Advanced Authentication 6.3

Device Service Installation Guide

December 2019

Legal Notices

Contents

About this Book

The Device Service Installation guide has been designed for users and describes the system requirements and installation procedure for Device Service. With Device Service you can use compliant devices, such as fingerprint readers, contact and contact-less cards, PKI smart cards, crypto sticks, and FIDO U2F tokens for enrollment on the Advanced Authentication Self-Service portal and for further authentication.

Intended Audience

This book provides information for individuals responsible for understanding administration concepts and implementing a secure, distributed administration model.

1 System Requirements

The following table provides information about the supported platforms for the Advanced Authentication Device Service:

	Microsoft Windows	Apple MacOS X	Linux
Card plug-in	x	x	x
Face plug-in	x	x	x
FIDO U2F plug-in	x	x	x
Fingerprint plug-in	x		
PKI plug-in	x	x	x
Bluetooth	x	x	x
Windows Hello plug-in	x		
Apple Touch ID		x	

Device Service for Windows supports the Card and PKI redirection to Remote Desktop and Citrix terminal sessions. You must install the Device Service on the terminal server to perform the redirection.

Device Service also supports virtual channel and you must install the Device Service on the both the terminal client and terminal server.

NOTE: Local administrator privileges for Windows and root privileges for Mac OS and Linux are required for installing and removing the Device Service.

For system requirements of Device Service, see Device Service.

NOTE: Advanced Authentication Device Service supports only Bluetooth. The BLE (Bluetooth Low Energy) is not supported. It is not recommended to use the Bluetooth feature on VMware virtual machines because a false authentication might happen when Bluetooth device is disabled or it is out of range.

For more information about the supported devices, see the following sections:

- Supported Card Readers and Cards
- Supported Devices for PKI
- Supported Fingerprint Readers

Supported Card Readers and Cards

Advanced Authentication stores the serial number of a card during enrollment and validates serial number later during the authentication.

The following table lists the supported card readers, smart cards, and unsupported card readers for Device Service:

Table 1-1

Device	Detail
Contactless card readers	◆ ACS ACR122 ◆ Broadcom Corp Contactless SmartCard ◆ Elatec RFID TWN3 ◆ HID OMNIKEY CardMan 5x25 ◆ HID OMNIKEY 5326 ◆ HID OMNIKEY 5x2x ◆ LEGIC LE-762-1N This device is supported on Microsoft Windows with Microsoft Visual C++ 2010 SP1 Redistributable Package installed and requires installation with specific parameters and disabling of other card plug-ins. The device is supported only when the parameter `card.smarfidManualMode` is set to `true`. ◆ LEGIC LM3000 This device is supported on Microsoft Windows with Microsoft Visual C++ 2010 SP1 Redistributable Package installed and requires installation with specific parameters and disabling of other card plug-ins. ◆ RF IDeas pcProx series ◆ NXP PR533
Contactless smart cards	◆ HID iClass series ◆ HID Prox series ◆ MIFARE Classic 1K/4K, Ultra Light, Ultra Light C, Plus ◆ MIFARE DESFIRE 0.6, MIFARE DESFIRE EV1, MIFARE SE, DESFire
Unsupported reader	LEGIC AIR ID series

Supported Devices for PKI

Advanced Authentication supports the certificate-based PKCS#11 contact smart cards and USB tokens (crypto sticks).

Device Service supports the following devices for PKI:

- Aladdin eToken PRO 32k/72k with SafeNet Authentication Client 9

- ruToken
- SafeNet Authentication eToken on the Mac OS.

To use PKI, specify a PKCS#11 module for your PKI device. For more information, see PKI Settings.

Ensure that the following requirements are met while using the used certificates:

1. Certificate must contain the Authority Information Access (AIA) and Certificate Revocation List (CRL) link to check the revocation status.
2. Certificate must contain a key pair: public and private key in the x509 format. The PKI service does not detect the certificates that do not comply with the requirements (are hidden during enrollment).

NOTE: The cards Cosmo polIC 64K V5.2 and Cyberflex Access 64K V1 SM 2.1 support the certificate-based enrollment only (key pair mode is not supported).

To enable the use of SafeNet Authentication eToken device (PKI) on Mac OS, perform the following steps:

1. Install the latest Device Service on Mac OS.
2. Install the `SafenetAuthenticationclient9.1.2.0.dmg` package.

   You can download SafeNet Authentication Client from Knowldege Symantec (https://knowledge.digicert.com/generalinformation/INFO1982.html) website.
3. Run the following commands to restart the Device Service:
   1. `sudo launchctl unload /Library/LaunchDaemons/com.netiq.deviceservice.plist`
   2. `sudo launchctl load /Library/LaunchDaemons/com.netiq.deviceservice.plist`
4. Connect the SafeNet Authentication eToken (PKI) to Mac OS workstation.

Supported Fingerprint Readers

Device Service supports the following methods that allow use of fingerprint readers:

- Fingerprint
- Windows Hello

Fingerprint

Fingerprint method supports the readers that has the capability of returning the fingerprint image beyond the boundaries of the chip.

Ensure that the system meets the following requirements for the WBF compliant readers:

- A reader must be visible in **Device Manager > Biometric devices**.
- The **Windows Biometric Service** must be active and set to `Automatic` in `services.msc`.

- The following policies must be enabled in **gpedit.msc > Computer Configuration > Administrative Templates > Windows Components > Biometrics**.
  - **Allow to use of biometrics**
  - **Allow users to log on using biometrics**
  - **Allow domain users to log on using biometrics**

The following table lists all the fingerprint readers that are supported and unsupported with respect to the Fingerprint method:

Supported Readers	Unsupported Readers
Lumidigm readers	**NOTE:** Fingerprint method does not support the readers that store the fingerprint and match fingerprints on a chip.
Digital Persona readers	SecuGen Hamster IV (HFDU04)
NEXT Biometrics NB-3010-UL	SecuGen Hamster (HFDU02R)
Precise Biometrics 100 X with AuthenTec AES2501B	Synaptics WBDI
Zvetco Verifi P2500 with AuthenTec AES2550	Futronic FS80, FS88
Zvetco Verifi P5100	Synaptics VFS7552
Zvetco Verifi P5200 with TouchChip Fingerprint Coprocessor	Microsoft Surface Pro type cover with the Fingerprint ID
Zvetco Verifi P6000	
Synaptic FP Sensors (WBF) (VID=138A, PID=0011)	
Synaptic FP Sensors (WBF) (VID=138A, PID=0017)	
Validity Sensor (VFS495) (VID=138A, PID=003F)	
Validity Sensors (WBF) (VID=138A, PID=0050)	
SecuGen Hamster Plus (HSDU03P)	
Green Bit DactyScan84c (Linux RHEL kernel 3.x.x)	
Nitgen eNBioScan-C1 (Linux RHEL kernel 3.x.x)	

To use fingerprint readers, you must configure some parameters manually. For more information, see Fingerprint Settings.

NOTE: You might face issues with matching the fingerprint while using the swipe readers. This is because of low quality sensors.

Windows Hello

The modern fingerprint readers do not return the fingerprint image outside the chip. However, store and match the images on the chip. Windows Hello method supports all the fingerprint readers and facial recognition devices that Microsoft Windows support. Microsoft Windows does not synchronize the fingerprints and faces between devices. Therefore, users can authenticate only on the devices where they enroll.

2 Installing and Upgrading Device Service

Before installing Device Service, ensure that you close all the web browsers. The installation procedure varies for different operating systems.

NOTE: You can find the Device Service component in the Advanced Authentication appliance distributive package.

To install and upgrade the Device Service based on the platform, see the following sections:

- “Installing Device Service on Windows” on page 13
- “Installing Device Service on Linux” on page 14
- “Upgrading Device Service on Linux” on page 16
- “Installing Device Service on Mac” on page 17

NOTE: After installing or upgrading the web browser, ensure to reinstall the Device Service.

WARNING: During the upgrade of Device Service on Apple Mac OS X and Linux, the configuration file is overwritten with a default one. Ensure that you have a copy of the file and put it back to the folder after the Device Service upgrade.

Installing Device Service on Windows

1 Run `naaf-deviceservice-x86-release-<version>.msi`.

IMPORTANT: For LEGIC readers, run the following command to install Device Service:

```
msiexec /i naaf-deviceservice-x86-release-<version>.msi TOKEN="XXX"
KEY="YYY"
```

where:

- XXX is token value (HEX <= 12 byte)
- YYY is 3Des Key (HEX 16 byte)

Device Service does not detect the LEGIC reader if keep the TOKEN/KEY parameter empty or specify invalid commands.

---

2 Click **Next**.

3 Accept the **Licence Agreement** and click **Next**.

4 Click **Next** to install on default folder or click **Change** to select different folder.

5 Click **Install**.

6 Click **Finish**.

NOTE: To upgrade Device Service on a Windows workstation that has a McAfee virus protection software installed, ensure to disable the McAfee protection. For more information about how to disable McAfee protection temporarily, see McAfee Support Community and Knowledge Center.

Installing Device Service on Linux

IMPORTANT: To use Device Service for FIDO U2F tokens, you must allow the FIDO U2F usage on Linux. For more information, see yubico FAQ.

