttt{-issuer} option to output the issuer name
   - the \texttt{-in} option to specify the input file name to read the certificate from
   - the \texttt{-nameopt} option with the \texttt{RFC2253} value to display the output with the most specific relative distinguished name (RDN) first
   
   If the input file contains an Identity Management certificate, the output of the command shows that the Issuer is defined using the \texttt{Organisation} information:

   \begin{verbatim}
   # openssl x509 -noout -issuer -in idm\_user.crt -nameopt RFC2253
   issuer=CN=Certificate Authority,O=REALM.EXAMPLE.COM
   \end{verbatim}

   If the input file contains an Active Directory certificate, the output of the command shows that the Issuer is defined using the \texttt{Domain Component} information:

   \begin{verbatim}
   # openssl x509 -noout -issuer -in ad\_user.crt -nameopt RFC2253
   issuer=CN=AD-WIN2012R2-CA,DC=AD,DC=EXAMPLE,DC=COM
   \end{verbatim}

2. Optionally, to create a new mapping rule in the CLI based on a matching rule which specifies that the certificate issuer must be the extracted \texttt{AD-WIN2012R2-CA} of the \texttt{ad.example.com} domain and the subject on the certificate must match the \texttt{certmapdata} entry in a user account
in IdM:

```
# ipa certmaprule-add simple_rule --matchrule '"<ISSUER>CN=AD-WIN2012R2-CA,DC=AD,DC=EXAMPLE,DC=COM' --maprule '(ipacertmapdata=X509:<I> {issuer_dn!nss_x500}<S>{subject_dn!nss_x500})'"
```

4.4. ADDITIONAL RESOURCES

- See the `sss-certmap(5)` man page.
CHAPTER 5. CONFIGURING SMART CARD AUTHENTICATION WITH THE WEB CONSOLE FOR CENTRALLY MANAGED USERS

Configure smart card authentication in the RHEL web console for users who are centrally managed by:

- Identity Management
- Active Directory which is connected in the cross-forest trust with Identity Management

Prerequisites

- The system for which you want to use the smart card authentication must be a member of an Active Directory or Identity Management domain.
- The certificate used for the smart card authentication must be associated with a particular user in Identity Management or Active Directory.
  For more details about associating a certificate with the user in Identity Management, see Adding a certificate to a user entry in the IdM Web UI or Adding a certificate to a user entry in the IdM CLI.

5.1. SMART CARD AUTHENTICATION FOR CENTRALLY MANAGED USERS

A smart card is a physical device, which can provide personal authentication using certificates stored on the card. Personal authentication means that you can use smart cards in the same way as user passwords.

You can store user credentials on the smart card in the form of a private key and a certificate. Special software and hardware is used to access them. You insert the smart card into a reader or a USB socket and supply the PIN code for the smart card instead of providing your password.

Identity Management (IdM) supports smart card authentication with:

- User certificates issued by the IdM certificate authority.
- User certificates issued by the Active Directory Certificate Service (ADCS) certificate authority.

**NOTE**

If you want to start using smart card authentication, see the hardware requirements: Smart Card support in RHEL8+.

5.2. INSTALLING TOOLS FOR MANAGING AND USING SMART CARDS

To configure your smart card, you need tools which can generate certificates and store them on a smart card.

You must:

- Install the gnuts-utils package, which helps you to manage certificates.
- Install the opensc package, which provides a set of libraries and utilities to work with smart cards.

- Start the pcscd service, which communicates with the smart card reader.

**Procedure**

1. Install the opensc and gnutls-utils packages:
   ```shell
   # dnf -y install opensc gnutls-utils
   ```

2. Start the pcscd service.
   ```shell
   # systemctl start pcscd
   ```

Verify that the pcscd service is up and running.

**5.3. PREPARING YOUR SMART CARD AND UPLOADING YOUR CERTIFICATES AND KEYS TO YOUR SMART CARD**

This section describes smart card configuration with the pkcs15-init tool, which helps you to configure:

- Erasing your smart card
- Setting new PINs and optional PIN Unblocking Keys (PUKs)
- Creating a new slot on the smart card
- Storing the certificate, private key, and public key in the slot
- If required, locking the smart card settings as certain smart cards require this type of finalization

**NOTE**

The pkcs15-init tool may not work with all smart cards. You must use the tools that work with the smart card you are using.

**Prerequisites**

- The opensc package, which includes the pkcs15-init tool, is installed. For details, see *Installing tools for managing and using smart cards*.

- The card is inserted in the reader and connected to the computer.

- You have the private key, public key, and certificate to store on the smart card. In this procedure, `testuser.key`, `testuserpublic.key`, and `testuser.crt` are the names used for the private key, public key, and the certificate.

- You have your current smart card user PIN and Security Officer PIN (SO-PIN).

**Procedure**

1. Erase your smart card and authenticate yourself with your PIN:
$ pkcs15-init --erase-card --use-default-transport-keys
    Using reader with a card: Reader name
    PIN [Security Officer PIN] required.
    Please enter PIN [Security Officer PIN]:

    The card has been erased.

2. Initialize your smart card, set your user PIN and PUK, and your Security Officer PIN and PUK:

    $ pkcs15-init --create-pkcs15 --use-default-transport-keys \
        --pin 963214 --puk 321478 --so-pin 65498714 --so-puk 784123
        Using reader with a card: Reader name

    The pkcs15-init tool creates a new slot on the smart card.

3. Set the label and the authentication ID for the slot:

    $ pkcs15-init --store-pin --label testuser \
        --auth-id 01 --so-pin 65498714 --pin 963214 --puk 321478
        Using reader with a card: Reader name

    The label is set to a human-readable value, in this case, testuser. The auth-id must be two hexadecimal values, in this case it is set to 01.

4. Store and label the private key in the new slot on the smart card:

    $ pkcs15-init --store-private-key testuser.key --label testuser_key \
        --auth-id 01 --id 01 --pin 963214
        Using reader with a card: Reader name

    NOTE
    The value you specify for --id must be the same when storing your private key and storing your certificate in the next step. Specifying your own value for --id is recommended as otherwise a more complicated value is calculated by the tool.

5. Store and label the certificate in the new slot on the smart card:

    $ pkcs15-init --store-certificate testuser.crt --label testuser_crt \
        --auth-id 01 --id 01 --format pem --pin 963214
        Using reader with a card: Reader name

6. (Optional) Store and label the public key in the new slot on the smart card:

    $ pkcs15-init --store-public-key testuserpublic.key \
        --label testuserpublic_key --auth-id 01 --id 01 --pin 963214
        Using reader with a card: Reader name

    NOTE
    If the public key corresponds to a private key or certificate, specify the same ID as the ID of the private key or certificate.
7. (Optional) Certain smart cards require you to finalize the card by locking the settings:

```
$ pkcs15-init -F
```

At this stage, your smart card includes the certificate, private key, and public key in the newly created slot. You have also created your user PIN and PUK and the Security Officer PIN and PUK.