You can install Device Service on Linux, based on your Linux distribution:

- "Installing Device Service on Ubuntu and Debian (deb package)" on page 14
- "Installing Device Service on openSUSE and SUSE" on page 15
- "Installing Device Service on Fedora, CentOS, RHEL" on page 15

Installing Device Service on Ubuntu and Debian (deb package)

Before installing the Device Service on Ubuntu and Debian, ensure to install the following necessary components:

NOTE: Before installing Device Service on Debian 10, switch to root account. Run the following command to switch to root account:

```
su -l
```

Set the root path and edit `/root/.bashrc` with the root privileges to add the following line:

```
export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Run all commands to install all components and Device Service on Debian 10 without the prefix `sudo`.

- For **Card** and **PKI** plug-in: Run the following command to install `libnss3-tools` component:

  ```
  sudo apt-get install libnss3-tools
  ```
- For **HID OMNIKEY** reader: Run the following command to install `pcscd` component:

  ```
  sudo apt-get install pcscd
  ```
- For **Bluetooth** plug-in: Run the following command to install `bluez` component:

  ```
  sudo apt-get install bluez
  ```

Run the following command to install the Device Service on Ubuntu and Debian:

```
sudo dpkg -i naaf-deviceservice-linux64-release-<version>.deb
```

Installing Device Service on openSUSE and SUSE

Before installing the Device Service on openSUSE and SUSE, ensure to install the following necessary components:

- For **Card** and **PKI** plug-in: Run the following command to install `libpcsclite1` and `nss-tools` component:

  ```
  sudo zypper install libpcsclite1 sudo zypper install mozilla-nss-tools
  ```

- For **RF IDeas** card reader: Install the `libudev.so.0` library manually. Run the following command to link the `libudev.so.1` to `libudev.so.0`:

  ```
  sudo ln -s <location_of_libudev1>libudev.so.1
  <location_of_libudev1>libudev.so.0
  ```

- For **Bluetooth** plug-in: Run the following command to install `bluez` component:

  ```
  sudo zypper install bluez
  ```

Run the following command to install the Device Service on openSUSE and SUSE:

```
sudo rpm -i naaf-deviceservice-linux64-release-<version>.rpm
```

NOTE: While installing the Device Service on SUSE operating system, there may be dependency issues related to the `pcsc-lite` package. Therefore, you must install the required package with **zypper install pcsc-lite** and initiate the Device Service installation again.

Installing Device Service on Fedora, CentOS, RHEL

Before installing the Device Service on Fedora, CentOS, and RHEL, ensure to install the following necessary components:

- For **Card** and **PKI** plug-in: Run the following command to install `nss-tools` component:

  ```
  sudo yum install nss-tools
  ```

- For **RF IDeas** card reader: Install the `libudev.so.0` library manually. Run the following command to link the `libudev.so.1` to `libudev.so.0`:

  ```
  sudo ln -s <location_of_libudev1>libudev.so.1
  <location_of_libudev1>libudev.so.0
  ```

- For **Bluetooth** plug-in: Run the following command to install `bluez` component:

  ```
  sudo yum install bluez
  ```

Run the following command to install the Device Service on Fedora, CentoOS, and RHEL:

```
sudo rpm -Uvh naaf-deviceservice-linux64-release-<version>.rpm
```

Run the following command to install the Device Service on Fedora, CentoOS, and RHEL without any dependencies:

```
sudo rpm -i --nodeps naaf-deviceservice-linux64-release<version>.rpm
```

NOTE: While installing the Device Service on CentOS or RHEL operating system, there may be dependency issues related to the `pcsc-lite` package. Therefore, you must install the required package with **yum install pcsc-lite** and initiate the Device Service installation again.

Upgrading Device Service on Linux

You can upgrade Device Service on Linux, based on your Linux distribution:

- "Upgrading Device Service on Ubuntu and Debian (deb package)" on page 16
- "Upgrading Device Service on openSUSE (rpm package)" on page 16
- "Upgrading Device Service on Fedora (rpm package)" on page 16

Upgrading Device Service on Ubuntu and Debian (deb package)

To upgrade the Device Service 5.6 or later, perform the following steps to remove the old package and install a new package:

1. Run the following command to remove the existing Device Service package from the machine:
   ```
   sudo apt-get remove deviceservice-<version>.x86_64
   ```
2. (Optional) Run the following command to install `bluez` component:
   ```
   sudo apt-get install bluez
   ```
3. Run the following command to install the Device Service package:
   ```
   sudo dpkg -i naaf-deviceservice-linux64-release-<version>.deb
   ```

Upgrading Device Service on openSUSE (rpm package)

1. Run the following command to remove the existing Device Service packager from the machine:
   ```
   sudo rpm -e deviceservice-<version>.x86_64
   ```
2. (Optional) Run the following command to install `bluez` component:
   ```
   sudo zypper install bluez
   ```
3. Run the following command to install the Device Service package:
   ```
   sudo rpm -i naaf-deviceservice-linux64-release-<version>.rpm
   ```

Upgrading Device Service on Fedora (rpm package)

1. Run the following command to remove the existing Device Service packager from the machine:
   ```
   sudo rpm -e deviceservice-<version>.x86_64
   ```
2. (Optional) Run the following command to install `bluez` component:
   ```
   sudo yum install bluez
   ```
3. Run the following command to install the Device Service package:
   ```
   sudo rpm -Uvh naaf-deviceservice-linux64-release-<version>.rpm
   ```

Installing Device Service on Mac

1. Double click the file `naaf-deviceservice-macos-release-<version>.dmg`.
2. The `naaf-deviceservice.pkg` and `uninstall` files are displayed.
3. Double click the file `naaf-deviceservice.pkg`.
4. Click **Continue**.
5. Read and accept the license agreement.
6. Select the disk where you want to install Device Service and click **Continue**.
7. Click **Install.**

   A prompt to specify the local administrator credentials is displayed.
8. Specify **User name** and **Password**.
9. Click **Install Software.**

   While installing the Device Service in Mac OS Catalina, the user is prompted with a message `"DeviceServiceTool" would like to receive keystrokes from any application.` If you use the RF Ideas reader, click **Open System Preferences** and grant access to this application in Security & Privacy - Privacy preferences. If you don't use RF Ideas reader, click **Deny**.

3 Configuring Device Service

After installing the Device Service, you must configure few parameters in the configuration file of Device Service to enable the use of devices on your workstation.

WARNING: During the upgrade of Device Service on Apple Mac OS X and Linux, the configuration file is overwritten with the default settings. Ensure that you have a copy of the file and replace the file after the upgrade.

NOTE: In the `host.ports` parameter, the supported ports are 8440, 8441, and 8442.

This chapter contains the following configurations:

Apple Touch ID

This section contains Configuring the Apple Touch ID:

Configuring the Apple Touch ID

You can configure Apple Touch ID timeout by performing the following steps:

1. Open `/Library/Application\Support/NetIQ/DeviceService.app/Contents/Resources/config.properties.`
2. Set the parameter `touchid.timeout` (Default value is 30).

   For example, if you want to set the timeout value to 10 seconds, set `touchid.timeout` to 10.
3. Save the changes.
4. Restart the operating system.

Card Settings

Advanced Authentication supports the Microsoft policy Interactive logon: Smart card removal behavior, which allows you to select an action on a card event. You can configure it to perform a force log off or lock a user session when a user presents card to the reader.

This section contains the following configurations:

Configuring the Card Settings

To use LEGIC LM3000 or LEGIC LE-762-1N readers, you must disable the other card plug-ins to avoid conflicts.To do this, perform the following steps:

NOTE: The LEGIC and RF IDeas readers are not supported on Linux and Mac operating systems.

1 Open the following configuration file for respective OS
   - In Microsoft Windows, open `C:\ProgramData\NetIQ\Device Service\config.properties`.
   - In Linux, open `/opt/NetIQ/Device Service/config.properties`.
   - In Apple Mac OS X, for 6.3 Service Pack 1 and newer versions, open `/Library/Application\ Support/NetIQ/DeviceService.app/Contents/Resources/config.properties`. For prior versions, open `/Library/LaunchDaemons/NetIQ/Device Service/config.properties`.
2 Set the preferred parameters based on the card reader:

Parameter	Description
`card.omnikeyEnabled`	Used for the omnikey type of readers. The default value is `true`. Set the value to `false` to disable the usage of the device.
`card.rfideasEnabled`	Used for the RF IDeas readers. The default value is `false`. Set the value to `true` to enable the usage of the device.
`card.rfideas.productType`	Used for RF IDeas readers. The possible values are `prox`, `sonar`, or `swipe`, or all. You can combine them as `prox;sonar;swipe`. The default value is `prox`.
`card.rfideas.deviceType`	The possible values are `usb`, `serial`, or `tcp`, or `all`. You cannot combine them. The default value is `usb`.

Parameter	Description
`card.forceVirtualChannels`	Used for RF IDeas readers to work in a terminal session. If you set `card.forceVirtualChannels` to `true`, the Device Service uses its own mechanism for card redirection through the virtual channels. You must install the Device Service on both the terminal server and terminal client. The default value is `false`.
`card.smarfidEnabled`	Used for the smarfid type of readers. The default value is `false`. Set the value to `true` to enable the usage of the device.
`card.smarfidManualMode`	Used for the smarfid card behavior. If you set `card.smarfidManualMode to false` or when the parameter is not available in the `config.properties` file, the reader's LED is in blue (read mode) by default and starts to blink when you place a card on the reader. If you set `card.smarfidManualMode` to `true`, the reader's LED is in green (ready mode) by default and does not blink when you place a card on the reader. The reader blinks only if you are in the Login or Unlock screen and Windows Client requests to place a card. You must disable the `1:N` functionality to disable auto-waiting of a card for the Login or Unlock screen. For more information about how to disable 1:N, see Disabling 1:N. You must disable the **Interactive logon: Smart card removal behavior** policy to disable the auto-waiting of a card when a user is logged in. For more information about how to disable **Smart card removal behavior** policy, see the Microsoft documentation. You can use the feature only for LEGIC readers.
`card.smarfidManualBeepEnabled`	Used for generating beeps from a supported LEGIC reader when you put a card on it. The default value of the parameter is `false` and the beeps are muted. Set `card.smarfidManualBeepEnabled` to `true` for this. You can use this option only when the manual mode is enabled (`card.smarfidManualMode=true`).
`card.isCardIdGenerated`	The feature can be used only for LEGIC readers. Used to generate a new card identifier during enrollment. and during each enrollment, the card identifier is not changed. The default value is `false`.
`card.desfireEnabled`	Used for the desfire type of readers. The default value is `true`. Set the value to `false` to disable the usage of the device.

3 Save the changes.

4 Restart the Device Service.

Configuring the Virtual Machine for Working of the RF IDeas Readers

You must perform the following configuration steps to ensure that the RF IDeas reader work with the VMware Mac virtual machine.

1 Add the following lines to the .vmx file of the virtual machine.

```
usb.generic.allowHID=true
usb.generic.allowLastHID=true
```

2 For 6.3 Service Pack 1 and newer versions, open `/Library/Application\ Support/NetIQ/DeviceService.app/Contents/Resources/config.properties`. For prior versions, open `/Library/LaunchDaemons/NetIQ/Device Service/config.properties`, and set the parameter `card.rfideasEnabled` to `true`.

You must perform the following configuration steps to ensure that the RF IDeas reader work with the VMware Windows virtual machine.

1 Add the following lines to the .vmx file of the virtual machine.

```
usb.generic.allowHID=true
usb.generic.allowLastHID=true
```

If the above does not achieve the redirection, go to step 2.

2 Go to the following url: http://kb.vmware.com/kb/1011600.

The VID (Vendor ID) and PID (Product ID) of the connected reader found in the Device Manager are generally listed as: `VID_0C27&PID_3BFA`. To ensure the VID and PID are included in the list, add the following to the registry:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\VMware, Inc.\VMwareVDM\USB]
AllowHardwareIDs=[REG_MULTI_SZ]"VID_0C27&PID_3BFA"
```

3 Set the following in the configuration file `C:\ProgramData\NetIQ\Device Servie\config.properties`

```
card.rfideasEnabled:true
```

Device Authentication Setting

The Trusted Platform Module (TPM) is a crypto-processor available in Windows workstation to achieve actions, such as generating, storing, and limiting the use of cryptographic keys. During the Device Authentication method enrollment, a key pair is generated and stored in the TPM chip. The stored key pair is verified to authenticate users.

By default, the TPM is enabled in Windows workstation. However, in some Windows workstation TPM chip is not available then you can store the generated key pair in the Local Security Authority (LSA) and encrypt the same using PIN.

To disable the TPM chip and allow Device Authentication enrollment in the generate key pair mode perform the following:

1 Open the configuration file `C:\ProgramData\NetIQ\Device Service\config.properties`.

If the file does not exist, create a new file.

2 Specify `deviceAuth.tpmEnabled: false`.

The default value is `True`.

3 Save the configuration.

NOTE: This setting is not required in Device Service for Linux and Mac because the TPM mode is not supported on these platforms. However, the non-TPM mode always used on these platforms.

Facial Recognition

When the user authenticates through Facial Recognition method, Advanced Authentication can use blink detection to differentiate live face and photos. You can configure Device Service to enable blink detection. So, the user needs to blink several times, depending on the service settings to get authenticated.

1 Open the following configuration file for respective OS

- In Microsoft Windows, open `C:\ProgramData\NetIQ\Device Service\config.properties`.
- In Linux, open `/opt/NetIQ/DeviceService/config.properties`.
- In Apple Mac OS X, open `/Library/Application\ Support/NetIQ/DeviceService.app/Contents/Resources/config.properties`.

2 Specify the following parameters:

Parameter	Description
`video.checkByBlinking`	Set this parameter to `True` to enable blink detection.
`video.blinkThreshold`	To specify the eye aspect ratio. The Eye Aspect Ratio is an estimate of the eye opening state. Eye ratio below the specified value will be counted as a blink. The default value is 0.2. The eye aspect ration of closed eye is between 0.08 - 0.2 and eye aspect ration of open eye is between 0.2 up to 0.32.
`video.blinkFrames`	To specify the number of consequent frames with the ratio below the threshold. If the eye ratio is below the threshold within the given number of consequent frames, that counted as one blink. The default value is 2. If the blink frame value is 2, make sure the eye aspect ratio in 2 consequent frames is lesser than the value specified in the blink threshold.
`video.blinkCount`	To specify the number of blinks needed to authenticate.

Parameter	Description
`video.deviceId`	Specify the value to select the camera. The default value is `0`. If you set the parameter to `0`, the Device Service picks the default camera. If you set the parameter to `1`, the Device Service selects the secondary camera. If you set the parameter to `2`, the Device Service selects the third camera and so on if you have many.

3 Save the changes.

4 Restart Device Service.

Fingerprint Settings

The following table describes the fingerprint modes that must be configured while using a specific fingerprint reader. Using the parameter `fingerprint.mode`, you can either configure a single or multiple fingerprint readers mode.

Mode Parameter	Description
fingerprint.mode: 1	To use the WBF API mode. In this mode, Advanced Authentication works with a processed fingerprint reader in Windows Biometric Framework API.
fingerprint.mode: 2	To use the WBF Direct mode. In this mode, Advanced Authentication works directly with a device driver. **NOTE:** Some WBF compliant readers may work only in the WBF Direct mode, for example, the NEXT Biometrics readers. You can download the NEXT Biometrics driver from the link.
fingerprint.mode: 3	To use the Lumidigm mode. You must install the Lumidigm Driver. You can download the driver from the HID Global website. Some devices require that the Lumidigm Device Service is installed.
fingerprint.mode: 4	To use the DigitalPersona mode. You must install the DigitalPersona U.are.U RTE. You can download it from the DigitalPersona website. **NOTE:** For compatibility between DigitalPersona RTE v3.x and old device like DigitalPersona U.are.U 4500 install the RTE 3.2 and later version without Digital Persona authentication service. During the setup, select **Custom setup** and then remove the authentication service feature.

Mode Parameter	Description
fingerprint.mode: 5	To use the Green Bit DactyScan84c (multi-finger reader) reader. This mode is supported only on Linux RHEL kernel 3.x.x. **Prerequisites**: Before using the Green Bit DactyScan84c reader, you must install the eNBioScan-C1 Drivers. Ensure to save the following SO files to the `/lib64` path of Linux workstation: ◆ **Nitgen eNBioScan-C1**: `libfp_device.so`, `libfp_device.so.1.0.0`, `libFPLib.so`, `libFPLib.so.6.0.1.9`, `libNBioBSPISO4JNI.so`, `libNBioBSPJNI.so`, `libNBioBSP.so`, `libNExportRawToISO.so`, `libvhm.so`, `libvhm.so.6.1.4.4`. ◆ **Green Bit DactyScan84c**: `libAN2K_LIB.so`, `libBozorth.so`, `libDID20IP.so`, `libDID20.so`, `libDS40u.so`, `libDS84C.so`, `libDS84t.so`, `libDSBeep.so`, `libGBFINIMG.so`, `libGBFIR.so`, `libGBImgTran.so`, `libGBJPEG.so`, `libGBMSAPI.so`, `libGBNFIQ2.so`, `libGBNFIQ.so`, `libLfsConv.so`, `libLfs.so`, `libMC517.so`, `libMS527.so`, `libMS527t.so`, `libopencv_core.so`, `libopencv_imgproc.so`, `libopencv_ml.so`, `libqsqlite.so`, `libqtaudio_alsa.so`, `libqtmedia_pulse.so`, `libqxcb.so`, `libusb1.0.20gb.so`, `libVsRoll.so`, `libWSQPack.so`.
fingerprint.mode: 6	To use the eNBioScan-C1 reader. You must install the eNBioScan-C1 Drivers. This mode is supported only on Linux RHEL kernel 3.x.x.
fingerprint.wsqBitrate	This Wavelet Scalar Quantization (WSQ) algorithm based parameter determines the amount of compression. The 2.25 value yields around 5:1 compression and the 0.75 value yields around 15:1 compression. The default value is 2.25. **NOTE:** As the service uses NBIS library, only Device Service for Windows supports the parameter.

Configuring Multiple Fingerprint Reader Modes

Device Service supports multiple fingerprint reader modes. You can configure multiple modes in one of the following ways:

- Specify numeric values assigned to each mode.

  For example: **fingerprint.mode: 1,2,3** to use WBF API, WBF Direct, and Lumidigm modes.
- Specify the mode names.

  For example: **fingerprint.mode: WbfDirect,DigitalPersona** to use WBF Direct and DigitalPersona modes.
- Specify the combination of numeric value and mode name.

  For example: **fingerprint.mode:1,WbfDirect,3** to use WBF API, WBF Direct, and Lumidigm modes.

NOTE: The `fingerprint.mode: auto` is the default mode which enables Lumidigm, DigitalPersona, and WbfDirect modes.

Configuring the Fingerprint Settings

1 Open the configuration file based on your platform:

- **Microsoft Windows**: `C:\ProgramData\NetIQ\Device Service\config.properties`.
- **Linux**: `/opt/NetIQ/Device Service/config.properties`.
- **Apple Mac OS X:** Fingerprint readers are not supported.

2 Specify the following parameters:

- `fingerprint.multifingerDevice` to configure the type of fingerprint device in use.

  Set `fingerprint.multifingerDevice: false` (default value) to use single finger readers such as Lumidigm, DigitalPersona, and so on.

  Set `fingerprint.multifingerDevice: true` to use the Green Bit DactyScan84c multi-finger reader.

- `fingerprint.mode` to configure fingerprint reader mode.

  Set `fingerprint.mode`: 3 to use the Lumidigm reader mode only.

  Set `fingerprint.mode: 1,WbfDirect,` 3 to use more than one reader modes, WBF API, WBF Direct, and Lumidigm.

  For example, to enable three single finger readers: Lumidigm, DigitalPersona, and WbfDirect, the parameters must be configured as follows:

  ```
  fingerprint.mode: auto
  ```

  To use a multi-finger device, the parameters must be configured as follows:

  ```
  fingerprint.multifingerDevice: true
  fingerprint.mode: 5
  ```

3 (Optional) Specify the following parameter to set the capture inactive time in seconds:

```
fingerprint.captureTimeout: 15
```

NOTE: The parameters are case-sensitive.

4 (Optional) Specify the following parameter to enable the DigitalPersona readers to work with the other services along with Device Service:

```
fingerprint.dp.cooperativeMode=true
```

The default value is `true`. You can set the value to `false` to stop the DigitalPersona with the other services.