### 5.4. ENABLING SMART CARD AUTHENTICATION FOR THE WEB CONSOLE

To be able to use smart card authentication in the web console, enable smart card authentication in the `cockpit.conf` file.

Additionally, you can disable password authentication in the same file.

**Prerequisites**

- The RHEL web console has been installed.

**Procedure**

1. Log in to the RHEL web console with administrator privileges.

2. Click **Terminal**.

3. In the `/etc/cockpit/cockpit.conf`, set the `ClientCertAuthentication` to **yes**:

```
[WebService]
ClientCertAuthentication = yes
```

4. Optionally, disable password based authentication in `cockpit.conf` with:

```
[Basic]
action = none
```

   This configuration disables password authentication and you must always use the smart card.

5. Restart the web console to ensure that the `cockpit.service` accepts the change:

```
# systemctl restart cockpit
```

### 5.5. LOGGING IN TO THE WEB CONSOLE WITH SMART CARDS

You can use smart cards to log in to the web console.

**Prerequisites**

- A valid certificate stored in your smart card that is associated to a user account created in a Active Directory or Identity Management domain.

- PIN to unlock the smart card.
The smart card has been put into the reader.

Procedure

1. Open your web browser and add the web console’s address in the address bar. The browser asks you to add the PIN protecting the certificate stored on the smart card.

2. In the Password Required dialog box, enter PIN and click OK.

3. In the User Identification Request dialog box, select the certificate stored in the smart card.

4. Select Remember this decision. The system does not open this window next time.

   NOTE
   This step does not apply to Google Chrome users.

5. Click OK.

You are now connected and the web console displays its content.

5.6. ENABLING PASSWORD-LESS SUDO FOR SMART CARD USERS

Once you logged into web console with a certificate, you may need to switch to administrative mode (root privileges through sudo). If your user account has a password, it can be used to authenticate to sudo.

As an alternative, if you are using Red Hat Identity Management, you can declare the initial web console certificate authentication as trusted for authenticating to sudo, SSH, or other services. For that purpose, the web console automatically creates an S4U2Proxy Kerberos ticket in the user session.

Prerequisites

- Identity Management
- Active Directory connected in the cross-forest trust with Identity Management
- Smart card set up to log in to the web console. See Configuring smart card authentication with the web console for centrally managed users for more information.

Procedure

1. Set up constraint delegation rules to list which hosts the ticket can access.

   **Example 5.1. Setting up constraint delegation rules**

   The web console session runs host host.example.com and should be trusted to access its own host with sudo. Additionally, we are adding second trusted host – remote.example.com.

   - Create the following delegation:
     - Run the following commands to add a list of target machines a particular rule can access:
To allow web console sessions (HTTP/principal) to access that host list, run the following commands:

```
# ipa servicedelegationtarget-add cockpit-target
# ipa servicedelegationtarget-add-member cockpit-target
    --principals=host/host.example.com@EXAMPLE.COM
    --principals=host/remote.example.com@EXAMPLE.COM
```

To allow the delegated S4U ticket is not forwarded to remote SSH hosts when connecting to them from the web console. Authenticating to sudo on a remote host with your ticket will not work.

2. Enable GSS authentication in the corresponding services:
   a. For sudo, enable the `pam_sss_gss` module in the `/etc/sssd/sssd.conf` file:
      i. As root, add an entry for your domain to the `/etc/sssd/sssd.conf` configuration file.
      
         ```
         [domain/example.com]
         pam_gssapi_services = sudo, sudo-i
         ```
      
      ii. Enable the module in the `/etc/pam.d/sudo` file on the first line.
      
         ```
         auth sufficient pam_sss_gss.so
         ```
   b. For SSH, update the `GSSAPIAuthentication` option in the `/etc/ssh/sshd_config` file to `yes`.

**WARNING**

The delegated S4U ticket is not forwarded to remote SSH hosts when connecting to them from the web console. Authenticating to sudo on a remote host with your ticket will not work.

**Verification**

1. Log in to the web console using a smart card.
2. Click the **Limited access** button.
3. Authenticate using your smart card.

**OR**

1. Try to connect to a different host with SSH.
5.7. LIMITING USER SESSIONS AND MEMORY TO PREVENT A DOS ATTACK

Certificate authentication is protected by separating and isolating instances of the cockpit-ws web server against attackers who wants to impersonate another user. However, this introduces a potential Denial of Service (DoS) attack: A remote attacker could create a large number of certificates and send a large number of HTTPS requests to cockpit-ws each using a different certificate.

To prevent this DoS, the collective resources of these web server instances are limited. By default, limits to the number of connections and to memory usage are set to 200 threads and a 75% (soft) / 90% (hard) memory limit.

The following procedure describes resource protection by limiting the number of connections and memory.

Procedure

1. In the terminal, open the system-cockpithttps.slice configuration file:

   # systemctl edit system-cockpithttps.slice

2. Limit the TasksMax to 100 and CPUQuota to 30%:

   [Slice]
   # change existing value
   TasksMax=100
   # add new restriction
   CPUQuota=30%

3. To apply the changes, restart the system:

   # systemctl daemon-reload
   # systemctl stop cockpit

Now, the new memory and user session limits protect the cockpit-ws web server from DoS attacks.
CHAPTER 6. CONFIGURING SMART CARD AUTHENTICATION WITH LOCAL CERTIFICATES

This chapter describes a scenario where:

- The host is not connected to a domain.
- You want to authenticate with a smart card on this host.
- You want to configure SSH access using smart card authentication.
- You want to configure the smart card with authselect.

Use the following configuration to accomplish this scenario:

- Obtain a user certificate for the user who wants to authenticate with a smart card. The certificate should be generated by a trustworthy Certification Authority used in the domain. If you cannot get the certificate, you can generate a user certificate signed by a local certificate authority for testing purposes,

- Store the certificate and private key in a smart card.

- Configure the smart card authentication for SSH access.

**IMPORTANT**

If a host can be part of the domain, add the host to the domain and use certificates generated by Active Directory or Identity Management Certification Authority.

For details about how to create IdM certificates for a smart card, see Configuring Identity Management for smart card authentication.

**Prerequisites**

- Authselect installed
  The authselect tool configures user authentication on Linux hosts and you can use it to configure smart card authentication parameters. For details about authselect, see Explaining authselect.

- Smart Card or USB devices supported by RHEL 9
  For details, see Smart Card support in RHEL 9.

**6.1. CREATING LOCAL CERTIFICATES**

This section describes how to perform these tasks:

- Generate the OpenSSL certificate authority

- Create a certificate signing request
WARNING

The following steps are intended for testing purposes only. Certificates generated by a local self-signed Certificate Authority are not as secure as using AD, IdM, or RHCS Certification Authority. You should use a certificate generated by your enterprise Certification Authority even if the host is not part of the domain.