5 Save the changes.

6 Restart the Device Service.

NOTE: The parameter `fingerprint.isoSupported: true` (default value is `true`) enables Device Service to extract ISO from raw image that is received when a user scans fingerprints during authentication. This parameter helps to remove this additional step on the server and improves the authentication speed.

If you set the parameter as `false`, Device Service sends a raw image to the Advanced Authentication server and the server extracts ISO to match the fingerprints with a stored authenticator. This may cause performance issues in environments where hundreds of users perform fingerprint authentication at the same time.

PKI Settings

This section contains the following configurations:

- Configuring the PKI Device
- Configuring e-Token PRO
- Configuring the YubiKey PKI
- Configuring OpenSC
- Configuring Gemalto Smart Card with Advanced Authentication

Configuring the PKI Device

To use PKI, you must specify a PKCS#11 module for your PKI device. To do this, perform the following steps:

1. Open the configuration file based on the operating system:
   - **Microsoft Windows**: `C:\ProgramData\NetIQ\Device Service\config.properties.`
   - **Linux**: `/opt/NetIQ/Device Service/config.properties`.
   - **Apple Mac OS X**: For 6.3 Service Pack 1 and newer versions, open `/Library/Application\ Support/NetIQ/DeviceService.app/Contents/Resources/config.properties`. For prior versions, open `/Library/LaunchDaemons/NetIQ/Device Service/config.properties`.
2. Remove the hash sign(`#`) before `vendorModule` to remove any comments from the parameter.
3. Set the vendor module specific dll file name to the parameter.

   `pki.vendorModule: <filename>.dll`

   For example, `pki.vendorModule: rtPKCS11.dll`.

   NOTE: You can specify more than one PKCS#11 library with semicolon in the format: `pki.vendorModule: eToken.dll;rtPKCS11.dll`

   If a vendor module is not located in the **system32** directory, use `\\` to specify the path. If there are any spaces in the path, ensure not to replace the space with `\\` in the path.

   For example, `pki.vendorModule: C:\\Program Files\\ActivIdentity\\ActivClient\\acpkcs211.dll.`

NOTE: If you have specified some `pki.vendorModules` separated by a semicolon, you must specify the same number of values for the parameter `pki.blockingMode`.

For example, `pki.blockingMode: true;false.`

PKI plug-in of the Device Service supports the automatic mode, where a few known vendor modules are auto-detected. You must specify: `pki.vendorModule: auto`.

4 (Optional) Specify the additional parameters:

Table 3-1

Method	Syntax	Description
Hash	`pki.hashMethod: SHA256`	The default value is SHA256 and you can specify this value, if a parameter is not presented. The following methods are also supported: SHA224, SHA384, SHA512. To set the methods, ensure that the PKCS#11 module supports the required hash method.
Padding	`pki.padding: PKCS#1`	The default value is PKCS#1 and you can specify this value, if a parameter is not presented.The following options are also supported: PSS, OAEP.
Key size	`pki.modulusBits: 2048`	The default value is 2048 bit. For example, eToken PRO 32k does not support it and you need to set 1024 to use it.
Blocking mode	`pki.blockingMode: true`	This parameter is used to detect and monitor the token connected to your system. It is set to `true` by default. OpenSC does not support the 'waiting for card' mechanism and it requires to change the option to `False`. Most of the vendors module work appropriately in the default mode.

NOTE: If you specify the `pki.vendorModule: auto` and `pki.blockingMode` parameters, the `pki.blockingMode` parameter does not overwrite a blocking mode that is pre-defined for an auto-detectable vendor module.

5 Save the changes.

6 Restart the Device Service.

Configuring e-Token PRO

1 Navigate to one of the following paths and open the configuration file based on the operating system:

- Microsoft Windows: `C:\ProgramData\NetIQ\Device Service\config.properties.`

- **Linux**: `/opt/NetIQ/Device Service/config.properties`.
- **Apple Mac OS X**: For 6.3 Service Pack 1 and newer versions, open `/Library/Application\ Support/NetIQ/DeviceService.app/Contents/Resources/config.properties`. For prior versions, open `/Library/LaunchDaemons/NetIQ/Device Service/config.properties`.

2 Remove the hash sign(#) before `vendorModule` to remove any comments from the parameter.

3 Set the vendor module specific dll file name to the parameter based on the operating system:

- **Microsoft Windows**:
  - `pki.vendorModule: eToken.dll`
  - `pki.blockingMode: true`
- **Linux**:
  - `pki.vendorModule: /usr/lib/libeTPkcs11.so`
  - `pki.blockingMode: true`
- **Mac OS X**:
  - `pki.vendorModule: libeTPkcs11.dylib`
  - `pki.blockingMode: true`

4 Save the changes.

5 Restart the Device Service.

Configuring the YubiKey PKI

Before configuring the YubiKey PKI, ensure to download the Yubico PIV (https://developers.yubico.com/yubico-piv-tool/Releases/) tools. You can unpack the zip file and navigate to `bin` directory.

To configure the PIV compliant Yubikey for public key authentication with OpenSC through PKCS11, perform the following steps:

1 Open the configuration file based on the operating system:

- **Microsoft Windows**: `C:\ProgramData\NetIQ\Device Service\config.properties`.
- **Linux**: `/opt/NetIQ/Device Service/config.properties`.
- **Apple Mac OS X**: For 6.3 Service Pack 1 and newer versions, open `/Library/Application\ Support/NetIQ/DeviceService.app/Contents/Resources/config.properties`. For prior versions, open `/Library/LaunchDaemons/NetIQ/Device Service/config.properties`.

2 Add hash symbol (#) as a prefix to the existing parameters that start with pki to set the parameter as a comment.

For example:

- `#pki.vendorModule=auto`
- `#pki.forceVirtualChannels=false`

3 Add the following parameter specific to the operating system:

- **Microsoft Windows**:
  - `pki.vendorModule=libykcs11-1.dll`
  - `pki.blockingMode=false`
- **Linux**:
  - `pki.vendorModule=/usr/local/lib/libykcs11.so`
  - `pki.blockingMode=false`
- **Mac OS X**:
  - `pki.vendorModule=/usr/lib/Libykcs11.1.dylib`
  - `pki.blockingMode=false`

4 Save the changes.

5 Perform one of following based on the operating system:

- **Microsoft Windows**: Open the Services app and restart the Device Service.
- **Linux**: Run the following commands:

  ```
  sudo service deviceservice stop
  sudo service deviceservice start
  ```

- **Mac OS X**: Run the following commands:

  ```
  sudo launchctl unload /Library/LaunchDaemons/
  com.netiq.deviceservice.plist
  sudo launchctl load /Library/LaunchDaemons/
  com.netiq.deviceservice.plist
  ```

IMPORTANT: The YubiKey PKCS module supports only the **Generate a key pair** mode and does not work with the existing certificates on the PKI token or smart card.

NOTE: If you are not able to enroll the PKI method using YubiKey PKI or import a certificate to YubiKey token, see to resolve these issues.

NOTE: ◆Sometimes the vendor specific module may stop working on Mac OS.

- Some certificates may not be accessible through the vendor specific module. The issue with certificate may display an error message `Operation failed exception`. This issue occurs when the vendor module does not retrieve the certificate body for some certificates.

Configuring OpenSC

OpenSC is a third party software that provides a set of libraries and utilities to work with different PKCS#11 tokens and cards. OpenSC implements the standard APIs to smart cards and tokens if these devices do not have the vendor specific PKCS module.

Before configuring the OpenSC on any PKCS#11 based tokens and cards, ensure that the following requirements are met:

- Download and install OpenSC (https://github.com/OpenSC/OpenSC/releases/).

  NOTE: For Microsoft Windows, you must install and use a 32bit version of OpenSC.

- Import a certificate to the token or card.

To configure token for public key authentication with OpenSC through PKCS11, perform the following steps:

1. Open the OpenSC configuration file based on the operating system:
   - **Microsoft Windows**: `c:\Program Files (x86)\OpenSC Project\OpenSC\opensc.conf`
   - **Linux**: `/usr/local/etc/opensc.conf`
   - **Apple Mac OS X**: `/Library/OpenSC/etc/opensc.conf`
2. Remove the hash symbol from following parameter to uncomment:

   `pin_cache_ignore_user_consent = true;`

   You can also see the following comments in the configuration file:

   # Older PKCS#11 applications not supporting CKA_ALWAYS_AUTHENTICATE

   # may need to set this to get signatures to work with some cards.

   # Default: false
3. Open the configuration file based on the operating system:
   - **Microsoft Windows**: `C:\ProgramData\NetIQ\Device Service\config.properties`
   - **Linux**: `/opt/NetIQ/Device Service/config.properties`
   - **Apple Mac OS X**: For 6.3 Service Pack 1 and newer versions, open `/Library/Application\ Support/NetIQ/DeviceService.app/Contents/Resources/config.properties`. For prior versions, open `/Library/LaunchDaemons/NetIQ/Device Service/config.properties`.
4. Add the following parameters specific to the operating system:
   - **Microsoft Windows**:
     - `pki.vendorModule=C:\\Program Files (x86)\\OpenSC Project\\OpenSC\\pkcs11\\opensc-pkcs11.dll`
     - `pki.blockingMode=false`
   - **Linux**:
     - `pki.vendorModule=/usr/local/lib/opensc-pkcs11.so`
     - `pki.blockingMode=false`

- Mac OS X:
  - `pki.vendorModule=/Library/OpenSC/lib/opensc-pkcs11.so`
  - `pki.blockingMode=false`

5 Save the changes.

6 Perform one of following based on the operating system:

- Microsoft Windows: Open the Services app and restart the Device Service.
- Linux: Run the following commands:

```
sudo service deviceservice stop
sudo service deviceservice start
```

- Mac OS X: Run the following commands:

```
sudo launchctl unload /Library/LaunchDaemons/
com.netiq.deviceservice.plist
sudo launchctl load /Library/LaunchDaemons/
com.netiq.deviceservice.plist
```

IMPORTANT: While using OpenSC, the Generate a key pair mode is not supported for Yubikeys and allows to work with the certificates that are existing on the PKI token or smart card.

Configuring Gemalto Smart Card with Advanced Authentication

This section provides the configuration information of the following Gemalto smart cards:

- IDPrime .NET Smart cards
- SafeNet eToken 51x0

To configure the Advanced Authentication with Gemalto smart card, perform the following configuration tasks:

- "Installing the SafeNet Authentication Client 10" on page 32
- "Generating the Customized MSI file" on page 33
- "Configuring PKCS Path in the Device Service" on page 33

Installing the SafeNet Authentication Client 10

1 Download the SafeNet Authentication Client 10.

2 Navigate to the Customization Package folder and execute the SAC`CustomizationPackage-10.0.msi` file.

The SafeNet Authentication Client Customization Package Installation wizard is displayed.

3 Click Next.

4 Read and accept the license agreement.

5 Click Next.

6 Click Change to select a different destination folder or install the Customization Tool's into the default folder:

`C:\Program Files\SafeNet\Authentication\`

7 Click **Install**.

8 Click **Finish**.

Generating the Customized MSI file

1 Click **Start** and navigate to **Programs > SafeNet > SACAdmin > SAC Customization Tool**.

2 Select **Features to install** in the left pane.

3 Select **IDGo 800 Compatible Mode** from the list.

4 Click **Actions > Generate MSI**.

5 Specify the file name and save files in the preferred folder.

The generated msi files are as follows:

- `<file name>msi-x32-10.0`
- `<file name>msi-x64-10.0`

6 Install the msi file according to the bits of your operating system.

The Installation wizard is displayed.

7 Follow the installation steps and click **Finish**.

NOTE: Ensure that the file `IDPrimePKCS11.dll` is available in one of the following paths:

- `C:\Program Files (x86)\Gemalto\IDGo 800 PKCS#11`
- `C:\Program Files\Gemalto\IDGo 800 PKCS#11`

Configuring PKCS Path in the Device Service

1 Install NetIQ Advanced Authentication Device Service.

2 Navigate to `C:\ProgramData\NetIQ\Device Service\config.properties`.

3 Set the `pki.vendorModule` to the customized PKCS file path as follows:

`pki.vendorModule= C:\\Program Files (x86)\\Gemalto\\IDGo 800 PKCS#11\\IDPrimePKCS11.dll`.

NOTE: Do not use a 64 bit library file (`IDPrimePKCS1164.dll`).

---

4 Save and Restart Device Service.

NOTE: If you have SafeNet Authentication Client (SAC) version v8.x, set the `pki.vendorModule` to `auto`. The SAC uses `eToken.dll` library for IDPrime cards.

Performing Bulk Replacement of Configuration File

With the Microsoft Group Policy Management Console (GPMC), you can update or customize the parameters of the configuration file in multiple machines of a domain by replacing the configuration file. To replace the configuration file in all machines within a domain, perform the following steps:

1. Create a configuration file `config.properties` with the preferred parameters.
2. Copy this configuration file to a network folder.
3. Open **Group Policy Management** console.
4. Right-click the domain name and select **Create GPO in this domain, and Link it here.**
5. Specify a name for the **Group Policy Object** and click **OK**.

   You can use the name to update the configuration file.
6. Right-click the created GPO and click **Edit**.
7. Click **Computer Configuration > Preferences > Windows Settings**.
8. Right-click **Files** and select **New > File**.
9. Change **Action** to **Replace**.
10. In **Source file(s)** specify the path of the configuration file located on the network folder.
11. In **Destination File**, specify the path: `C:\ProgramData\NetIQ\Device Service\config.properties.`
12. Clear all the **Attributes** options.
13. Click **OK**.
14. Create a group in the domain with computers on which you want to replace the Device Service configuration file.
15. In the **Security Filtering** section of the **Group Policy Management** console, for the used GPO remove the **Authenticated Users**.
16. Click **Add** and select the created group.
17. Click **Delegation**.
18. Right-click the added group and select **Edit settings**, **delete**, **modify security.**
19. Run **gpupdate /force** on the computer where you will replace the configuration file or wait till the policy is applied automatically.

Configuring the Security Settings

To secure the user information that is stored in the digital certificates of PKI authenticator and other authentication methods supported by Device Service, you can control and process the HTTPS requests from a preferred domain. With this approach, you can grant the access to secured

resources only for the requests from the Advanced Authentication server and deny access for any requests from an unidentified domain. With the security settings, you can also avoid the cross-origin HTTPS request and click-jacking vulnerabilities.

To configure the security settings for the Device Service, perform the following steps:

1. Open the configuration file based on the operating system:
   - **Microsoft Windows**: `C:\ProgramData\NetIQ\Device Service\config.properties`
   - **Linux**: `/opt/NetIQ/Device Service/config.properties`
   - **Apple Mac OS X**: For 6.3 Service Pack 1 and newer versions, open `/Library/Application\ Support/NetIQ/DeviceService.app/Contents/Resources/config.properties`. For prior versions, open `/Library/LaunchDaemons/NetIQ/Device Service/config.properties`.
2. Specify the following parameters:
   - `host.accessControlOrigin=<origin>`.

     Where, `<origin>` is secured domain. Default value is asterisk symbol (*). With the default value, the HTTPS request from any origin can access the secured resource. This may be vulnerable and cause issues to the secured resource.

     For example, set the parameter as `host.accessControlOrigin=https://myexample.company.com` then the HTTPS requests from specified origin can only access the digital certificates list.
   - `host.xFrameOptions=allow-from <domain URL>`.

     X-Frame-Options header that you can set using `host.xFrameoptions` parameter are not supported on the browsers, Google Chrome and Safari.

     Where, `<origin>` is secured domain.

     For example, `host.xFrameOptions=allow-from https://sample.company.com`. This allows the PKI related pages to be loaded in a frame only on the specified origin or domain.
   - `host.contentSecurityPolicy=frame-ancestors 'none'`

     To prevent embedding a page using `<frame>` or `<iframe>`, you can set the `frame-ancestor` to none (empty). This parameter prevents Cross Site Scripting (XSS) vulnerabilities.
3. Save the changes.
4. Restart the Device Service.

4 Uninstalling Device Service

To uninstall the Device Service based on the platform, see the following sections:

Uninstalling Device Service on Windows

You can uninstall Device Service in one of the following ways:

- Uninstalling Device Service through Setup Wizard
- Uninstalling Device Service through Control Panel

Uninstalling Device Service through Setup Wizard

1. Run `naaf-deviceservice-x86-release-<version>.msi`.
2. Click **Next**.
3. Select **Remove** and click **Next**.
4. Click **Remove** to confirm the deletion.

Uninstalling Device Service through Control Panel

To uninstall Device Service through Control Panel, perform one of the following according to your operating system:

- Microsoft Windows 7
- Microsoft Windows 8.1
- Microsoft Windows 10

Microsoft Windows 7

1. In the **Start** menu, select **Control panel** and double-click **Programs and Features**.
2. Select **NetIQ Device Service** and click **Uninstall**.

Microsoft Windows 8.1

1. In the **Search** menu, select **Apps > Control Panel > Programs > Programs and Features.**
2. Select **NetIQ Device Service** and click **Uninstall**.

Microsoft Windows 10

1. Right-click **Start** and select **Control Panel > Programs and Features**.
2. Select **NetIQ Device Service** and click **Uninstall**.

Uninstalling Device Service on Linux

You can uninstall Device Service on Linux, based on your Linux distribution:

- "Uninstalling Device Service on Ubuntu and Debian (deb package)" on page 38
- "Uninstalling Device Service on openSUSE, CentOS, RHEL, and Fedora" on page 38

Uninstalling Device Service on Ubuntu and Debian (deb package)

Run the following command to remove Device Service:

```
sudo dpkg --purge naaf-deviceservice-<version>.x86_64
```

Uninstalling Device Service on openSUSE, CentOS, RHEL, and Fedora

Run the following command to remove Device Service:

```
rpm -e naaf-deviceservice-<version>.x86_64
```

Uninstalling Device Service on Mac

Following are the ways to uninstall Device Service in Mac:

- "Uninstalling Device Service with dmg File" on page 38
- "Uninstalling Device Service without dmg File" on page 39

Uninstalling Device Service with dmg File

Perform the following steps to uninstall Device Service:

1. Double click the file `naaf-deviceservice-macos-release-<version>.dmg`.
   The `naaf-deviceservice.pkg` and `uninstall` files are displayed.
2. Click the `uninstall` file.
3. Specify the local administrator credentials.

Uninstalling Device Service without dmg File

Perform the following steps to uninstall Device Service:

1. Navigate to `/Library/Application Support/NetIQ`.
2. Click the `uninstall` file.
3. Specify the local administrator credentials.

5 Troubleshooting

This chapter contains the following sections on troubleshooting:

Debugging Logs

This section describes procedure to collect the logs for Device Service on the following platforms:

Debugging Logs on Linux

On Linux, to enable the logs for the Device Service, perform the following steps:

1. Create a text file `config.properties` file in the `/opt/NetIQ/Logging/` path.
2. Add a string to the file: `logEnabled=True` that ends with a line break.
3. Save the changes.
4. Create a folder `Logs` in the `/opt/NetIQ/Logging/` path.
5. Run the following command in the terminal to stop the service:

   ```
   sudo service deviceservice stop
   ```
6. Run the following command to start the service:

   ```
   sudo service deviceservice start
   ```

The generated logs are stored in the `/opt/NetIQ/Logging/Logs` path.

Debugging Logs on Mac OS

On Mac OS, you can collect the logs for Advanced Authentication Mac OS X Client and Device Service in one of the following ways:

- “Using the Diagnostic Tool” on page 42
- “Manual” on page 42

NOTE: You can find the Diagnostic Tool component in the Advanced Authentication appliance distributive package.