Procedure

1. Create a directory where you can generate the certificate, for example:

   ```
   # mkdir /tmp/ca
   # cd /tmp/ca
   ```

2. Set up the certificate (copy this text to your command line in the ca directory):

   ```
   cat > ca.cnf <<EOF
   [ ca ]
   default_ca = CA_default

   [ CA_default ]
   dir = .
   database = \$dir/index.txt
   new_certs_dir = \$dir/newcerts
   certificate = \$dir/rootCA.crt
   serial = \$dir/serial
   private_key = \$dir/rootCA.key
   RANDFILE = \$dir/rand
   default_days = 365
   default_crl_days = 30
   default_md = sha256
   policy = policy_any
   email_in_dn = no
   name_opt = ca_default
   cert_opt = ca_default
   copy_extensions = copy

   [ usr_cert ]
   authorityKeyIdentifier = keyid, issuer

   [ v3_ca ]
   subjectKeyIdentifier = hash
   authorityKeyIdentifier = keyid:always,issuer:always
   basicConstraints = CA:true
   keyUsage = critical, digitalSignature, cRLSign, keyCertSign

   [ policy_any ]
   ```
3. Create the following directories:

```
# mkdir certs crl newcerts
```

4. Create the following files:

```
# touch index.txt crlnumber index.txt.attr
```

5. Write the number 01 in the serial file:

```
# echo 01 > serial
```

This command writes a number 01 in the serial file. It is a serial number of the certificate. With each new certificate released by this CA the number increases by one.

6. Create an OpenSSL root CA key:

```
# openssl genrsa -out rootCA.key 2048
```

7. Create a self-signed root Certification Authority certificate:

```
# openssl req -batch -config ca.cnf \
-x509 -new -nodes -key rootCA.key -sha256 -days 10000 \ 
-set_serial 0 -extensions v3_ca -out rootCA.crt
```

8. Create the key for your username:

```
# openssl genrsa -out example.user.key 2048
```

This key is generated in the local system which is not secure, therefore, remove the key from the system when the key is stored in the card.

You can create a key directly in the smart card as well. For doing this, follow instructions created by the manufacturer of your smart card.

9. Create the certificate signing request configuration file (copy this text to your command line in the ca directory):

```
cat > req.cnf <<EOF
```

Red Hat Enterprise Linux 9 Managing smart card authentication
[ req ]
distinguished_name = req_distinguished_name
prompt = no

[ req_distinguished_name ]
O = Example
OU = Example Test
CN = testuser

[ req_exts ]
basicConstraints = CA:FALSE
nsCertType = client, email
nsComment = “testuser”
subjectKeyIdentifier = hash
keyUsage = critical, nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth, emailProtection, msSmartcardLogin
subjectAltName = otherName:msUPN;UTF8:testuser@example.com,
email:testuser@example.com
EOF

10. Create a certificate signing request for your example.user certificate:

```
# openssl req -new -nodes -key example.user.key 
-reqexts req_exts -config req.cnf -out example.user.csr
```

11. Configure the new certificate. Expiration period is set to 1 year:

```
# openssl ca -config ca.cnf -batch -notext 
-keyfile rootCA.key -in example.user.csr -days 365 
-extensions usr_cert -out example.user.crt
```

At this point, the certification authority and certificates are successfully generated and prepared for import into a smart card.

### 6.2. COPYING CERTIFICATES TO THE SSSD DIRECTORY

Gnome Desktop Manager (GDM) requires SSSD. If you use GDM, you need to copy the PEM certificate to the `/etc/sssd/pki` directory.

**Prerequisites**

- The local CA authority and certificates have been generated

**Procedure**

1. Ensure that you have SSSD installed on the system.

```
# rpm -q sssd
sssd-2.0.0.43.el8_0.3.x86_64
```

2. Create a `/etc/sssd/pki` directory:
3. Copy the rootCA.crt as a PEM file in the /etc/sssd/pki directory:

```bash
# cp /tmp/ca/rootCA.crt /etc/sssd/pki/sssd_auth_ca_db.pem
```

Now you have successfully generated the certificate authority and certificates, and you have saved them in the /etc/sssd/pki directory.

**NOTE**

If you want to share the Certificate Authority certificates with another application, you can change the location in sssd.conf:

- SSSD PAM responder: `pam_cert_db_path` in the `[pam]` section
- SSSD ssh responder: `ca_db` in the `[ssh]` section

For details, see man page for `sssd.conf`.

Red Hat recommends keeping the default path and using a dedicated Certificate Authority certificate file for SSSD to make sure that only Certificate Authorities trusted for authentication are listed here.

### 6.3. INSTALLING TOOLS FOR MANAGING AND USING SMART CARDS

To configure your smart card, you need tools which can generate certificates and store them on a smart card.

You must:

- Install the `gnutls-utils` package, which helps you to manage certificates.
- Install the `opensc` package, which provides a set of libraries and utilities to work with smart cards.
- Start the `pcscd` service, which communicates with the smart card reader.

**Procedure**

1. Install the `opensc` and `gnutls-utils` packages:

   ```bash
   # dnf -y install opensc gnutls-utils
   ```

2. Start the `pcscd` service.

   ```bash
   # systemctl start pcscd
   ```

Verify that the `pcscd` service is up and running.

### 6.4. PREPARING YOUR SMART CARD AND UPLOADING YOUR CERTIFICATES AND KEYS TO YOUR SMART CARD
This section describes smart card configuration with the `pkcs15-init` tool, which helps you to configure:

- Erasing your smart card
- Setting new PINs and optional PIN Unblocking Keys (PUKs)
- Creating a new slot on the smart card
- Storing the certificate, private key, and public key in the slot
- If required, locking the smart card settings as certain smart cards require this type of finalization

**NOTE**

The `pkcs15-init` tool may not work with all smart cards. You must use the tools that work with the smart card you are using.

**Prerequisites**

- The `openssl` package, which includes the `pkcs15-init` tool, is installed. For details, see [Installing tools for managing and using smart cards](#).
- The card is inserted in the reader and connected to the computer.
- You have the private key, public key, and certificate to store on the smart card. In this procedure, `testuser.key`, `testuserpublic.key`, and `testuser.crt` are the names used for the private key, public key, and the certificate.
- You have your current smart card user PIN and Security Officer PIN (SO-PIN).