Using the Diagnostic Tool

To collect the logs using the Diagnostic tool, perform the following steps:

1. Run the file `DiagTool.app`.
2. Click **Enable**.
3. Restart your system.
4. Reproduce the issue.
5. Run the file `DiagTool.app`.
6. Click **Save** in the **Debug logs** tab.

   The logs file is saved in the `logs-year-month-date-hour:minute:seconds.zip` format in the `/tmp` directory.

   For example, logs file is saved as `logs-2017-10-23-15:30:20.zip`.
7. Click **Save**.

You can perform the following actions in the **Debug logs** tab:

- **Disable** to disable the logging.
- **Refresh** to update the logs list.
- **Open** to open any specific log.
- **Clear All** to delete the existing logs.

Manual

1. Create a text file `config.properties` in the directory `/Library/Logs/NetIQ/`.
2. Add a string to the file `logEnabled=True` that ends with a line break.
3. Create a directory named `Logs` in the path `/Library/Logs/NetIQ/`.
4. Restart the system.
5. Reproduce the issue.
6. Compress the logs located in the path `/Library/Logs/NetIQ/Logs/` into a zip file.
7. Change `logEnabled=True` to `logEnabled=False` in the file `/Library/Logs/NetIQ/config.properties`.

Debugging Logs on Windows

On Windows, you can collect the logs for Advanced Authentication Windows Client and Device Service in one of the following ways:

- “Using a Diagnostic Tool” on page 43
- “Manual” on page 43

NOTE: You can find the Diagnostic Tool component in the Advanced Authentication appliance distributive package.

Using a Diagnostic Tool

Before you use the Diagnositic tool, ensure that the following requirements are met as prerequisites:

- Microsoft .NET Framework 3.5 installed
- The `DiagTool.exe` file is available with the following files in the same directory:
  - `DiagTool.exe.config`
  - `Ionic.Zip.dll`
  - `JHSoftware.DNSClient.dll`

To collect the logs using the Diagnostic tool, perform the following steps:

1. Run `DiagTool.exe`.
2. Click **Clear All** (if applicable) in the **Debug logs** tab.
3. Click **Enable**.
4. Restart the Windows system.
5. Reproduce your problem.
6. Run `DiagTool.exe`.
7. Click **Save** logs in the **Debug logs** tab.
8. Specify a file name and path.
9. Click **Save** to save the logs.
10. Click **Disable** to disable the logging.
11. Click **Clear All**.

Manual

If you do not have the Diagnostic tool, you can collect the logs using the following steps:

1. Create a text file `C:\ProgramData\NetIQ\Logging\config.properties`.
2. Add a string to the file: `logEnabled=True` that ends by a line break.
3. Create a directory: `C:\ProgramData\NetIQ\Logging\Logs\`.
4. Restart the system.
5. Reproduce your problem.

6. Compress the logs located in the path `C:\ProgramData\NetIQ\Logging\Logs\` into a zip package.
7. Change `logEnabled=True` to `logEnabled=False` in the file `C:\ProgramData\NetIQ\Logging\config.properties`.

Generic Issues

Issue: After users install a new browser and try to enroll or test a method, an error message `Service is not available` is displayed. This issue may occur for the services: Bluetooth, Card, Fingerprint, PKI, and FIDO U2F.

Reason: The Device Service sets the certificates aside during installation. As the browser is installed after the Device Service, the required certificates are inaccessible to the browser.

Workaround: Open a browser and access one of the following URLs based on the method to apply the appropriate certificate:

- **Buletooth**: `https://127.0.0.1:8440/api/v1/bluetooth/getdevices`
- **Card**: `https://127.0.0.1:8440/api/v1/card/getmessage?nowait`
- **Fingerprint**: `https://127.0.0.1:8442/api/v1/fingerprint/capture`
- **PKI**: `https://127.0.0.1:8440/api/v1/pki/getmessage?nowait`
- **FIDO U2F**: `https://127.0.0.1:8441/api/v1/fidou2f/abort`

Card Related Issues

You can browse the following URL to troubleshoot the Card related issues:

`https://127.0.0.1:8440/api/v1/card/getmessage?nowaitTo`

The response is displayed in the following format:

```
{
result: [<status>],
cardid: <card id>,
readerid: <reader id>
}
```

The following are the different status that are displayed as response for the Card service:

- `NO_READER`: Indicates that the card service is unable to detect the connected card reader.
- `READER_ON`: Indicates that the card service detected the connected card reader.
- `NO_CARD`: Indicates that there is no card on the reader.
- `CARD_ON`: Indicates that a card is presented to the reader.

NOTE: The `cardid` parameter is used only with the `CARD_ON` and `NO_CARD` statuses.

RF Ideas does not work in Mac OS Catalina

1. Open the configuration file `/Library/Application\ Support/NetIQ/DeviceService.app/Contents/Resources/config.properties`.
2. Set the parameter `card.rfideasEnabled:true`
3. Navigate to `/Library/Application Support/NetIQ`.
4. Click **DeviceServiceTool.app**.

   You are prompted with a message `"DeviceServiceTool" would like to receive keystrokes from any application`. Click **Open System Preferences** and grand access to this application in Security & Privacy - Privacy preferences.
5. After adding the Device Service to Input Monitoring, run the following commands to restart the Device Service.
   1. `sudo launchctl unload /Library/LaunchDaemons/com.netiq.deviceservice.plist`
   2. `sudo launchctl load /Library/LaunchDaemons/com.netiq.deviceservice.plist`

FIDO U2F Related Issues

You can browse the following URL to troubleshoot the FIDO U2F related issues:

`https://127.0.0.1:8441/api/v1/fidou2f/abort`

With the FIDO U2F token connected, the service returns following response:

```
{ "result":"ok" }
```

Fingerprint Related Issues

You can browse the following URL and place your finger on the reader to troubleshoot the fingerprint related issues:

`https://127.0.0.1:8442/api/v1/fingerprint/capture`

The service returns the response in the following format:

`{"BitsPerPixel":x,"BytesPerLine":xxx,"Dpi":xxx,"Height":xxx,"Image":"<fingerprintdata>","Width":xxx,"captureStatus":"Ok"}`.

For example:
`{"BitsPerPixel":8,"BytesPerLine":256,"Dpi":508,"Height":360,"Image":"<fingerprintdata>","Width":256,"captureStatus":"Ok"}`.

The following table describes the different parameters of the response:

Parameter	Description
captureStatus	Indicates status of capture. Possible values are: ◆ Ok ◆ Timeout ◆ Error ◆ NoReader
Width, Height	Fingerprint image size (width and height) in pixels.
Dpi	Dots per inch. This is used while matching the fingerprint.
BitsPerPixel	Bits per pixel. Typically 6 bits.
BytesPerLine	Bytes per line in image.
Image	Fingerprint image encoded using the Base-64 format in gray scale.

This section contains the following fingerprint issue:

- "Mismatch Error After Migrating from Advanced Authentication 5.6 to 6.0" on page 46
- "The Nitgen Device Hangs If Disconnected and Reconnected to a Workstation" on page 46

Mismatch Error After Migrating from Advanced Authentication 5.6 to 6.0

Issue: After migrating from Advanced Authentication 5.6 to 6.0, while authenticating with the SecuGen Hamster Pro 20 fingerprint reader an error message `Mismatch` is displayed on Windows operating system.

Workaround: Perform the following steps:

1. Open the configuration file `C:\ProgramData\NetIQ\Device Service\config.properties`.
2. Add the parameter `fingerprint.nbisEnabled=false`.

   The default value is `true`.
3. Save the changes.
4. Restart the Device Service.

The Nitgen Device Hangs If Disconnected and Reconnected to a Workstation

Issue: While enrolling or authenticating with the Nitgen eNBioScan-C1 device if you disconnect the device from a workstation and reconnect, the device hangs. Also, the workstation does not detect the device.

Workaround: Restart the workstation after you reconnect the device.

PKI Related Issues

You can browse the following URL to troubleshoot the PKI related issues:

`https://127.0.0.1:8440/api/v1/pki/getmessage?nowait`

The PKI service returns one of the following as response:

- `NO_READER` indicates no reader is connected.
- `NO_CARD` if a card is not presented.
- `CARD_ON` if a card is presented.

This section contains the following PKI issues:

- "Issue with YubiKey PKI" on page 47
- "Unable to Import a Certificate to the YubiKey Token" on page 47

Issue with YubiKey PKI

Issue: When you connect the PKI token to your system and initiate enrollment on the Self-Service portal, if an error message `Unexpected service status: PLUGIN_NOT_INITTED` is displayed. This issue occurs due to the invalid dll path in the configuration file.

Workaround: Ensure valid path to the dll file is specified in the configuration file. You can search for `opensc-pkcs11.dll` or `libykcs11-1.dll` in the C drive and specify the full path using \\ in place of \.

You can plug the Yubikey token to your system and navigate to the URL `https://127.0.0.1:8441/api/v1/pki/getmessage?nowait` to view the status of the token. The status must display as `CARD_ON`.

When you import the certificate to the token, navigate to the URL `https://127.0.0.1:8441/api/v1/pki/getcertificates` to view the certificate data.

If you are unable to enroll PKI using YubiKey token on the Self-Service portal then try to export the logs to investigate the issue.

Unable to Import a Certificate to the YubiKey Token

Issue: When you try to import certificate to the YubiKey token using the yubico-piv-tool, an error message `Failed authentication with the application` is displayed.

Workaround: You must reset PIN of the token in one of the following ways:

- Specify incorrect PIN three times consecutively and then reset the PIN (default PIN is 123456).
- Specify incorrect PUK code (default PUK code is 12345678) of the same length (for example, 87654321) then reset the PIN.

You can import the certificate to the YubiKey token after resetting the PIN.

Bluetooth Issues

To troubleshoot the Bluetooth related issues, navigate to the following URL:

`https://127.0.0.1:8440/api/v1/bluetooth/getdevices`

It returns a list of Bluetooth devices that are discovered.

For more information on Bluetooth, see Bluetooth Plug-in.

Microsoft Edge Related Issues

This section contains the issues related to Microsoft Edge browser.

Users Unable to Test the Enrolled Authenticators on the Microsoft Edge Browser

Issue: When users try to test an enrolled authenticator on the Self-Service portal, an error message `Card service is unavailable` is displayed.

Workaround: Perform the following steps to run Device Service on Microsoft Edge:

1. Open the command prompt with elevated privileges.
2. Run the following command:

   ```
   CheckNetIsolation LoopbackExempt -a -
   n="Microsoft.MicrosoftEdge_8wekyb3d8bbwe"
   ```
3. Open **about:flags** and ensure that the **Allow localhost loopback** option is enabled.

Firefox Related Issues

This section contains the issues related to the Firefox browser.

Users Unable to Enroll the Card and FIDO U2F Methods on the Firefox Browser

Issue: On macOS, if users try to test the Card and U2F methods on the Self-Service portal using the Firefox browser, an error message is displayed. This issue occurs when NetIQ certificate is not available in the browser.

Workaround: Perform the following steps to run remove broken profile:

1. Delete the Firefox user profile.
2. Update Firefox.
3. Open Firefox and recreate a profile.

6 Developer Information

The Device Service supports the open ports 8440, 8441, and 8442. It is recommended to use port 8440 as the other ports may be deprecated in the upcoming releases.

This chapter contains the developer information of the following plug-ins:

Card Plug-in

You can browse the following URL to check the Card service:

```
https://127.0.0.1:8440/api/v1/card/getmessage?nowait
```

The response is displayed in the following format:

```
{
result: [<status>],
cardid: <card id>,
readerid: <reader id>
}
```

The following table describes the different status that the Card service displays as a response.

Status	Description
NO_READER	The Card service has not detected the connected card reader
READER_ON	The Card service has detected the connected card reader
NO_CARD	There is no card on the reader
CARD_ON	A card is presented to the reader

NOTE: The `cardid` parameter is used only with the CARD_ON and NO_CARD statuses.

The following table lists the GET methods and the respective response that the Card service returns.

Method	Response
`https://127.0.0.1:8440/api/v1/card/getmessage?nowait`	Displays the current status of the reader and card instantly. Possible status values are: ◆ NO_READER ◆ NO_CARD ◆ CARD_ON
`https://127.0.0.1:8440/api/v1/card/getmessage?wait`	Waits for the next action. For example, tapping or removal of a card from the reader. **NOTE:** If you disconnect the reader with a card placed on the reader, two messages NO_CARD and NO_READER are displayed. But the first one will be caught with getmessage?wait. When you connect a reader with a card on, two events READER_ON and CARD_ON take place. As a result, READER_ON is displayed as response.
`https://127.0.0.1:8440/api/v1/card/getreaderon?nowait`	Displays the current status of reader. Possible status values are: ◆ READER_ON ◆ NO_READER
`https://127.0.0.1:8440/api/v1/card/getreaderon?wait`	Displays READER_ON if the reader is connected or waits till you connect the reader.
`https://127.0.0.1:8440/api/v1/card/getcardon?nowait`	Displays the current status of card. Possible status values: ◆ NO_READER ◆ NO_CARD ◆ CARD_ON
`https://127.0.0.1:8440/api/v1/card/getcardon?wait`	Displays NO_READER if a reader is not connected or waits till a card is presented on the reader. **NOTE:** If a card is present on the reader, the service waits for the next tap of the card.
`https://127.0.0.1:8440/api/v1/card/getcardoff?nowait&cardid=<cardid>`	Possible status values: ◆ NO_READER ◆ NO_CARD ◆ CARD_ON Use the `cardid` parameter to make the service wait when a specific card is removed.

Method	Response
`https://127.0.0.1:8440/api/v1/card/getcardoff?wait`	Possible status values: ◆ `NO_READER` ◆ `NO_CARD` If a card is present on the reader, the service waits till the card is removed from the reader.
`https://127.0.0.1:8440/api/abort?cancel-cookie=xxx`	All the `wait` methods support `cancel-cookie=xxx` parameter. For example, `https://127.0.0.1:8440/api/v1/card/getmessage?wait&cancel-cookie=xxx`. If you call abort with the cancel-cookie, all the waiting methods with the specified cookie are terminated.

FIDO U2F Plug-in

You can browse the following URL to check the FIDO U2F service:

`https://127.0.0.1:8441/api/v1/fidou2f/abort`

When a FIDO U2F token is connected to the system, the service returns the following response:

```
{ "result":"ok" }
```

Methods

The following table lists the POST and GET methods and the respective response that the FIDO U2F service returns.

Method	Syntax	Description	Response
`sign`	`https://127.0.0.1:8441/api/v1/fidou2f/sign`	This POST method obtains an identity assertion from the connected U2F token and performs the authentication	`{ "signRequests": [ {"challenge":"tRiTY3C8YerfmH6IIlfoCZjs5CMkKUWDrNhS7v5gCPQ", "version":"U2F_V2, "keyHandle":"knQD88Ue6ZT6tyutHr8ipZaiTRV2uT9qzwGqWjYo5HCwAiV5z2kc1vr08tWbdOLQ4S-ODg09vpp62P6owh4qmQ", "appId":"https://demo.yubico.com" } ] }`

Method	Syntax	Description	Response
register	https://127.0.0.1:8441/api/v1/fidou2f/register	This POST method registers a U2F token for a user account	{ "registerRequests": [{"challenge":"tRiTY3C8YerfmH6IIlfoCZjs5CMkKUWDrNhS7v5gCPQ", "version":"U2F_V2, "appId":"https://demo.yubico.com" }], "signRequests":[] } signRequest can be empty, or contain serial for the key handle validation { "challenge":"tRiTY3C8YerfmH6IIlfoCZjs5CMkKUWDrNhS7v5gCPQ", "version":"U2F_V2, "keyHandle":"knQD88Ue6ZT6tyutHr8ipZaiTRV2uT9qzwGqWjYo5HCwAiV5z2kc1vr08tWbdOLQ4S-ODg09vpp62P6owh4qmQ", "appId":"https://demo.yubico.com" }
abort	https://127.0.0.1:8441/api/v1/fidou2f/abort	This GET method terminates all the pending operations	{ "result":"ok" }

In case, if there is an issue with the token or configuration, error is displayed in the following format:

```
{ "errorCode"=1, "errorMessage"="Error Text"}
```

where:

- errorCode is an integer indicating the general error that occurred.
- errorMessage is additional text that provides details on the error.

The following table lists all the error codes of FIDO U2F service with description.

Error Code	Possible Cause
1	Token is not connected. Error message `Please connect a U2F token`.
2	Indicates bad request and the request cannot be processed. The navigated URL does not match with app ID or HTTPS is not prefixed to the URL.
3	Indicates configuration is not supported.
4	Indicates the connected token is not eligible for this request or token is already registered. To enable the registration process, specify `signRequests` in the body of register request.
5	Indicates timeout and no response from the token because the user did not touch the token within the given time frame.

Fingerprint Plug-in

You can navigate to the following URL to check the WBF Capture Service and place the finger on the reader while the URL is loading:

```
https://127.0.0.1:8442/api/v1/fingerprint/capture
```

The service returns the response in the following format:

```
{"BitsPerPixel":x,"BytesPerLine":xxx,"Dpi":xxx,"Height":xxx,"Image":"<fingerprintdata>","Width":xxx,"captureStatus":"Ok"}.
```

For example:

```
{"BitsPerPixel":8,"BytesPerLine":256,"Dpi":508,"Height":360,"Image":"<fingerprintdata>","Width":256,"captureStatus":"Ok"}.
```

The following table describes the different parameters of the response.

Parameter	Description
captureStatus	Indicates status of capture. Possible values are: ◆ Ok ◆ Timeout ◆ Error ◆ NoReader
Width, **Height**	Fingerprint image size (width and height) in pixels.
Dpi	Dots per inch. This is used while matching the fingerprint.
BitsPerPixel	Bits per pixel. Typically 6 bits.
BytesPerLine	Bytes per line in image.
Image	Fingerprint image encoded using the Base-64 format in gray scale.

You can navigate to the following URL to check the multiple fingerprint reader and place the correct fingers on the reader while the URL is loading:

```
https://127.0.0.1:8442/api/vi/fingerprint/capture?index=<index_value>
```

The `index_value` can be one of the following:

- `1` indicates four fingers of the left hand.
- `2` indicates four fingers of the right hand.
- `3` indicates two thumbs.

The service returns the response in the following format:

```
{"Finger":x,"Image":{"BitsPerPixel":x,"BytesPerLine":xxx,"Dpi":xxx,"Height
":xxx,"Image":"<fingerprintdata>","Width":xxx,"captureStatus":"Ok"}}
```

For example:

```
{"Finger":1,"Image":{"BitsPerPixel":8,"BytesPerLine":256,"Dpi":508,"Height
":360,"Image":"<fingerprintdata>","Width":256,"captureStatus":"Ok"}}
```

where the `finger` represents the finger ID. Possible values are:

- `1` for the right thumb.
- `2` for the left thumb.
- `3` for the right index.
- `4` for the left index.
- `5` for the right middle.
- `6` for the left middle.
- `7` for the right ring.
- `8` for the left ring.
- `9` for the right little.
- `10` for the left little.

PKI Plug-in

The following table lists all the parameters that the PKI plug-in supports.

Parameter	Description
`pki.vendorModule=<library-file-name>.dll`	To set the PKCS#11 implementation library that the vendor module requires.
`pki.hashMethod: SHA256`	The default value is SHA256 and you can specify this value, if a parameter is not presented. The following methods are also supported: SHA224, SHA384, SHA512. To set the methods, ensure that the PKCS#11 module supports the required hash method.
`pki.padding: PKCS#1`	The default value is PKCS#1 and you can specify this value, if a parameter is not presented.The following options are also supported: **PSS, OAEP**.