**Procedure**

1. Erase your smart card and authenticate yourself with your PIN:

   ```
   $ pkcs15-init --erase-card --use-default-transport-keys
   Using reader with a card: Reader name
   PIN [Security Officer PIN] required.
   Please enter PIN [Security Officer PIN]:
   
   The card has been erased.
   ```

2. Initialize your smart card, set your user PIN and PUK, and your Security Officer PIN and PUK:

   ```
   $ pkcs15-init --create-pkcs15 --use-default-transport-keys \
   --pin 963214 --puk 321478 --so-pin 65498714 --so-puk 784123
   Using reader with a card: Reader name
   
   The `pkcs15-init` tool creates a new slot on the smart card.
   ```

3. Set the label and the authentication ID for the slot:

   ```
   $ pkcs15-init --store-pin --label testuser \
   --auth-id 01 --so-pin 65498714 --pin 963214 --puk 321478
   Using reader with a card: Reader name
   ```
The label is set to a human-readable value, in this case, `testuser`. The `auth-id` must be two hexadecimal values, in this case it is set to `01`.

4. Store and label the private key in the new slot on the smart card:

```
$ pkcs15-init --store-private-key testuser.key --label testuser_key \
   --auth-id 01 --id 01 --pin 963214
```

Using reader with a card: `Reader name`

**NOTE**

The value you specify for `--id` must be the same when storing your private key and storing your certificate in the next step. Specifying your own value for `--id` is recommended as otherwise a more complicated value is calculated by the tool.

5. Store and label the certificate in the new slot on the smart card:

```
$ pkcs15-init --store-certificate testuser.crt --label testuser_crt \
   --auth-id 01 --id 01 --format pem --pin 963214
```

Using reader with a card: `Reader name`

6. (Optional) Store and label the public key in the new slot on the smart card:

```
$ pkcs15-init --store-public-key testuserpublic.key \
   --label testuserpublic_key --auth-id 01 --id 01 --pin 963214
```

Using reader with a card: `Reader name`

**NOTE**

If the public key corresponds to a private key or certificate, specify the same ID as the ID of the private key or certificate.

7. (Optional) Certain smart cards require you to finalize the card by locking the settings:

```
$ pkcs15-init -F
```

At this stage, your smart card includes the certificate, private key, and public key in the newly created slot. You have also created your user PIN and PUK and the Security Officer PIN and PUK.

### 6.5. Configuring SSH Access Using Smart Card Authentication

SSH connections require authentication. You can use a password or a certificate. This section describes the configuration necessary for enabling authentication using a certificate stored on a smart card.

For details about configuring smart cards with `authselect`, see Configuring smart cards using authselect.

**Prerequisites**

- The smart card contains your certificate and private key.
The card is inserted in the reader and connected to the computer.

SSSD is installed and configured.

Your username matches the Common Name (CN) or User ID (UID) in the certificate’s SUBJECT.

The pcscd service is running on your local machine.
For details, see Installing tools for managing and using smart cards.

Procedure

1. Create a new directory for SSH keys in the home directory of the user who uses smart card authentication:

   ```
   # mkdir /home/example.user/.ssh
   ```

2. Run the `ssh-keygen -D` command with the opensc library to retrieve the existing public key paired with the private key on the smart card, and add it to the `authorized_keys` list of the user’s SSH keys directory to enable SSH access with smart card authentication.

   ```
   # ssh-keygen -D /usr/lib64/pkcs11/opensc-pkcs11.so >>
   ~example.user/.ssh/authorized_keys
   ```

3. SSH requires access right configuration for the `.ssh` directory and the `authorized_keys` file. To set or change the access rights, enter:

   ```
   # chown -R example.user:example.user ~example.user/.ssh/
   # chmod 700 ~example.user/.ssh/
   # chmod 600 ~example.user/.ssh/authorized_keys
   ```

4. Optionally, display the keys:

   ```
   # cat ~example.user/.ssh/authorized_keys
   ```

   The terminal displays the keys.

5. Verify that the smart card authentication is enabled in the `/etc/sssd/sssd.conf` file:
   In the `[pam]` section, enable the pam certificate authentication module: `pam_cert_auth = True`

   If the `sssd.conf` file has not been created yet, you can create the minimal functional configuration by copying the following script to the command line:

   ```
   # cat > /etc/sssd/sssd.conf <<EOF
   [sssd]
   services = nss, pam
   domains = shadowutils
   
   [nss]
   
   [pam]
   pam_cert_auth = True
   EOF
   ```
To use the SSH keys, configure authentication with the **authselect** command:

```
# authselect select sssd with-smartcard --force
```

Now, you can verify the SSH access with the following command:

```
# ssh -l /usr/lib64/opensc-pkcs11.so -l example.user localhost hostname
```

If the configuration is successful, you are prompted to enter the smart card PIN.

The configuration works now locally. Now you can copy the public key and distribute it to **authorized_keys** files located on all servers on which you want to use SSH.
This section describes how to configure your smart card to achieve one of the following aims:

- Enable both password and smart card authentication
- Disable password and enable smart card authentication
- Enable lock on removal

**Prerequisites**

- Authselect installed
  The authselect tool configures user authentication on Linux hosts and you can use it to configure smart card authentication parameters. For details about authselect, see Configuring user authentication using authselect.

- Smart Card or USB devices supported by RHEL 9
  For details, see Smart Card support in RHEL9.

### 7.1. CERTIFICATES ELIGIBLE FOR SMART CARDS

Before you can configure a smart card with authselect, you must import a certificate into your card. You can use the following tools to generate the certificate:

- Active Directory (AD)
- Identity Management (IdM)
  For details about how to create IdM certificates, see Requesting a new user certificate and exporting it to the client.
- Red Hat Certificate System (RHCS)
  For details, see Managing Smart Cards with the Enterprise Security Client.
- Third-party Certification Authority (CA)
- Local Certification Authority. You can use a certificate generated by the Local Certification Authority if the user is not part of a domain or for testing purposes.
  For details about how to create and import local certificates into a smart card, Configuring and importing local certificates to a smart card.

### 7.2. CONFIGURE YOUR SYSTEM TO ENABLE BOTH SMART CARD AND PASSWORD AUTHENTICATION

This section describes how to enable both smart card and password authentication on your system.

**Prerequisites**

- The Smart card contains your certificate and private key.
- The card is inserted into the reader and connected to the computer.
- The authselect tool is installed on your system.
Procedure

- Enter the following command to allow smart card and password authentication:

```
# authselect select sssd with-smartcard --force
```

At this point, smart card authentication is enabled, however, password authentication will work if you forget your smart card at home.

### 7.3. Configuring Your System to Enforce Smart Card Authentication

The `authselect` tool enables you to configure smart card authentication on your system and to disable the default password authentication. The `authselect` command includes the following options:

- `with-smartcard` — enables smart card authentication in addition to password authentication
- `with-smartcard-required` — enables smart card authentication and disables password authentication

**NOTE**

The `with-smartcard-required` option only enforces exclusive smart card authentication for login services, such as `login`, `gdm`, `xdm`, `kdm`, `xscreensaver`, `gnome-screensaver`, and `kscreensaver`. Other services, such as `su` or `sudo` for switching users, do not use smart card authentication by default and will continue to prompt you for a password.