Parameter	Description
`pki.modulusBits: 2048`	The default value is 2048 bit. For example, eToken PRO 32k does not support it so you need to set 1024 to use it.
`pki.blockingMode: true`	Detects and monitors the token connected to the system. It is set to `true` by default. OpenSC does not support the 'waiting for card' mechanism and it requires to change the option to `False`. Most of the vendors module work appropriately in the default mode.

PKI plug-in uses the simulator API for a card or token detection and POST methods.

POST Methods

The following table lists the different POST methods of PKI service and the respective response that the service returns.

Method	Syntax	Description	Response
`getcertificates`	`https://127.0.0.1:8440/api/v1/pki/getcertificates`	Retrieves all certificates from the connected token.	`{ "readerid"=0, "certificates" : [ { "keypairid":"9beb","certificate":"30820371308202daa00....0b90d7290a1a76b0450264dd536d2cb057230f8dbfa8cfda05" } ] }` where: ◆ `keypairid` indicates ID of the key pair in the certificate. Save this ID for future logon operations. ◆ `certificate` indicates certificate value in DER format.

Method	Syntax	Description	Response

generatekeypair	https://127.0.0.1:8440/api/v1/pki/generatekeypair- POST method, Request Body: {"pin":"your_pin"}
	
	Replace your_pin with actual token PIN or leave it empty if there is no PIN.
		Generate a Public Key Infrastructure (PKI) public and private key pair for a local digital certificate.	{
			
			"readerid"=your_reader_id,
			"keypairid":"6f4712e554544ac3",
			"modulus":"a1709fb049c35fdc6695193e9dd980c713c.....91daaa9d2604eeeaad73d13b1",
			"exponent":"010001"
			}
			
			where:
			
			◆ keypairid indicates ID of the key pair in the certificate. Save this ID for future logon operations.
			
			◆ modulus
			
			◆ exponent

signchallenge	https://127.0.0.1:8440/api/v1/pki/signchallenge - POST method, Request Body: {"challenge":"3128", "pin":"your_pin", "keypairid":"9beb"
	}
	
	where:
	
	◆ challenge is in hex-string format
	
	◆ pin is PIN of the token
	
	◆ keypairid is ID of keypair from token.
		Enables the PKI plug-in to sign the challenge from the authentication server. User is provided with an interface to specify PIN and keypair ID.	If the challenge is successful, signature of given challenge is returned as response.
			
			{
			
			"readerid"=your_reader_id, "hash":"SHA1",
			"padding":"PKCS#1",
			"signature":"58ad84f3a9b7244031aa55c0d0ad753b1a480ae709a37210d48....493130d7b11f128ea2be1fcc42d123bdb715a153974e992b16d022"
			}
			
			where:
			
			◆ hash indicates hash method that is used.
			
			◆ padding indicates the padding method that is used.
			
			◆ signature indicates signature for given challenge in hex format.

Method	Syntax	Description	Response
verifychallenge	https://127.0.0.1:8440/api/v1/pki/verifychallenge - POST method, Request Body {"challenge":"3128", "pin":"your_pin", "keypairid":"9beb", "signature":"58ad84f3a9b72....bdb715a153974e992b16d022" }	Verifies the PKI plug-in challenge from the authentication server. User is provided with an interface to specify PIN, keypair ID, and signature.	

if there is an issue with token or configuration, the above methods display error in the following format:

```
{ "result":"ERROR_ID"}
```

The following table lists all the error IDs for the POST methods of PKI service with description.

Error ID	Description
PLUGIN_NOT_INITTED	A vendor module or library is not present, invalid, or not specified
METHOD_NOT_FOUND	Method not found
NO_CARD	No token or card is presented. Use wait methods to get an event
JSON_PARSE_FAILED	Bad request
WRONG_PIN	Incorrect PIN
GET_PRIVATE_KEY_FAILED	Error while retrieving a private key from the token
OPERATION_FAILED	general operation failure

GET Methods

You can browse the following URL to check the PKI service:

```
https://127.0.0.1:8440/api/v1/pki/getmessage?nowait
```

The response is displayed in the following format:

```
{
result: [<status>],
cardid: <card id>,
readerid: <reader id>
}
```

The following table describes the different status that the PKI service displays as response.

Status	Description
NO_READER	The Card service has not detected the connected card reader or the reader is not connected to the system
READER_ON	The Card service has detected the connected card reader
NO_CARD	A card is not inserted in the reader
CARD_ON	A card is inserted in the reader

NOTE: The `cardid` parameter is used only with the CARD_ON and NO_CARD statuses.

The following table lists the different GET methods of the PKI service and the respective response that the service returns.

Method	Response
`https://127.0.0.1:8440/api/v1/pki/getmessage?nowait`	Displays the current status of the reader and card instantly. Possible status values are: ◆ NO_READER ◆ NO_CARD ◆ CARD_ON
`https://127.0.0.1:8440/api/v1/pki/getmessage?wait`	Waits for the next action. For example, insertion or removal of a card from the reader. **NOTE:** If you disconnect the reader with a card being inserted in reader, two messages NO_CARD and NO_READER are displayed. When you connect a reader with a card inserted, two events READER_ON and CARD_ON take place. As a result, READER_ON is displayed as a response.
`https://127.0.0.1:8440/api/v1/pki/getreaderon?nowait`	Displays the current status of reader. Possible status values are: ◆ READER_ON ◆ NO_READER
`https://127.0.0.1:8440/api/v1/pki/getreaderon?wait`	Displays READER_ON if the reader is connected or waits till you connect the reader.

Method	Response
`https://127.0.0.1:8440/api/v1/pki/getcardon?nowait`	Displays the current status of the card. Possible status values: ◆ NO_READER ◆ NO_CARD ◆ CARD_ON
`https://127.0.0.1:8440/api/v1/pki/getcardon?wait`	Displays NO_READER if a reader is not connected or waits till a card is inserted in the reader. **NOTE:** If a card is inserted in the reader, the service waits till the card is removed and inserted again.
`https://127.0.0.1:8440/api/v1/pki/getcardoff?nowait&cardid=<cardid>`	Possible status values: ◆ NO_READER ◆ NO_CARD ◆ CARD_ON Use the `cardid` parameter to make the service wait when a specific card is removed.
`https://127.0.0.1:8440/api/v1/card/getcardoff?wait`	Possible status values: ◆ NO_READER ◆ NO_CARD If a card is present on the reader, the service waits till the card removed from the reader.
`https://127.0.0.1:8440/api/abort?cancel-cookie=xxx`	All the `wait` methods support `cancel-cookie=xxx` parameter. For example, `https://127.0.0.1:8440/api/v1/pki/getmessage?wait&cancel-cookie=xxx`. If you call abort with the cancel-cookie, all the waiting methods with the specified cookie are terminated.

Bluetooth Plug-in

The following table lists all the methods that the Bluetooth plug-in supports.

Method	Syntax	Description	Response

getdevices	https://127.0.0.1:8440/api/v1/bluetooth/getdevices	This GET method either returns a JSON array of all discovered Bluetooth devices or an error code if Bluetooth is turned off.	<pre>{
			"devices":
			[
			{
			"name":"MagicKeyboard","address":"9cd746e1234","type":"peripheral","hash":"9b67e2d07088a1f0bd64bde8c44ab7cdc279463bd6d93735ab778afda79d0bde"
			},
			{
			"name":"MagicMouse","address":"1abcd22dafae","type":"peripheral","hash":"dbf75830268ab5516a0d658d28105761b6d6ec062a42317a84b3a82e8e4d643f"},
			{
			"name":"Lex'siPhone","address":"40cd0150cf58","type":"phone","hash":"ac904cc2e2626ca27eb7f4100166e0ae07957da89a5a3aa52f0a5d182b6ba42e"
			}
			]
			}</pre>
			where:
			◆ name indicates the Bluetooth device name.
			◆ address indicates address of the device
			◆ type indicates device type. The type can be one of the following:
			◆ computer
			◆ phone
			◆ lan_access
			◆ audio
			◆ peripheral
			◆ imaging
			◆ unclassified

Method	Syntax	Description	Response
detectdevice	https://127.0.0.1:8440/api/v1/bluetooth/detectdevice Request Body {"address":"[RSA encoded address]"} where RSA encoded address is address of Bluetooth device encoded with an RSA public key (from certificate) in the hex-string format.	This POST method is used to test the presence of device with its address	If the device is in range, the service returns: {"result":"CONNECTED","address":"40cd0150cf58"} if the device is not within the range or the Bluetooth is turned OFF on the device, the service returns: {"result":"DISCONNECTED"} Following are the other possible result values for this method: ◆ FAILED: Indicates general failure ◆ DECRYPT_FAILED: Indicates failure while decoding ◆ INVALID_ADDRESS: Indicates invalid address of the device ◆ hash: Indicates SHA256 hash of the address ◆ BLUETOOTH_DISABLED: Indicates Bluetooth is turned OFF.
getpublickey	https://127.0.0.1:8440/api/v1/bluetooth/getpublickey	This GET method returns the public certificate in the PEM format. The Bluetooth address is encoded with the public key in that certificate.	{"publicKey":"[PUBLIC_CERT]"} where: ◆ PUBLIC_CERT indicates the public certificate in the PEM format. ◆ publickey displays public key of the device in the following format: "-----BEGIN RSA PUBLIC KEY-----\n" "MIGHAoGBAKqGJxyB/ZgrTEsfqmMdE4GRwGH+XOioOa0EiQ8+HYcR8Pcg57j1Cc5k\n" "D1TrGNKpayWUWW7YEsXvfSpc5a5x9qwsEe06Iak5eP/PcGNLUViLwy2CN9oy5mSM\n" "Izpd607GNBUzEwWg0sIpm3FBEvtFFDxBb7PzE9W4hE//t0LQkGcTAgED\n" "-----END RSA PUBLIC KEY-----";