**Prerequisites**

- Smart card contains your certificate and private key.
- The card is inserted into the reader and connected to the computer.
- The `authselect` tool is installed on your local system.

**Procedure**

- Enter the following command to enforce smart card authentication:

```
# authselect select sssd with-smartcard with-smartcard-required --force
```

**NOTE**

Once you run this command, password authentication will no longer work and you can only log in with a smart card. Ensure smart card authentication is working before running this command or you may be locked out of your system.

### 7.4. Configuring Smart Card Authentication with Lock on Removal

The `authselect` service enables you to configure your smart card authentication to lock your screen instantly after removing the smart card from the reader. The `authselect` command must include the following variables:
• **with-smartcard** – enabling smart card authentication

• **with-smartcard-required** – enabling exclusive smart card authentication (authentication with a password is disabled)

• **with-smartcard-lock-on-removal** – enforcing log out after the smart card removal

**NOTE**

The **with-smartcard-lock-on-removal** option only works on systems with the GNOME desktop environment. If you are using a system that is **tty** or console based and you remove your smart card from its reader, you are not automatically locked out of the system.

**Prerequisites**

• Smart card contains your certificate and private key.

• The card is inserted into the reader and connected to the computer.

• The **authselect** tool is installed on your local system.

**Procedure**

• Enter the following command to enable smart card authentication, disable password authentication, and enforce lock on removal:

```
# authselect select sssd  with-smartcard  with-smartcard-required with-smartcard-lock-on-removal --force
```

Now, when you remove the card, the screen locks. You must re-insert your smart card to unlock it.
CHAPTER 8. AUTHENTICATING TO SUDO REMOTELY USING SMART CARDS

This section describes how to authenticate to sudo remotely using smart cards. After the `ssh-agent` service is running locally and can forward the `ssh-agent` socket to a remote machine, you can use the SSH authentication protocol in the sudo PAM module to authenticate users remotely.

After logging in locally using a smart card, you can log in through SSH to the remote machine and run the `sudo` command without being prompted for a password by using SSH forwarding of the smart card authentication.

For the purposes of this example, a client is connecting to the IPA server through SSH and running the `sudo` command on the IPA server with credentials stored on a smart card.

- Creating sudo rules in IdM
- Setting up the PAM module for sudo
- Connecting to sudo remotely using a smart card

8.1. CREATING SUDO RULES IN IDM

This procedure describes how to create sudo rules in IdM in order to give `ipau$er1` permission to run `sudo` on the remote host.

For the purposes of this example, the `less` and `whoami` commands are added as sudo commands to test the procedure.

Prerequisites

- The IdM user has been created. For the purpose of this example, the user is `ipau$er1`.
- You have the hostname of the system where you are running sudo remotely. For the purpose of this example, the host is `server.ipa.test`.

Procedure

1. Create a `sudo` rule named `adminrule` to allow a user to run commands.
   ```bash
   ipa sudorule-add adminrule
   ```

2. Add `less` and `whoami` as `sudo` commands:
   ```bash
   ipa sudocmd-add /usr/bin/less
   ipa sudocmd-add /usr/bin/whoami
   ```

3. Add the `less` and `whoami` commands to the `adminrule`:
   ```bash
   ipa sudorule-add-allow-command adminrule --sudocmds /usr/bin/less
   ipa sudorule-add-allow-command adminrule --sudocmds /usr/bin/whoami
   ```

4. Add the `ipau$er1` user to the `adminrule`:
   ```bash
   ```
ipa sudorule-add-user adminrule --users ipauser1

5. Add the host on which you are running sudo to the adminrule:

ipa sudorule-add-host adminrule --hosts server.ipa.test

Additional resources

- See `ipa sudorule-add --help`.
- See `ipa sudocmd-add --help`.

### 8.2. SETTING UP THE PAM MODULE FOR SUDO

This procedure describes how to install and set up the `pam_ssh_agent_auth.so` PAM module for sudo authentication with a smart card on any host where you are running sudo.

**Procedure**

1. Install the PAM SSH agent:

   ```bash
dnf -y install pam_ssh_agent_auth
   ```

2. Add the `authorized_keys_command` for `pam_ssh_agent_auth.so` to the `/etc/pam.d/sudo` file before any other `auth` entry:

   ```
   # %PAM-1.0
   auth sufficient pam_ssh_agent_auth.so
   authorized_keys_command=/usr/bin/sss_ssh_authorizedkeys
   auth include system-auth
   account include system-auth
   password include system-auth
   session include system-auth
   ```

3. To enable the SSH agent forwarding to work when you run sudo commands, add the following to the `/etc/sudoers` file:

   ```bash
   Defaults env_keep += "SSH_AUTH_SOCK"
   ```

   This allows users who have their public keys from smart cards stored in IPA/SSSD to authenticate to sudo without entering a password.

4. Restart the `sssd` service:

   ```bash
   systemctl restart sssd
   ```

Additional resources

- See the `pam` man page.

### 8.3. CONNECTING TO SUDO REMOTELY USING A SMART CARD
This procedure describes how to configure the SSH agent and client in order to connect to `sudo` remotely using a smart card.

**Prerequisites**

- You have created sudo rules in IdM.
- You have installed and set up the `pam_ssh_agent_auth` PAM module for sudo authentication on the remote system where you are going to run `sudo`.

**Procedure**

1. Start the SSH agent (if not already running).
   ```
   eval `ssh-agent`
   ```
2. Add your smart card to the SSH agent. Enter your PIN when prompted:
   ```
   ssh-add -s /usr/lib64/opensc-pkcs11.so
   ```
3. Connect via SSH with ssh-agent forwarding enabled (using the `-A` option) to the system where you are going to run `sudo` remotely:
   ```
   ssh -A ipauser1@server.ipa.test
   ```

**Verification steps**

- Run the `whoami` command with `sudo`:
  ```
  sudo /usr/bin/whoami
  ```

You should not be prompted for a PIN or password.
CHAPTER 9. TROUBLESHOOTING AUTHENTICATION WITH SMART CARDS

The following sections describe how to resolve some of the issues you might encounter when setting up smart card authentication.

- Testing smart card authentication
- Troubleshooting smart card authentication with SSSD
- Verifying that IdM Kerberos KDC can use PKINIT and that the CA certificates are correctly located
- Increasing SSSD timeouts
- Troubleshooting certificate mapping and matching rules

9.1. TESTING SMART CARD ACCESS ON THE SYSTEM

This procedure describes how to test whether you can access your smart card.

Prerequisites

- You have installed and configured your IdM Server and client for use with smart cards.
- You have installed the certutil tool from the nss-tools package.
- You have the PIN or password for your smart card.

Procedure

1. Using the lsusb command, verify that the smart card reader is visible to the operating system:

   $ lsusb
   Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
   Bus 001 Device 003: ID 072f:b100 Advanced Card Systems, Ltd ACR39U
   Bus 001 Device 002: ID 0627:0001 Adomax Technology Co., Ltd
   Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

   For more information on the smart cards and readers tested and supported in RHEL, see Smart Card support in RHEL 9.

2. Ensure that the pcsconf service and socket are enabled and running:

   $ systemclt status pcsconf.service pcsconf.socket
   - pcsconf.service - PC/SC Smart Card Daemon
     Loaded: loaded (/usr/lib/systemd/system/pcsconf.service; indirect; vendor preset: disabled)
     Active: active (running) since Fri 2021-09-24 11:05:04 CEST; 2 weeks 6 days ago
     TriggeredBy: ● pcsconf.socket
     Docs: man:pcsconf(8)
     Main PID: 3772184 (pcsconf)
3. Using the `p11-kit list-modules` command, display information about the configured smart card and the tokens present on the smart card:

```bash
$p11-kit list-modules
p11-kit-trust: p11-kit-trust.so
[...]
opensc: opensc-pkcs11.so
   library-description: OpenSC smartcard framework
   library-manufacturer: OpenSC Project
   library-version: 0.20
   token: MyEID (sctest)
      manufacturer: Aventra Ltd.
      model: PKCS#15
      serial-number: 8185043840990797
      firmware-version: 40.1
   flags:
      rng
      login-required
      user-pin-initialized
      token-initialized
```

4. Verify you can access the contents of your smart card:

```bash
$ pkcs11-tool --list-objects --login
Using slot 0 with a present token (0x0)
Logging in to "MyEID (sctest)".
Please enter User PIN:
Private Key Object; RSA
   label: Certificate
   ID: 01
   Usage: sign
   Access: sensitive
Public Key Object; RSA 2048 bits
   label: Public Key
   ID: 01
   Usage: verify
   Access: none
Certificate Object; type = X.509 cert
```
5. Display the contents of the certificate on your smart card using the `certutil` command:

   a. Run the following command to determine the correct name of your certificate:

   ```
   $ certutil -d /etc/pki/nssdb -L -h all
   Certificate Nickname                          Trust Attributes
   SSL,S/MIME,JAR/XPI
   Enter Password or Pin for "MyEID (sctest)":
   Smart Card CA 0f5019a8-7e65-46a1-afe5-8e17c256ae00 CT,C,C
   MyEID (sctest):Certificate u,u,u
   ```

   b. Display the contents of the certificate on your smart card:

   ```
   NOTE
   Ensure the name of the certificate is an exact match for the output displayed in the previous step, in this example `MyEID (sctest):Certificate`.
   ```

   ```
   $ certutil -d /etc/pki/nssdb -L -n "MyEID (sctest):Certificate"
   Enter Password or Pin for "MyEID (sctest)":
   Certificate:
   Data:
   Version: 3 (0x2)
   Serial Number: 15 (0xf)
   Signature Algorithm: PKCS #1 SHA-256 With RSA Encryption
   Issuer: "CN=Certificate Authority,O=IDM.EXAMPLE.COM"
   Validity:
   Not Before: Thu Sep 30 14:01:41 2021
   Not After : Sun Oct 01 14:01:41 2023
   Subject: "CN=idmuser1,O=IDM.EXAMPLE.COM"
   Subject Public Key Info:
   Public Key Algorithm: PKCS #1 RSA Encryption
   RSA Public Key:
   Modulus:
   [...]  
   Exponent: 65537 (0x10001)
   Signed Extensions:
   Name: Certificate Authority Key Identifier
   Key ID:
   2f:56:f9:1a
   Name: Authority Information Access
   Method: PKIX Online Certificate Status Protocol
   Location:
   URI: "http://ipa-ca.idm.example.com/ca/ocsp"
   Name: Certificate Key Usage
   ```
Critical: True
Usages: Digital Signature
Non-Repudiation
Key Encipherment
Data Encipherment

Name: Extended Key Usage
  TLS Web Server Authentication Certificate
  TLS Web Client Authentication Certificate

Name: CRL Distribution Points
Distribution point:
  URI: "http://ipa-ca.idm.example.com/ipa/crl/MasterCRL.bin"
  CRL issuer:
    Directory Name: "CN=Certificate Authority,O=ipaca"

Name: Certificate Subject Key ID
Data:
  be:07:1f:36

Signature Algorithm: PKCS #1 SHA-256 With RSA Encryption
Signature:
  [...]  
  Fingerprint (SHA-256):
  D:7B:C9:E0:EC
  Fingerprint (SHA1):

Mozilla-CA-Policy: false (attribute missing)
Certificate Trust Flags:
  SSL Flags:
    User
  Email Flags:
    User
  Object Signing Flags:
    User

Additional resources
  
  - See certutil(1) man page.

9.2. TROUBLESHOOTING SMART CARD AUTHENTICATION WITH SSSD

This procedure describes how to troubleshoot authentication with SSSD using smart cards.

Prerequisites
  
  - You have installed and configured your IdM Server and client for use with smart cards.
  - You have installed the sssd-tools package.
You are able to detect your smart card reader and display the contents of your smart card. See Testing smart card access on the system.

**Procedure**

1. Verify you can authenticate with your smart card using `su`:

   ```
   $ su - idmuser1 -c 'su - idmuser1 -c whoami'
   PIN for MyEID (sctest):
   idmuser1
   ```

   If you are not prompted for the smart card PIN, and either a password prompt or an authorization error are returned, check the SSSD logs. See Troubleshooting authentication with SSSD in IdM for information on logging in SSSD. The following is an example of an authentication failure:

   ```
   $ su - idmuser1 -c 'su - idmuser1 -c whoami'
   PIN for MyEID (sctest):
   su: Authentication failure
   ```

   If the SSSD logs indicate an issue from the `krb5_child`, similar to the following, you may have an issue with your CA certificates. To troubleshoot issues with certificates, see Verifying that IdM Kerberos KDC can use Pkinit and that the CA certificates are correctly located.

   ```
   [Pre-authentication failed: Failed to verify own certificate (depth 0): unable to get local issuer certificate: could not load the shared library]
   ```

   If the SSSD logs indicate a timeout either from `p11_child` or `krb5_child`, you may need to increase the SSSD timeouts and try authenticating again with your smart card. See Increasing SSSD timeouts for details on how to increase the timeouts.

2. Verify your GDM smart card authentication configuration is correct. A success message for PAM authentication should be returned as shown below:

   ```
   # sssctl user-checks -s gdm-smartcard "idmuser1" -a auth
   user: idmuser1
   action: auth
   service: gdm-smartcard
   
   SSSD nss user lookup result:
   - user name: idmuser1
   - user id: 603200210
   - group id: 603200210
   - gecos: idm user1
   - home directory: /home/idmuser1
   - shell: /bin/sh
   
   SSSD InfoPipe user lookup result:
   - name: idmuser1
   - uidNumber: 603200210
   - gidNumber: 603200210
   - gecos: idm user1
   - homeDirectory: /home/idmuser1
   - loginShell: /bin/sh
testing pam_authenticate

PIN for MyEID (sctest)
pam_authenticate for user [idmuser1]: Success

PAM Environment:
- PKCS11_LOGIN_TOKEN_NAME=MyEID (sctest)
- KRB5CCNAME=KCM:

If an authentication error, similar to the following, is returned, check the SSSD logs to try and determine what is causing the issue. See Troubleshooting authentication with SSSD in IdM for information on logging in SSSD.

pam_authenticate for user [idmuser1]: Authentication failure

PAM Environment:
- no env -

If PAM authentication continues to fail, clear your cache and run the command again.

# sssctl cache-remove
SSSD must not be running. Stop SSSD now? (yes/no) [yes] yes
Creating backup of local data...
Removing cache files...
SSSD needs to be running. Start SSSD now? (yes/no) [yes] yes

9.3. VERIFYING THAT IDM KERBEROS KDC CAN USE PKINIT AND THAT THE CA CERTIFICATES ARE CORRECTLY LOCATED

This procedure describes how to verify that IdM Kerberos KDC can use PKINIT and also describes how to verify your CA certificates are correctly located.

Prerequisites

- You have installed and configured your IdM Server and client for use with smart cards.
- You are able to detect your smart card reader and display the contents of your smart card. See Testing smart card access on the system.

Procedure

1. Run the kinit utility to authenticate as the idmuser1 with the certificate stored on your smart card:

   $ kinit -X X509_user_identity=PKCS11: idmuser1
   MyEID (sctest) PIN:

2. Enter your smart card PIN. If you are not prompted for your PIN, check that you can detect your smart card reader and display the contents of your smart card. See Testing smart card authentication.

3. If your PIN is accepted and you are then prompted for your password, you might be missing your CA signing certificate.
a. Verify the CA chain is listed in the default certificate bundle file using `openssl` commands:

```
$ openssl crl2pkcs7 -nocrl -certfile /var/lib/ipa-client/pki/ca-bundle.pem | openssl pkcs7 -print_certs -noout
subject=O = IDM.EXAMPLE.COM, CN = Certificate Authority
issuer=O = IDM.EXAMPLE.COM, CN = Certificate Authority
```

b. Verify the validity of your certificates:
   
i. Find the user authentication certificate ID for `idmuser1`:
   
   ```
   $ pkcs11-tool --list-objects --login
   [...]  
   Certificate Object; type = X.509 cert  
   label: Certificate  
   subject: DN: O=IDM.EXAMPLE.COM, CN=idmuser1  
   ID: 01
   ```
   
   ii. Read the user certificate information from the smart card in DER format:
   
   ```
   $ pkcs11-tool --read-object --id 01 --type cert --output-file cert.der
   Using slot 0 with a present token (0x0)
   ```
   
   iii. Convert the DER certificate to PEM format:
   
   ```
   $ openssl x509 -in cert.der -inform DER -out cert.pem -outform PEM
   ```
   
   iv. Verify the certificate has valid issuer signatures up to the CA:
   
   ```
   $ openssl verify -CAfile /var/lib/ipa-client/pki/ca-bundle.pem <path>/cert.pem
   cert.pem: OK
   ```
   
4. If your smart card contains several certificates, `kinit` might fail to choose the correct certificate for authentication. In this case, you need to specify the certificate ID as an argument to the `kinit` command using the `certid=<ID>` option.

   a. Check how many certificates are stored on the smart card and get the certificate ID for the one you are using:
   
   ```
   $ pkcs11-tool --list-objects --type cert --login
   Using slot 0 with a present token (0x0)
   Logging in to "MyEID (sctest)".
   Please enter User PIN:
   Certificate Object; type = X.509 cert  
   label: Certificate  
   subject: DN: O=IDM.EXAMPLE.COM, CN=idmuser1  
   ID: 01  
   Certificate Object; type = X.509 cert  
   label: Second certificate  
   subject: DN: O=IDM.EXAMPLE.COM, CN=ipauser1  
   ID: 02
   ```
   
   b. Run `kinit` with certificate ID 01:
5. Run `klist` to view the contents of the Kerberos credentials cache:

```
$ klist
Ticket cache: KCM:0:11485
Default principal: idmuser1@EXAMPLE.COM

Valid starting    Expires          Service principal
10/04/2021 10:50:04  10/05/2021 10:49:55  krbtgt/EXAMPLE.COM@EXAMPLE.COM
```

6. Destroy your active Kerberos tickets once you have finished:

```
$ kdestroy -A
```

Additional resources

- See `kinit` man page.
- See `kdestroy` man page.

### 9.4. INCREASING SSSD TIMEOUTS

If you are having issues authenticating with a smart card, check the `krb5_child.log` and the `p11_child.log` file for timeout entries similar to the following:

```
krb5_child: Timeout for child [9607] reached.....consider increasing value of krb5_auth_timeout.
```

If there is a timeout entry in the log file, try increasing the SSSD timeouts as outlined in this procedure.

**Prerequisites**

- You have configured your IdM Server and client for smart card authentication.

**Procedure**

1. Open the `sssd.conf` file on the IdM client:

   ```
   # vim /etc/sssd/sssd.conf
   ```

2. In your domain section, for example `[domain/idm.example.com]`, add the following option:

   ```
   krb5_auth_timeout = 60
   ```

3. In the `[pam]` section, add the following:

   ```
   p11_child_timeout = 60
   ```

4. Clear the SSSD cache:

   ```
   # sssctl cache-remove
   ```
SSSD must not be running. Stop SSSD now? [yes/no] [yes] yes
Creating backup of local data…
Removing cache files…
SSSD needs to be running. Start SSSD now? [yes/no] [yes] yes

Once you have increased the timeouts, try authenticating again using your smart card. See Testing smart card authentication for more details.

9.5. TROUBLESHOOTING CERTIFICATE MAPPING AND MATCHING RULES

If you are having issues authenticating with a smart card, check that you have linked your smart card certificate correctly to a user. By default, a certificate is associated with a user when the user entry contains the full certificate as part of the `usercertificate` attribute. However, if you have defined certificate mapping rules, you may have changed how certificates are associated with users. To troubleshoot certificate mapping and matching rules, refer to the following sections:

- Checking how the certificates are mapped to users
- Checking the user associated with a smart card certificate

NOTE

If you are using your smart card to authenticate using SSH, you need to add the full certificate to the user entry in Identity Management (IdM). If you are not using your smart card to authenticate using SSH, you can add certificate mapping data using the `ipa user-add-certmapdata` command.

9.5.1. Checking how the certificates are mapped to users

By default, a certificate is associated with a user when the user entry contains the full certificate as part of the `usercertificate` attribute. However, if you have defined certificate mapping rules, you may have changed how certificates are associated with users. This procedure describes how to check your certificate mapping rules.

**Prerequisites**

- You have installed and configured your Identity Management (IdM) server and client for use with smart cards.
- You are able to detect your smart card reader and display the contents of your smart card. See Testing smart card access on the system.
- You have mapped your smart card certificate to an IdM user. See Certificate mapping rules for configuring authentication on smart cards.

**Procedure**

1. Verify the certificate mapping rules currently configured for IdM:

   ```
   # ipa certmaprule-find
   1 Certificate Identity Mapping Rule matched
   ```
Rule name: smartcardrule
Mapping rule: (ipacertmapdata=X509:<I>{issuer_dn!nss_x500}<S>{subject_dn!nss_x500})
Matching rule: <ISSUER>CN=Certificate Authority,O=IDM.EXAMPLE.COM
Enabled: TRUE
----------------------------
Number of entries returned 1
----------------------------

You can expect to see one of the following mapping rules defined:

- **ipacertmapdata** indicates that the IdM user entry **certmapdata** attribute is used.
- **altSecurityIdentities** specifies that Active Directory’s user entry name mapping attribute is used.
- **userCertificate;binary=** indicates that the whole certificate in either IdM or AD is used.

You can define many matching options but some of the typically configured options are as follows:

- `<ISSUER>CN=[…]` specifies the issuer attribute of the certificate being used is checked to make sure it matches this.
- `<SUBJECT>.*,DC=MY,DC=DOMAIN` indicates the subject of the certificate is checked.

2. Enable System Security Services Daemon (SSSD) logging by adding `debug_level = 9` to the `/etc/sssd/sssd.conf` file on the IdM server:

```
[domain/idm.example.com]
...
debug_level = 9
```

3. Restart SSSD:

```
# systemctl restart sssd
```

4. You should see the following entry in the `/var/log/sssd/sssd_idm.example.com.log` file if the mapping is read correctly:

```
[be[idm.example.com]] [sdap_setup_certmap] (0x4000): Trying to add rule [smartcardrule][-1] [<ISSUER>CN=Certificate Authority,O=IDM.EXAMPLE.COM][((userCertificate;binary=
{certbin})(ipacertmapdata=X509:<I>{issuer_dn!nss_x500}<S>{subject_dn!nss_x500}))].
```

5. If your mapping rule contains an invalid syntax, an entry similar to the following can be seen in the log file:

```
[be[idm.example.com]] [sss_certmap_init] (0x0040): sss_certmap initialized.
[be[idm.example.com]] [ipa_certmap_parse_results] (0x4000): Trying to add rule [smartcardrule][-1] [<ISSUER>CN=Certificate Authority,O=IDM.EXAMPLE.COM][((ipacertmapdata=X509:<I>{issuer_dn!nss_x500}<S>{subject_dn!nss_x500})).]
[be[idm.example.com]] [parse_template] (0x0040): Parse template invalid.
[be[idm.example.com]] [parse ldap mapping rule] (0x0040): Failed to add template.
[be[idm.example.com]] [parse_mapping_rule] (0x0040): Failed to parse LDAP mapping rule.
[be[idm.example.com]] [ipa_certmap_parse_results] (0x0020): sss_certmap_add_rule failed for rule [smartcardrule], skipping. Please check for typos and if rule syntax is supported.
```
6. Check your mapping rule syntax.

```
# ipa certmaprule-show smartcardrule
Rule name: smartcardrule
Mapping rule: (|(userCertificate;binary={cert!bin})(ipacertmapdata=X509:<I>{issuer_dn!nss_x500}<S>{subject_dn!nss_x500}))
Matching rule: <ISSUER>CN=Certificate Authority,O=IDM.EXAMPLE.COM
Domain name: ipa.test
Enabled: TRUE
```

7. If required, modify your certificate mapping rule:

```
# ipa certmaprule-mod smartcardrule --maprule ’(ipacertmapdata=X509:<I>{issuer_dn!nss_x500}<S>{subject_dn!nss_x500})’
```

Additional resources

- See the `sss-certmap` man page.

9.5.2. Checking the user associated with a smart card certificate

If you are having issues authenticating with a smart card, verify the correct user is associated with your smart card certificate.

Prerequisites

- You have installed and configured your Identity Management (IdM) server and client for use with smart cards.

- You are able to detect your smart card reader and display the contents of your smart card. See Testing smart card access on the system.

- You have mapped your smart card certificate to an IdM user. See Certificate mapping rules for configuring authentication on smart cards.

- You have a copy of the certificate from your smart card in PEM format, for example, `cert.pem`.

Procedure

1. Verify the user is associated with your smart card certificate:

```
# ipa certmap-match cert.pem
---------
1 user matched
---------
Domain: IDM.EXAMPLE.COM
User logins: idmuser1
```
If the user or domain are not correct, check how your certificates are mapped to users. See [Checking how the certificates are mapped to users](#).

2. Check if the user entry contains the certificate:

   ```bash
   # ipa user-show idmuser1
   User login: idmuser1
   [...]  
   Certificate:MIIEejCCAuKgAwIBAgIBCzANBgkqhkiG9w0BAQsFADAzMREwDwYDVQQKDAhJ
   UEEuVEVTVEDeMBwGA1UEAwVQ2VydGlmaWNhdGUGQXV0aG9yaXR5MB4XD
   ```

3. If your user entry does not contain the certificate, add your base-64 encoded certificate to the user entry:

   a. Create an environment variable containing the certificate with the header and footer removed and concatenated into a single line, which is the format expected by the `ipa user-add-cert` command:

   ```bash
   $ export CERT=`openssl x509 -outform der -in idmuser1.crt | base64 -w0 -`
   
   Note that the certificate in the `idmuser1.crt` file must be in PEM format.

   b. Add the certificate to the profile of `idmuser1` using the `ipa user-add-cert` command:

   ```bash
   $ ipa user-add-cert idmuser1 --certificate=$CERT
   ```

   c. Clear the System Security Services Daemon (SSSD) cache.

   ```bash
   # sssctl cache-remove
   SSSD must not be running. Stop SSSD now? (yes/no) [yes] yes
   Creating backup of local data…
   Removing cache files…
   SSSD needs to be running. Start SSSD now? (yes/no) [yes] yes
   ```

4. Run `ipa certmap-match` again to confirm the user is associated with your smart card certificate.